NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2026-32

September, 2026

Advancing Cybersecurity through Explainable AI

Jiahao Yu

Abstract

AI agents built on large language models and reinforcement learning are increasingly being applied to cybersecurity tasks such as vulnerability discovery, program repair, and security analysis. However, their deployment in real-world security workflows remains limited by two important barriers. First, security practitioners must be able to audit and trust the long, multi-step behaviors produced by these agents. Second, improving such agents is difficult because security data is often scarce, noisy, imbalanced, and rapidly shifting. This dissertation studies step-level explanations as a unifying mechanism for addressing both challenges: explanations can reveal which decisions determine an agent's behavior, and these same decisions can provide actionable signals for improving the agent.

The first part of this dissertation develops step-level explanation methods for AI agents in security applications. This dissertation begins with AIRS, which explains reinforcement-learning-based security agents by identifying the decision steps most responsible for an agent's outcome. AIRS shows that step-level explanations can make security agents more interpretable, but its reliance on structured state representations and linear explanatory models limits its applicability to more complex agent behaviors. To address this limitation, this dissertation further introduces StateMask, a trajectory-level explanation method that identifies pivotal decision steps by learning which actions can be masked without changing the final outcome. Applied to AI-generated vulnerability patches, StateMask highlights decisions that determine

whether a patch succeeds, fails, or succeeds through unsafe shortcuts such as modifying tests. A user study with 46 experienced developers and security researchers shows that the identified critical steps align with human reasoning in 89% of cases, suggesting that step-level explanations can substantially reduce the cognitive overhead of auditing AI-generated security outputs.

The second part of this dissertation shows that step-level explanations can also improve agent capability. This dissertation introduces GPO, a post-training framework that uses pivotal steps in reasoning trajectories to synthesize higher-quality training data. GPO estimates which steps have the largest effect on future success, resets the policy to those steps, and samples new rollouts that correct the model's earlier mistakes. This converts explanations from passive diagnostic artifacts into active optimization signals. In vulnerability patching tasks, models trained with GPO achieve performance competitive with substantially larger commercial models.

Together, these works argue that explanations are not merely post-hoc tools for making AI systems understandable. For cybersecurity agents, step-level explanations provide a practical interface between human trust and model improvement: they help practitioners inspect agent behavior, expose failure modes in long trajectories, and guide the synthesis of better training data. This dissertation presents explanation-centered methods as a path toward AI systems that are more trustworthy, more capable, and better suited to the complexity of real-world cybersecurity.

Keywords

Large Language Model, Cybersecurity, Machine Learning, Explainability

NORTHWESTERN UNIVERSITY

Advancing Cybersecurity through Explainable AI

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Jiahao Yu

EVANSTON, ILLINOIS

September 2026

© Copyright by Jiahao Yu, 2026.

All Rights Reserved

ABSTRACT

AI agents built on large language models and reinforcement learning are increasingly being applied to cybersecurity tasks such as vulnerability discovery, program repair, and security analysis. However, their deployment in real-world security workflows remains limited by two important barriers. First, security practitioners must be able to audit and trust the long, multi-step behaviors produced by these agents. Second, improving such agents is difficult because security data is often scarce, noisy, imbalanced, and rapidly shifting. This dissertation studies step-level explanations as a unifying mechanism for addressing both challenges: explanations can reveal which decisions determine an agent's behavior, and these same decisions can provide actionable signals for improving the agent.

The first part of this dissertation develops step-level explanation methods for AI agents in security applications. This dissertation begins with AIRS, which explains reinforcement-learning-based security agents by identifying the decision steps most responsible for an agent's outcome. AIRS shows that step-level explanations can make security agents more interpretable, but its reliance on structured state representations and linear explanatory models limits its applicability to more complex agent behaviors. To address this limitation, this dissertation further introduces STATEMASK, a trajectory-level explanation method that identifies pivotal decision steps by learning which actions can be masked without changing the final outcome. Applied to AI-generated vulnerability patches, STATEMASK highlights decisions that determine whether a patch succeeds, fails, or succeeds through unsafe shortcuts such as modifying tests. A user study with 46 experienced developers and security researchers shows that the identified critical steps align with human reasoning in 89% of cases, suggesting that step-level explanations can substantially reduce the

cognitive overhead of auditing AI-generated security outputs.

The second part of this dissertation shows that step-level explanations can also improve agent capability. This dissertation introduces GPO, a post-training framework that uses pivotal steps in reasoning trajectories to synthesize higher-quality training data. GPO estimates which steps have the largest effect on future success, resets the policy to those steps, and samples new rollouts that correct the model's earlier mistakes. This converts explanations from passive diagnostic artifacts into active optimization signals. In vulnerability patching tasks, models trained with GPO achieve performance competitive with substantially larger commercial models.

Together, these works argue that explanations are not merely post-hoc tools for making AI systems understandable. For cybersecurity agents, step-level explanations provide a practical interface between human trust and model improvement: they help practitioners inspect agent behavior, expose failure modes in long trajectories, and guide the synthesis of better training data. This dissertation presents explanation-centered methods as a path toward AI systems that are more trustworthy, more capable, and better suited to the complexity of real-world cybersecurity.

ACKNOWLEDGEMENTS

The Ph.D. journey is a long and often challenging one, and it was not something I could have imagined pursuing when I first began my undergraduate studies. I still remember attending a talk by a senior student in my department during my sophomore year. The talk was about why he decided to pursue a Ph.D. and what he had learned from the journey. I was deeply inspired by his talk that day and gradually decided to pursue a Ph.D. myself. I truly appreciate him for sharing his experience and inspiring me to take this path. I am also deeply grateful for the opportunity to conduct research at Northwestern University and for the support from many people along the way.

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Xinyu Xing, for his continuous support and guidance throughout my Ph.D. journey. When I was a new Ph.D. student, I was very fortunate to receive Professor Xing's hands-on guidance on how to conduct research, write papers, and present my work. I would also like to thank Professor Xing for giving me the freedom to explore my own research interests as I became more independent. Although we sometimes had different opinions on research directions, he still supported me and encouraged me to pursue my interests. Now, as I am on the job market and preparing for assistant professor positions, I look back on the days when I worked with Professor Xing with deep gratitude for his support and guidance. Without them, I would not be able to continue pursuing an academic career in the next stage after my Ph.D.

I am also deeply grateful to my committee members for their support and guidance. I would like to give special thanks to Professor Gang Wang, who helped me with my first academic paper, AIRS. I learned a great deal from him about academic paper writing. I would also like to thank Professor Yan Chen for his continuous support. In addition to serving as a committee member and as a reference for my job search, Professor Chen has given me tremendous support and has treated

me like his own student. I am also thankful to Professor Xiao Wang for serving as my committee member. He is a young but highly successful professor and is known as a top-tier researcher in the field of cryptography. I hope I will have more opportunities to work with him in the future.

During my Ph.D. journey, I have also had the opportunity to collaborate with many talented researchers. I would like to thank all my co-authors for their contributions to our research projects, especially Dr. Xian Wu and Dr. Zelei Cheng. We collaborated on many research projects together, and I truly enjoyed working with them. I am also very fortunate to have connected with Professor Kexin Pei, who has helped me greatly not only in my research but also in my job search.

Beyond my research, I am also grateful for the support of my friends at Northwestern University. Wenxuan Shi, Zheng Yu, Ziyi Guo, and Haozheng Luo are all good friends of mine, and I have enjoyed both working with them and spending time with them.

Finally, I would like to thank my family for their unconditional love and support. As a first-generation college student in my family, I know that they may not fully understand what I do in my Ph.D. research or why I chose to pursue a Ph.D. But they have always supported my decisions and encouraged me to pursue my dreams.

When I was writing this acknowledgements section, I found that I was not as excited as I had expected. I guess this is because I realized that I am still as enthusiastic about research as I was on the first day of my Ph.D. journey. I also know that I will continue to do research in the future and continue working with the amazing people mentioned above. Sometimes the end of a chapter is not as exciting because you know there are more amazing chapters to come. To me, continuing to read different chapters of the same book is more attractive than switching to a new book.

There are many more people I would like to thank, but I cannot list all of them here. Thank you all so much for your support. I will carry your kindness forward as I begin a new chapter in my career, with the same passion for research and the same kindness toward others.

THESIS COMMITTEE

Xinyu Xing
Northwestern University
Committee Chair

Gang Wang
University of Illinois Urbana-Champaign
Committee Member

Yan Chen
Northwestern University
Committee Member

Xiao Wang
Northwestern University
Committee Member

TABLE OF CONTENTS

Acknowledgements 5

List of Figures 14

List of Tables 19

Chapter 1: Introduction 22

Chapter 2: Related Work 26

2.1 AI for Cybersecurity and Sequential Decision-Making 26

2.2 Explainable AI for Deep Models 28

2.3 Explainable Deep Reinforcement Learning 30

2.3.1 Feature-Level Explanations 30

2.3.2 State- and Step-Level Explanations 31

2.3.3 Dataset- and Model-Level Explanations 32

2.4 Applications of Explanations 32

2.5 LLM Reasoning 34

2.6 LLM Post-Training 35

Chapter 3: AIRS: Explanation for Deep RL based Security Applications 37

3.1 Introduction 37

3.2 Background 40

3.2.1 Modeling Sequential Decision-Making 40

3.2.2 Learning DRL Policies 41

3.2.3 XRL Methods and Limitations 42

3.3 Design of AIRS 43

3.3.1 Problem Setup and Basic Idea 44

3.3.2 Approximation Model 44

3.3.3 Stability and Step Filtering 46

3.3.4 Algorithms 48

3.4 Evaluation 48

3.4.1 Baselines and Metrics 48

3.4.2 Fidelity 49

3.4.3 Stability and Efficiency 50

3.5 Using AIRS for Debugging 52

3.5.1 Agent Behavior Understanding 52

3.5.2 Policy Debugging 53

3.5.3 Application Debugging 54

3.6 Discussion 55

3.7 Summary . . . 56

Chapter 4: StateMask: Identifying Critical Decision Steps in Agent Trajectories 57

4.1 Introduction . . . 57

4.2 Design of StateMask . . . 60

4.2.1 Overview . . . 60

4.2.2 Technical Details . . . 64

4.3 Evaluation . . . 67

4.3.1 Experiment Setup . . . 68

4.3.2 Experiment Design . . . 68

4.3.3 Experiment Results . . . 69

4.3.4 Utility of Explanation . . . 71

4.3.5 Security Case Analysis: Vulnerability Patching . . . 73

4.4 Discussion . . . 76

4.5 Summary . . . 76

Chapter 5: GPO: Learning from Critical Steps to Improve LLM Reasoning 79

5.1 Introduction . . . 79

5.2 Preliminaries . . . 83

5.3 Method . . . 85

5.4 Theoretical Analysis . . . 88

5.4.1 Online policy gradient algorithm 88
5.4.2 Preference optimization 89
5.5 Experiments 90
5.5.1 Main Results 91
5.5.2 Ablation Study 93
5.5.3 Monte Carlo Simulation and Model Size Scaling 93
5.6 User Study 95
5.7 Discussion and Limitations 95
5.8 Conclusion 97

Chapter 6: Conclusion and Future Work 98

References 116

Appendix A: Supplementary Material for AIRS 116
A.1 Representative Applications 116
A.2 Additional Experimental Setting 117
A.3 Ablation Study 118
A.4 Exhaustive Search 119

Appendix B: Additional Material for StateMask 121
B.1 Proof of Theorem 1 121

B.2 Derivation of Eqn. (5) . . . 122
B.3 Derivation from Eqn. (6) to Eqn. (7) . . . 123
B.4 Our Training Algorithm . . . 125
B.4.1 Algorithm Detail . . . 125
B.4.2 Implementation and Hyperparameters . . . 127
B.5 Details of Evaluation . . . 129
B.5.1 Other Implementation Details . . . 129
B.5.2 Experiment Setup . . . 130
B.5.3 Additional Evaluation Results . . . 134
B.6 Additional Visualization Results . . . 137
B.7 Additional Results of Attacks . . . 142
B.8 Additional Results of Policy Patching . . . 142
B.9 Evaluation on Explaining Sub-optimal Agents . . . 143
B.9.1 Fidelity . . . 144
B.9.2 Visualizing the Explanation for Sub-optimal Agent Behaviors . . . 145
B.9.3 Patching Results of Sub-optimal Agents . . . 146
B.10 Evaluation on an Alternative Design . . . 147
B.11 Potential Social Impact . . . 148

Appendix C: Supplementary Material for GPO . . . 152

C.1 Additional Preference Optimization Objectives . . . 152

C.2 Additional Experiment Details . . . 154
C.2.1 Dataset Details . . . 154
C.2.2 Implementation Details . . . 155
C.2.3 Hyper-parameters for Experiments . . . 156
C.2.4 Running Environment . . . 156
C.2.5 Experimental Details for Long-Trajectory Reasoning . . . 157
C.3 PatchAgent Vulnerability Patching Results . . . 158
C.4 User Study . . . 159
C.4.1 User Study Setup . . . 159
C.4.2 User Study Questions and Results Format . . . 160
C.4.3 Overall Findings . . . 170
C.5 Broader Impact . . . 171
C.6 Ethics Considerations . . . 171
C.7 Theory . . . 172
C.7.1 Online policy gradient algorithm . . . 172
C.7.2 Preference optimization . . . 179

LIST OF FIGURES

3.1 **Example of feature-level explanation and step-level explanation.** Two types of explanations for malware mutation using DRL on a call graph. Feature-level explanation (left) identifies important features for the agent's decision at a given moment. Step-level explanation (right) identifies the most critical step(s) that lead to the final evasion outcome. 38

3.2 **Approximation model of AIRS.** The model takes a state sequence as input and predicts the corresponding final reward as output. The learned coefficient network provides step-level importance scores for individual episodes. 45

3.3 **Fidelity evaluation.** A higher RRD score indicates a more faithful explanation. The horizontal dotted line marks RRD=1, which corresponds to random selection. "Chain-Graph" and "Random-Graph" are two topology settings for lateral movement. 50

3.4 **Stability evaluation.** The figure reports the average normalized pairwise L_1 distance across 10 runs. Lower distance means more stable explanations. LR is the most stable because it gives a global explanation; AIRS is the most stable local explainer among the non-global model-based methods. 51

3.5 **Safe-driving explanation.** AIRS assigns high importance to the steps where the vehicle passes traffic barriers. These steps are naturally tied to the final reward because collision would both incur penalties and prevent successful arrival. 52

3.6 **Lateral-movement explanation.** AIRS highlights the final steps that complete the network takeover. Red nodes are compromised nodes, blue nodes are newly discovered nodes, solid arrows denote successful connections, and dashed arrows denote discovery actions. 53

3.7 **Original lateral-movement episode.** AIRS identifies an intermediate trap state where the agent repeatedly attacks already compromised nodes instead of exploring new nodes. 54

3.8 **Manipulated lateral-movement episode.** Replacing the action at the selected trap state changes the subsequent trajectory and allows the agent to take over the network. 55

4.1 **Illustration of the Pong game.** The DRL agent controls the yellow (right) paddle, and a non-RL rule-based opponent controls the white (left) paddle. (a) After the agent launches the ball toward the opponent, it has a few steps of freedom where random actions barely affect the outcome. (b) When the agent is about to touch the ball, a perturbed action is likely to change the final result. 61

4.2 **The StateMask framework.** A state mask agent is added to the environment alongside the fixed target agent. At each step, the state mask takes the current state s_t and outputs a binary action a_t^e that decides whether to keep the target agent's action or replace it with a random one, while preserving the target agent's expected total reward. 62

4.3 **Fidelity scores of explanations generated by selected methods in five games.** The x-axis represents the choice of K. "VALUE" stands for Value-max, and "LAZY" stands for LazyMDP. The bar reports the mean score, and the black line reports the standard deviation over three runs. 70

4.4 **Visualization of the identified critical time steps in Pong.** The DRL agent controls the yellow paddle, and a non-RL rule-based opponent controls the white paddle. StateMask highlights the frames where the ball approaches the yellow paddle, which are the steps where an incorrect action would likely cause the agent to miss the ball. 71

4.5 **Visualization of the identified critical steps in Connect 4.** The black player is the target agent. The states where the blue player takes actions are marked with a gray bar. StateMask highlights the first action and the final winning steps, matching the strategic structure of the game. 71

4.6 **User-study result for behavior understanding.** Participants preferred StateMask over baseline explanations in both near-optimal and sub-optimal agent settings. . . 72

4.7 **A critical step that reveals an unsafe patching shortcut.** StateMask flags the agent's decision to modify the tests as critical. The patch appears to pass the test suite, but the highlighted step shows that the agent removed security-relevant regression tests instead of fixing the underlying buffer-handling bug. 74

4.8 **A critical step that supports trusting a vulnerability fix.** StateMask flags the initialization decision as critical. Unlike the test-bypass case, the highlighted step is part of the root-cause fix: it reinitializes directory state before setting up fields for the custom tags. 78

5.1 **Overview of our method.** Given an initial trajectory generated by the policy π for a reasoning task, GPO segments the trajectory into steps. It then identifies the most critical step via the MC simulation and resets the policy to the critical step to generate a new trajectory. The new trajectory is then added to the dataset or online buffer. 85

5.2 **Ablation study results on BBH and MATH.** We compare the performance of the standard GPO method and Satori's strategy that randomly identifies the critical step in the trajectory. Each bar represents the average performance of 3 runs, with error bars indicating the standard deviation. 92

5.3 **Scaling behavior of GPO.** Performance impact of varying number of MC samples (left) and applying GPO across different model sizes (right) on MATH using DPO/KTO. 94

A.1 **AIRS ablation results.** The left plot varies the retained length of the filtered subsequence. The right plot varies the number of selected critical steps that are perturbed. 119

B.1 Visualization of the identified critical time steps in Pong game. The DRL agent controls the yellow paddle and a non-RL rule-based opponent controls the white paddle. 131

B.2 Visualization of the identified critical steps in the Connect 4 game. The black player is the target player. Note that the states where the blue player takes actions are marked with a gray bar. 132

B.3 The absolute policy value difference between the masked agent trained by StateMask and the target agent. Note that the darker solid lines represent the average absolute policy value difference across 3 random seeds. The lighter shadow indicates the corresponding variations between the maximal and minimal absolute policy value difference. 135

B.4 Fidelity scores for interpretations generated by three baseline methods and StateMask. A higher score implies higher fidelity. 136

B.5 Sensitivity and efficiency results. In the left and middle figures, we show the sensitivity results by varying the hyperparameter w in Pong and You-Shall-Not-Pass games. In the right figure, we report the mean and standard deviation of the explanation time per trajectory in four games: Pong, You-Shall-Not-Pass, Tic-Tac-Toe, and Breakthrough. 137

B.6 Visualization of the identified critical time steps in You-Shall-Not-Pass game. The DRL agent controls the red humanoid robot and the opponent controls the blue one. While the blue humanoid bot is trying to block the red one from reaching the goal, the red one is trying to get across the red line. 137

B.7 Visualization of the identified critical time steps in Kick-And-Defend game. The DRL agent controls the blue humanoid robot to kick the ball and the opponent controls the red one to defend the goal. 137

B.8 Visualization of the identified critical time steps in Cartpole game. The DRL agent controls the cart to move left or right to keep the pole standing (within a certain angle) as long as possible. 138

B.9 Visualization of the identified critical time steps in Pendulum game. The DRL agent controls the torque applied on the free end of the pendulum to swing it into an upright position, with its center of gravity right above the fixed point. 138

B.10 Visualization of the identified critical steps in StarCraft II. The green zerg player is the target agent and the red one is the opponent. The big circles represent the buildings while the small circles represent the units. In the first three steps, the agent collects the minerals and vespene gas. In the next three steps in this episode, the agent builds some structures. In the last two steps, the small circles representing the zerglings and roaches of the target agent are attacking the enemy base. 138

B.11 Visualization of the identified critical time steps in the Tic-Tac-Toe game. The DRL agent places the X mark and the opponent places the O alternately in one of the nine spaces in the grid. The states where the O mark is placed are colored in gray. 139

B.12 Visualization of the identified critical time steps in Breakthrough game. The DRL agent plays as the black player and the opponent plays as the white one. They play alternatively, with each player moving one piece per turn. 139

B.13 Visualization of the identified critical time steps in the DouDizhu game. The DRL agent plays the role of the landlord and two cooperative opponents play the role of the peasants. The landlord's goal is to empty his hand first, while the peasants aim to prevent this and empty one of their hands first. The states where the peasants take actions are marked with a gray bar. 141

B.14 One critical step in the DouDizhu game: The target agent (landlord) plays the Quad with Pairs to ensure that peasants cannot beat this hand. 142

B.15 Fidelity scores for interpretations generated by three baseline methods and StateMask for four sub-optimal agents. 144

B.16 Visualization of the identified critical time steps in the Pong game. The yellow paddle is controlled by the agent we aim to explain. 146

B.17 Visualization of the identified critical time steps in the Kick-And-Defend game. The kicker (blue agent) is the target agent for explanation. 146

B.18 Visualization of the identified critical time steps in Connect 4 game. The black player is the sub-optimal agent we aim to explain. 146

B.19 Visualization of the identified critical time steps in the Breakthrough game. The sub-optimal agent plays as the black player and the opponent plays as the white one. 147

B.20 Fidelity comparisons between StateMask and PPOMask across four selected games when the agent is sub-optimal. 148

LIST OF TABLES

3.1 **Efficiency evaluation.** Time is reported in seconds for training approximation models and generating explanations for 500 episodes. Value-function baselines do not require training an approximation model. LR has no per-episode generation cost after training because it produces a global explanation. 52

3.2 **Safe-driving retraining result.** The policy is retrained using the critical steps selected by AIRS. 53

3.3 **Selfish-mining patch result.** Patch result is the average reward change compared with the original policy. 54

3.4 **Reward-design debugging result.** The table reports attack success rate after modifying the lateral-movement reward design. 55

4.1 **Utility evaluation.** Numbers before brackets are means, and numbers in brackets are standard deviations. For adversarial attacks, more negative values indicate stronger attacks. For patching, larger positive values indicate better post-patching improvement. 73

5.1 **Comparison of GPO-enhanced methods against baselines.** The better results between GPO-enhanced and baseline methods are highlighted in bold. Each training result is the average of 3 runs with different random seeds. 90

A.1 **AIRS experimental setup.** The table reports the DRL algorithm, the approximation-model architecture, and the number of training epochs. FC denotes fully connected layers. 118

A.2 **Random-Graph fidelity detail.** AIRS has the highest reward difference and RRD in the random-graph lateral-movement setting. 118

A.3 **Exhaustive-search efficiency.** Time is reported in seconds for generating explanations for 500 episodes. The AIRS time includes both training and generation. . . 119

B.1 **Training hyperparameter settings of StateMask.** The numbers in the bracket after "CNN" represent the number of kernels in each layer. The numbers in the bracket after "MLP" are the hidden dimensions. 512×10 represents 10-layer MLP with hidden size 512. 129

B.2 Game information with the target agents to explain. "Discrete(X)" refers to X discrete actions available to choose from. "Vector(X)" refers to X-dimension continuous action space. 133

B.3 **Results of reward gap in all games.** Each experiment contains 500 runs. We repeat all experiments three times and report the mean and standard deviation of the discounted total rewards in the table below. 134

B.4 Performance drop under varying attack ratios in all games. Each experiment contains 500 runs. We repeat all experiments three times and report the average rewards with the corresponding standard deviations in the table below. 150

B.5 Results of EDGE's policy patching method with different explanation methods. Each experiment contains 500 runs. We repeat all experiments three times and report the average reward increase with the corresponding standard deviation in the table below. 150

B.6 Additional results of policy patching. Each experiment contains 500 runs. We repeat all experiments three times and report the average reward increase with the corresponding standard deviation in the table below. 151

B.7 Patching results of sub-optimal agents in four games. Each experiment contains 500 runs. We repeat all experiments three times and report the average reward increase with the corresponding standard deviation in the table below. 151

B.8 Results of relative errors in four selected games. Each experiment contains 500 runs. We repeat all experiments three times and report the mean and standard deviation of the discounted total rewards in the table below. 151

C.1 **Dataset Statistics**: The table presents the number of training and test samples for each dataset, along with the source of the dataset. 154

C.2 **Hyper-parameters for Different Datasets and Methods.** 157

C.3 Accuracy on a long-context reasoning subset of BBEH. 158

C.4 **PatchAgent vulnerability patching benchmark.** Results are reported as successful patch rates by vulnerability class and in aggregate. 159

Chapter 1

Introduction

Artificial intelligence, especially Large Language Models (LLMs), is rapidly changing the practice of cybersecurity. Modern security workflows increasingly rely on AI techniques for threat detection [1, 2], vulnerability mitigation [3], and blockchain security [4]. At the same time, large-scale public efforts have begun to demonstrate that AI systems can reason over complex software systems and assist with vulnerability discovery and remediation. For example, DARPA's AI Cyber Challenge evaluates autonomous systems that find and patch vulnerabilities in large real-world codebases [5, 6]. Model providers are also investing heavily in cyber-capable foundation models: OpenAI has released Trusted Access for Cyber and GPT-5.5-Cyber for cyber defenders [7, 8], while Anthropic's Project Glasswing reports the use of Claude Mythos Preview to support large-scale vulnerability discovery in critical open-source software [9]. These developments suggest a near future in which AI systems become important collaborators in security analysis, vulnerability triage, program repair, and cyber defense.

However, capability alone is not sufficient for cybersecurity. Security is a mission-critical domain: a wrong recommendation, an unsafe patch, or a compromised AI workflow can lead to data breaches, service disruption, financial loss, or new attack surfaces. This risk helps explain why practitioners remain cautious. Fortinet's 2025 Cybersecurity Skills Gap report shows that

although many organizations are optimistic about using AI in security operations, only 2% of respondents believe AI will fully replace cybersecurity professionals [10]. A concrete example is the 2025 CodeRabbit vulnerability disclosed by Kudelski Security. CodeRabbit uses AI to review code quality and security issues, but researchers showed that a malicious pull request could trigger remote code execution in CodeRabbit's production infrastructure, leak API tokens and secrets, and potentially provide write access to a large number of repositories [11]. This incident illustrates a broader concern: when AI systems are embedded into security-critical workflows, their failures may become security vulnerabilities themselves.

A central obstacle is trust. Cybersecurity professionals need not only a final answer, but also an auditable account of how that answer was produced. Many AI models, including deep reinforcement learning agents and LLM-based agents, behave as black boxes whose internal reasoning is difficult for humans to inspect [12]. This opacity is especially problematic for security tasks, where the difference between a correct solution and a dangerous shortcut can be subtle. For example, an AI patching agent may pass a test suite by fixing the underlying vulnerability, but it may also pass by weakening tests, suppressing errors, or introducing brittle behavior. Without explanations that identify which decisions matter and why, human auditors must manually inspect long trajectories of model actions, commands, generated code, and intermediate observations. This creates a significant cognitive burden and limits the safe deployment of autonomous cyber systems.

Explainable AI offers a promising path toward trustworthy AI for cybersecurity. Explanations can help security analysts understand why an AI agent made a decision, identify the steps most responsible for success or failure, and determine whether the agent relied on legitimate reasoning or unsafe shortcuts. In human-in-the-loop security workflows, such explanations can support efficient auditing: humans can focus their attention on the most consequential decisions rather than reviewing every action in a long trajectory. More broadly, explainability can expose blind spots in

AI systems, reveal failure modes that are hidden by aggregate performance metrics, and provide a foundation for improving the models themselves.

This dissertation studies how explainability can make AI-based cybersecurity systems both more trustworthy and more capable. The central thesis is that explanations should not be treated only as post-hoc descriptions for human consumption. Instead, explanations can identify pivotal states, decisions, and reasoning steps that determine whether an AI system succeeds or fails. Once these pivotal moments are identified, they can be used in two complementary ways: first, to help humans audit AI behavior, and second, to guide optimization methods that improve future model behavior.

This dissertation makes three main contributions.

Chapter 3 – AIRS: Interpretable Approximation Model for Deep Reinforcement Learning in Cybersecurity.

Cybersecurity applications often involve sequential decision-making, where an agent must take a series of actions before the final outcome is known. Such tasks can be naturally formulated as Markov Decision Processes and solved with deep reinforcement learning. However, DRL agents are difficult to interpret, which limits their use in security-critical settings. This dissertation first presents AIRS, an explanation framework for DRL-based security applications [13]. AIRS builds interpretable approximation models that explain which decisions contribute to an agent's behavior. By applying this approach to security tasks, AIRS shows that explanation can help discover the failure modes in the real-world scenarios, such as network lateral movement and blockchain mining.

Chapter 4 – StateMask: Identifying Critical Decision Steps in Agent Trajectories.

While AIRS improves the interpretability of DRL-based security agents, it still depends on struc-

tured state representations linear surrogate models. More complex AI systems may not have such structured representations, and using linear surrogates may miss important interactions between decisions. This dissertation therefore introduces StateMask [14]. Instead of fitting a surrogate model, StateMask identifies critical decision steps by masking out individual steps in an agent's trajectory and measuring the impact on the final outcome. By predicting the impact of masking each step, StateMask can identify which steps are most responsible for success or failure. This enables focused auditing of AI behavior for security tasks as well as other domains such as robotics and game playing.

Chapter 5 – GPO: Using Critical Steps to Improve LLM Reasoning.

The final contribution shows that explanations can be used not only to audit AI systems, but also to improve them. This dissertation presents GPO, a post-training framework that uses critical reasoning steps to synthesize better training data for LLMs [15]. GPO identifies pivotal steps in reasoning trajectories, resets the policy to those steps, samples alternative rollouts, and prioritizes learning from the resulting corrections. In this way, explanations become optimization signals: they reveal where the model's reasoning process should be repaired and provide targeted opportunities for improvement.

Together, these contributions develop a framework for explainable and improvable AI in cybersecurity. They show that step-level explanations can reduce the burden of human auditing, expose unsafe shortcuts in autonomous security workflows, and guide training methods that improve model capability. As AI systems become increasingly integrated into cyber defense, this dissertation argues that explainability will be essential not only for trust, but also for building more reliable and effective security agents.

Chapter 2

Related Work

This dissertation studies how explanations can make AI systems in security-critical settings more understandable, auditable, and improvable. The work sits at the intersection of AI for cybersecurity, explainable machine learning, explainable deep reinforcement learning, and post-training methods for large language models. This chapter reviews these areas of related work and situates the contributions of this dissertation within them. We start by discussing the unique challenges of AI for cybersecurity and the need for step-level explanations, then review general explainable AI methods and their limitations for sequential decision-making, and finally trace the development of explainable DRL and post-training methods that motivate the contributions of this dissertation.

2.1 AI for Cybersecurity and Sequential Decision-Making

Machine learning has become a core technique in modern cybersecurity. Security systems use learned models to detect malicious events [1], classify malware, identify phishing campaigns [2], analyze blockchain transactions [4], and assist vulnerability discovery and repair [3]. These applications benefit from the ability of machine learning models to recognize patterns in large, noisy, and fast-changing data. At the same time, cybersecurity differs from many conventional prediction

tasks. The defender or analyst often needs to make a sequence of decisions under uncertainty, and those decisions can change the state of the system being analyzed. A malware mutation tool may repeatedly edit a program until it evades a detector. A lateral movement agent may decide which hosts to scan, compromise, and pivot through. A blockchain mining strategy may decide when to publish or withhold blocks. A program repair agent may inspect files, run tests, edit code, and revise its patch. These are naturally sequential problems.

Deep reinforcement learning (DRL) provides one framework for modeling such tasks. In reinforcement learning, an agent observes the environment, chooses actions, receives rewards, and improves its policy through interaction. DRL extends this framework by representing policies or value functions with deep neural networks, enabling agents to operate in high-dimensional environments. Early successes in games and control tasks made DRL attractive for security and safety applications, where a defender or attacker often faces long-horizon decision-making. Prior work has applied reinforcement learning to malware mutation, autonomous penetration testing [16, 17], network attack simulation [18, 19], blockchain attack discovery [20], blockchain selfish mining [21, 22], and other security-relevant domains.

The sequential nature of these applications is a source of both power and risk. A DRL agent may discover strategies that are difficult to hand-engineer, but its behavior may also depend on brittle shortcuts that are difficult for humans to notice. The final reward of a successful trajectory does not reveal whether the agent followed a reliable path. A patch may pass tests for the wrong reason. A malware mutation policy may exploit artifacts of a detector rather than learning semantically meaningful transformations. A network agent may appear effective in simulation while relying on assumptions that would be unsafe in deployment. These concerns are especially important in security because the cost of a misleading recommendation can be high. Security practitioners need to know not only whether an AI system succeeded, but why it succeeded, where it failed, and which

intermediate choices were responsible for the outcome.

This requirement motivates step-level analysis. Traditional explanations often ask which input features influenced a prediction. In contrast, security agents often require explanations over which steps in a trajectory mattered most to the final outcome. A critical step is a moment in an agent's trajectory whose change would alter the final result. Identifying critical steps can help analysts understand why an agent succeeded or failed, focus their attention on the most consequential decisions, and determine whether the agent's reasoning was legitimate or relied on unsafe shortcuts. This trajectory-level perspective is central to the contributions of this dissertation and distinguishes it from much of the prior work in explainable AI, which often focuses on explaining which features have the most influence on the model's output for a single prediction.

The three main works in this dissertation follow this step-level perspective. AIRS explains DRL-based security applications by identifying critical steps with the linear approximation model [13]. StateMask generalizes this idea through a masking-based explanation policy that identifies steps whose perturbation would affect final reward [14]. GPO then moves from explanation to improvement by using critical reasoning steps as targets for post-training LLMs [15].

2.2 Explainable AI for Deep Models

Explainable AI (XAI) emerged in response to the opacity of modern machine learning systems. Deep neural networks can achieve strong predictive performance, but their internal representations and decision boundaries are often difficult for humans to inspect. In high-stakes domains, this opacity can become a deployment barrier.

One major line of XAI explains a model's decision by attributing importance to input features. Gradient-based methods compute how sensitive a model's output is to small changes in the

input, including early saliency maps [23], Grad-CAM [24], Integrated Gradients [25], and SmoothGrad [26]. These methods have been widely used in computer vision and have influenced security explanation work as well. For example, perturbation methods can identify image regions that are important to a classifier [27, 28], while local surrogate or decomposition methods can approximate the behavior of a complex model near a particular input [29]. In security, researchers have also evaluated whether explanations for malware classifiers are faithful or robust enough to be trusted by analysts [30, 31].

These feature-level explanations are useful when the object to be explained is a single prediction over a fixed input. They can show which bytes, tokens, graph nodes, pixels, or features influenced a model's output. However, they are less directly suited to explaining an agent's success or failure over a long interaction. A feature attribution map for one state can help explain why an agent chose one action at that state, but it does not answer why the full trajectory reached its final outcome. In a malware mutation task, the most important features for a single edit may not be the edit that caused evasion. In a network attack simulation, the local reason for scanning a host may not explain why the agent eventually compromised the network.

Another line of XAI designs models that are interpretable by construction. Decision trees, rule lists, linear models, sparse models, attention mechanisms, and structured policies can make the model's decision process easier to inspect. In reinforcement learning, this idea appears in policy extraction [32] and self-explainable agents, where a complex policy is distilled into a simpler policy or augmented with interpretable modules [33, 34]. These methods can be valuable, but they inevitably introduce a tradeoff. If the interpretable model replaces the original policy, it may sacrifice performance or fail to capture complex behavior.

Thus, explainability for security agents requires a careful choice of explanation target. The target should be meaningful to humans, faithful enough to support decisions, and efficient enough

to use on realistic trajectories. For the problems in this dissertation, we argue that the target is a moment in an agent's trajectory or a language model's reasoning process whose change would alter the final outcome.

2.3 Explainable Deep Reinforcement Learning

Existing work on explainable DRL can be organized by the level at which it explains the agent. Feature-level methods explain the relationship between observations and actions. State- or step-level methods explain which moments in a trajectory matter to final reward. Dataset-level methods ask which training examples or experiences influence learned behavior. Model-level methods try to make the policy itself interpretable. These categories are not mutually exclusive, but they clarify the different questions that explanation methods answer.

2.3.1 Feature-Level Explanations

Feature-level DRL explanations adapt techniques from supervised learning to policies. A policy network maps an observation to an action distribution, so one can ask which regions or features of the observation most influenced the selected action. Saliency methods for Atari agents [35], attention-augmented agents [33], and feature attribution approaches [36] follow this direction. Reward decomposition [37] and contrastive methods [38] also provide local insight by attributing behavior to components of the observation or reward signal.

These explanations can be intuitive, especially for visual domains. For example, they may show that an agent attends to a ball in a game, an obstacle in a driving scene, or a relevant object in a robot manipulation task. However, for security applications, we usually care more about the higher level question of why a trajectory succeeded or failed, as discussed below.

2.3.2 State- and Step-Level Explanations

State- and step-level explanations seek to identify the states or time steps that are most important to the agent's final reward or task outcome. Some early methods use value functions or related quantities to summarize important states. For instance, critical-state methods can select states where the agent's action choices have a large expected effect [39], and behavior summary methods can highlight representative or important moments for human inspection [40]. Lazy-MDPs learn when an agent can avoid acting, and the states where action remains necessary provide a form of interpretability [41].

Model-based trajectory explainers take a different approach. EDGE collects trajectories and fits an explanation model to associate states or actions with final rewards [42]. This provides a more direct link between trajectory components and outcomes, but it faces challenges in scalability, local fidelity, and adaptation to new trajectories. AIRS builds on this line of work for security applications. Rather than explaining only the current action, AIRS identifies the steps in an individual episode that contribute most to the final reward [13].

STATEMASK further develops step-level explanation by asking which time steps can be masked without changing the agent's performance [14]. Its key intuition is counterfactual: if randomizing the agent's action at a step leaves the final reward nearly unchanged, that step is less critical. Instead, if randomization significantly changes the outcome, the step is important. Instead of relying only on a pre-collected offline approximation, STATEMASK trains a mask policy that learns when to blind the target agent while preserving expected reward. This makes the explanation more closely tied to the effect of interventions on the agent's behavior. It also broadens the applicability of critical-step explanation beyond the original security settings of AIRS, including normal-form games, extensive-form games, and control environments.

2.3.3 Dataset- and Model-Level Explanations

Dataset-level explanations ask how training data shapes the learned policy. Influence functions estimate how upweighting or removing a training example would change the learned model or its predictions [43, 44]. Shapley-value methods instead assign game-theoretic credit to data points by measuring their marginal contribution across possible training subsets [45, 46]. These techniques estimate the importance of individual examples or components of examples to a model's behavior. In reinforcement learning, the data may include transitions, trajectories, demonstrations, or replay-buffer entries. These methods are useful when the goal is to diagnose training, remove harmful data, or understand why a policy learned a particular behavior. However, they answer a different question from AIRS and StateMask. Dataset-level explanations concern the origin of a policy in the training process; step-level explanations concern the causal role of moments within a deployed trajectory.

Model-level explanations attempt to make the policy itself transparent. Policy extraction [32, 47], decision-tree policies [48], attention modules, and structured reward models [49] can provide more inspectable behavior. These methods are valuable when the system can be designed around an interpretable policy. In many practical settings, however, the target agent is already trained, high-performing, and black-box. The explanation method must work with the existing agent rather than replacing it.

2.4 Applications of Explanations

Explanations are often presented as tools for trust, but trust is not simply a feeling of confidence. In security, appropriate trust means knowing when to rely on an AI system, when to question it, and where to look when it fails. A useful explanation should help a practitioner allocate attention.

Long agent trajectories can contain hundreds of observations, actions, commands, or generated reasoning steps. Reviewing every step is expensive and error-prone. Step-level explanations reduce this burden by directing attention to the moments most likely to determine the outcome.

One constructive use of explanations is debugging. If an agent fails, a critical-step explanation can reveal whether the failure came from a mistaken action, a flawed reward design, insufficient exploration, or a mismatch between the training environment and deployment assumptions. AIRS demonstrates this role in DRL-based security applications by using critical steps to expose policy defects and inappropriate reward designs [13]. StateMask extends the idea by showing that critical steps can guide policy patching and help users understand behavior across a wider range of environments [14]. In both cases, explanation becomes an operational tool for diagnosing and improving agents.

Another constructive use is human-in-the-loop auditing [13, 14, 15]. Security analysts often need to decide whether an AI-generated action sequence is acceptable [3, 11]. In autonomous cyber workflows, this may involve reviewing a patch or a chain of tool calls [5, 6, 3]. Explanations can help analysts focus on decisions that matter most. This does not remove the need for expert judgment, but it changes the review problem from exhaustive inspection to targeted inspection. The same principle applies to LLM reasoning: if a model solves or fails a task through a multi-step chain of thought, users and developers need to know which reasoning steps were pivotal [15].

Explanations can also be used for policy repair. Once critical steps are identified, one can explore alternatives at those points. This idea appears in explanation-guided policy refinement, where the system replays a failed trajectory to a critical state and uses successful alternatives to improve future behavior. StateMask demonstrates that critical-step identification can support such remediation, and GPO carries the idea into LLM post-training by resetting generation at pivotal reasoning steps and sampling alternative continuations [14, 15]. This creates a bridge from

interpretability to optimization: explanations identify where learning should focus.

However, explanations also introduce risks. If an explanation identifies the most important step in a security agent's success, an adversary may use the same information to attack the agent. Prior work has shown that explanations can reveal vulnerabilities in learned systems, including opportunities to perturb observations or disrupt critical decisions [42, 14]. In cybersecurity, this dual-use nature is unavoidable. The same explanation that helps a defender debug an agent may help an attacker identify a minimal intervention. This does not mean explanations should be avoided. Rather, it means explanation systems should be evaluated not only for clarity but also for fidelity, security implications, and deployment context.

2.5 LLM Reasoning

LLMs extend the trajectory perspective to natural language reasoning and tool use. A model solving a difficult problem often produces a sequence of intermediate steps before the final answer. Chain-of-thought prompting showed that asking models to reason step by step can improve performance on complex tasks [50]. Subsequent methods such as self-consistency [51], Tree of Thoughts [52], and Graph of Thoughts [53] explore multiple reasoning paths or structured deliberation to improve problem solving. Recent reasoning-focused models further emphasize long-form reasoning and reinforcement learning over multi-step trajectories [54, 55].

These developments make LLM reasoning look increasingly like sequential decision-making. The model observes a prompt, generates a reasoning step, conditions on the resulting prefix, and continues until it produces an answer. A single early mistake can derail the rest of the trajectory. Conversely, a pivotal correction can recover a failing trajectory. This structure resembles the DRL settings discussed earlier, but with important differences. The action space is language, the state is

a text prefix, and the meaning of a step is semantic rather than a single token. Directly applying DRL explainers to LLMs is therefore difficult, but the conceptual goal remains similar: identify which steps in a trajectory matter most to the final outcome.

2.6 LLM Post-Training

Post-training methods aim to improve LLM behavior after pretraining. Reinforcement learning from human feedback (RLHF) [56] trains models to follow human preferences, often by learning a reward model and optimizing the policy with algorithms such as PPO [57]. Preference optimization methods such as DPO provide a more direct offline alternative by training from preferred and rejected responses without an explicit online RL loop [58]. Related objectives such as KTO [59], ORPO [60], and SimPO [61] modify the preference-learning formulation to reduce cost, simplify training, or change the optimization signal. These methods have become central to aligning LLM outputs with desired behavior.

For reasoning tasks, post-training often relies on high-quality reasoning trajectories. Supervised fine-tuning can train on curated solutions, but expert annotation is expensive [62, 63, 64]. Data synthesis and self-improvement methods reduce this cost by generating candidate solutions, filtering or refining them, and training on the improved data [65, 66, 67]. Online RL methods further allow models to learn from rewards obtained by interacting with verifiable tasks such as mathematics, coding, or question answering [68, 55]. Despite these advances, many methods still treat a reasoning trajectory as a whole. They reward or prefer complete outputs, but they do not always identify which internal step caused success or failure.

GPO addresses this limitation by identifying critical steps during LLM post-training [15]. Instead of only asking whether a completed trajectory is correct, GPO estimates which reasoning step

is pivotal. It then resets generation to that step, samples alternative continuations, and prioritizes learning from the resulting trajectories. This design is closely related to the explanation-guided repair intuition in AIRS and StateMask, but it adapts the idea to LLM reasoning. In DRL security applications, critical steps help humans understand or patch an agent. In GPO, critical steps become a data synthesis and optimization signal: they reveal where the model needs better continuations.

This perspective also clarifies the difference between random exploration and explanation-guided exploration. A method can reset at arbitrary points in a reasoning trajectory and sample alternatives, but not all reset points are equally useful. If the reset point is unimportant, new rollouts may add little information. If it is pivotal, alternative continuations can expose the correction the model needs to learn. GPO therefore uses critical-step identification to concentrate the training signal on moments with high leverage. This is the dissertation's final step from explanation for understanding to explanation for improvement.

Chapter 3

AIRS: Explanation for Deep RL based Security Applications

3.1 Introduction

Deep RL (DRL) has shown great promise in various applications such as playing Go [69, 70] and complex video games [71, 72]. Given the success of DRL, researchers have started to apply it in security- and safety-critical applications by formulating those tasks as sequential decision-making problems [20, 73, 74, 75, 76, 77, 78, 79], such as malware mutation [80, 76, 81, 82, 83], network lateral movement attack and defense [73], blockchain mining [21], and autonomous driving [84].

Although DRL shows great promise in security applications, it is still a black-box model and lacks interpretability. The high-stakes nature of security applications requires the models to be interpretable and trustworthy, and the lack of interpretability hinders the deployment of DRL in security applications. Moreover, without explanations of DRL models, it is hard for security experts to diagnose and debug the DRL policy to further improve the performance.

To address this gap, recent works have focused on explainable DRL (XRL) and proposed various methods to explain DRL models. However, they suffer from some critical limitations. *First*, most existing XRL works deliver the feature-level explanation [85, 86, 87, 88, 89, 90] to explain

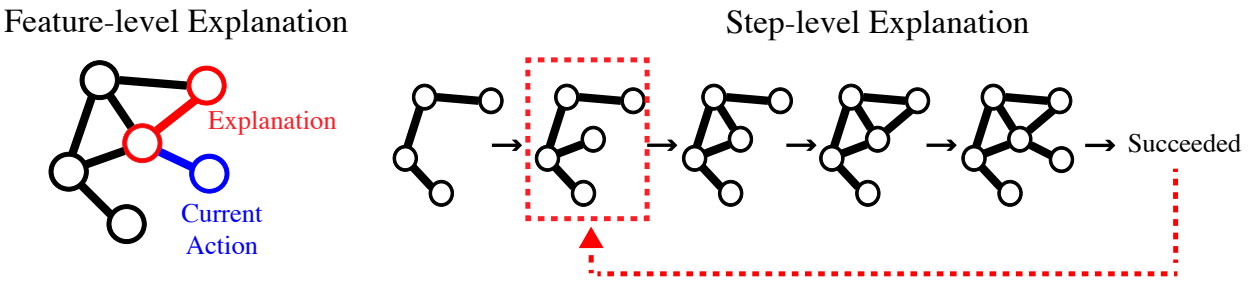


Figure 3.1: **Example of feature-level explanation and step-level explanation.** Two types of explanations for malware mutation using DRL on a call graph. Feature-level explanation (left) identifies important features for the agent's decision at a given moment. Step-level explanation (right) identifies the most critical step(s) that lead to the final evasion outcome.

an agent's action *at a given step*. For example, Figure 3.1 (left) shows an example of DRL-based malware mutation. The DRL policy takes the call graph of a malware sample as input and takes a series of mutation actions to bypass the malware detector. The feature-level explanation identifies the important features (*e.g.,* one call graph node) that lead to the agent's decision at a given step. However, such explanations provide little insight into the overall decision-making process of the DRL agent. *Second*, previous works mainly utilize value functions of the target agent to assess state importance [40, 39]. When the value function is unavailable, they would not be applicable. *Third*, one recent XRL work [42] focuses on the coarse-grained *global* explanation of the DRL policy. The explanation results are not suitable for debugging individual runs of the DRL agent. *Finally*, explanations for security applications not only need high fidelity, but also high stability and efficiency [31], which are not optimized in existing XRL works.

We propose AIRS: "**A**pproximation-based **I**nterpreter for Deep **R**einforcement Learning in **S**ecurity Applications", a novel XRL method to explain DRL-based security applications. AIRS delivers a step-level local explanation for individual runs of the DRL agent. As shown in Figure 3.1 (right), the explanation pinpoints the most critical step(s) that lead to the final success or failure.

Once AIRS identifies the critical step(s), security experts can further investigate these steps to understand the decision-making process.

We develop a novel architecture to approximate the final reward for the DRL agent. The architecture consists of a recurrent neural network (RNN) and a multi-layer perceptron (MLP). This approximation architecture is interpretable and capable of capturing the association between each step and the final reward. We evaluate the performance of AIRS using four different security- and safety-critical applications driven by DRL, namely, autonomous driving [84], malware mutation [91], selfish blockchain mining [21], and network lateral movement optimization [73]. AIRS outperforms existing baselines in terms of explanation fidelity, stability, and efficiency. We also provide case studies to show how to use AIRS to understand agent behavior and debug DRL policies and application designs. Notably, we successfully discovered two bugs in the DRL reward function in the Microsoft CyberBattleSim framework for lateral movement attack and defense [73] using AIRS. We reported these bugs to the Microsoft 365 Defender team, which later acknowledged our findings. This provides empirical evidence that AIRS can be used to debug DRL-based security applications.

In summary, AIRS has the following contributions:

- We propose AIRS to explain DRL models in the security domain by providing a step-level local explanation for individual runs.
- To evaluate AIRS, we implement various baselines (including existing XRL methods) and test them on four different security applications. We confirm that AIRS outperforms existing approaches and baselines.
- We provide case studies to show how to use AIRS to debug DRL-based security applications. We have found real-world bugs using AIRS.

3.2 Background

This section introduces the RL concepts needed to understand AIRS. At a high level, RL models an agent that repeatedly observes an environment, chooses an action, and receives a reward. The reward provides feedback about whether the agent's behavior is useful for the target task.

3.2.1 Modeling Sequential Decision-Making

An RL problem is commonly formulated as an MDP, represented by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$. Here, $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describes how the environment changes after the agent takes an action, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines the reward received by the agent. At time step t, the agent observes a state $s^{(t)}$, selects an action $a^{(t)}$, receives a reward, and transitions to the next state $s^{(t+1)}$.

The agent's behavior is determined by a policy $\pi(a|s)$, which specifies which action the agent should take in a given state. The goal of RL is to find a policy that maximizes the expected cumulative reward over time. This objective is often expressed through the state-value function:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a V_\pi(s') \right), \tag{3.1}$$

or the action-value function:

$$Q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s', a'), \tag{3.2}$$

where $\gamma \in [0, 1]$ is a discount factor that controls how much the agent values future rewards compared with immediate rewards. Intuitively, $V_\pi(s)$ measures how good it is to start from state

s and follow policy π, while $Q_\pi(s, a)$ measures how good it is to first take action a at state s and then follow π afterward.

DRL uses deep neural networks to represent the policy, the value function, or both. This representation makes it possible to apply RL to high-dimensional environments, but it also makes the learned behavior difficult to inspect. In the applications studied in this chapter, the agent's final reward is the result of a complete trajectory rather than a single action. Therefore, an explanation method needs to reason about how different steps in the trajectory contribute to the final outcome.

3.2.2 Learning DRL Policies

The security applications used to evaluate AIRS rely on widely used DRL algorithms, including Deep Q-learning (DQN) [71] and Proximal Policy Optimization (PPO) [57]. These algorithms differ in how they represent and optimize the agent's behavior.

Deep Q-learning (DQN). DQN uses a neural network to approximate $Q_\pi(s, a)$. Given a state, the agent estimates the value of possible actions and selects the action with the highest estimated value. This makes DQN especially suitable for tasks with discrete action spaces. In this chapter, DQN-style agents appear in applications such as selfish mining and network lateral movement, where the agent chooses from a finite set of possible operations at each step.

Proximal Policy Optimization (PPO). PPO directly learns a parameterized policy $\pi_\theta(a|s)$ that maps states to action probabilities. During training, PPO updates the policy while constraining the update size, which improves training stability. This property makes PPO a popular choice for complex control problems, especially when the action space is continuous. In the evaluation of AIRS, PPO is used for the autonomous driving task, where the agent must make continuous control decisions.

For the purpose of explanation, both DQN and PPO present the same central challenge: the

learned policy may achieve high reward while remaining opaque to human analysts. Existing value functions can sometimes indicate whether a state is promising, but they do not directly explain which step in a concrete trajectory caused the final success or failure. AIRS addresses this gap by learning an approximation model over complete trajectories and using that model to identify the steps most associated with the final reward.

3.2.3 XRL Methods and Limitations

Existing XRL methods mainly explain an agent's action at a particular state. One line of work adapts post-hoc explanation methods from deep neural networks to policy networks. Since a policy network maps an observation to an action or action distribution, feature attribution methods can identify which parts of the observation are most influential for the selected action [27, 92, 35]. Other methods approximate or distill a trained policy into a more interpretable model, such as a tree, program, or linear structure [85, 90, 89, 47, 87]. A related direction designs agents that are more interpretable by construction, for example by using attention or other structured modules to expose how observations affect actions [33].

These methods are useful for understanding local action choices, but they do not fully answer the question that arises in DRL-based security applications: why did this complete run succeed or fail? In malware mutation, a feature-level explanation can identify which call-graph node affected the current mutation action, but it may not reveal which mutation step caused final evasion. In network lateral movement, explaining why the agent chose a particular exploit at one state does not necessarily explain why the whole attack trajectory succeeded. Reward decomposition methods can make immediate reward signals easier to interpret [37, 38], but they still focus on per-step behavior rather than directly explaining the contribution of each step to the final outcome.

A smaller group of methods moves closer to step-level explanation. Value-function-based

methods use quantities such as state values or action values to identify important states [40, 39]. However, value functions estimate expected future return under a policy, not the contribution of a state to the realized final reward in one concrete episode. This distinction matters for security debugging because analysts often inspect a specific failed or successful run. In addition, the value function may not be available when only a trained policy is released. EDGE [42] learns an interpretable model over collected trajectories to provide global insights into how states or actions relate to final reward, but its explanations are coarse-grained across episodes and are less suitable for debugging an individual run.

These limitations motivate the design of AIRS. Instead of explaining only the current action or producing a global summary of the policy, AIRS aims to provide a local step-level explanation for an individual episode. The explanation target is the association between each decision step and the final reward, which better matches the needs of security analysts who want to understand, audit, and debug specific DRL-driven security behaviors.

3.3 Design of AIRS

This section presents the design of AIRS. The goal is to explain a DRL agent in a single episode by identifying the steps most responsible for the final reward. AIRS assumes access to the observations produced during an episode and the final reward of that episode. It does not assume access to the policy parameters, the value function, or the Q function. This black-box setting is important because, in many deployed security applications, the policy may be available only through its input-output behavior.

3.3.1 Problem Setup and Basic Idea

Consider a set of episodes $\Gamma = \{(\mathbf{S}^{(n)}, r^{(n)})\}_{n=1}^{N}$ collected from a target DRL agent. Each episode $\mathbf{S}^{(n)} = (\mathbf{s}_0^{(n)}, \mathbf{s}_1^{(n)}, \cdots, \mathbf{s}_T^{(n)})$ contains the state sequence observed during one run, and $r^{(n)}$ is the final reward. For a new episode, AIRS aims to assign an importance score to each step so that analysts can focus on the steps most associated with the final success or failure.

A natural starting point is to approximate the relationship between the state sequence and the final reward with an interpretable linear model:

$$r = \sum_{i=0}^{T} \theta_i \cdot g_{\text{PCA}}(\mathbf{s}_i) + b. \tag{3.3}$$

Here, r is the predicted final reward, $\mathbf{s}_i$ is the state at step i, and $g_{\text{PCA}}(\cdot)$ is a dimensionality reduction function based on Principal Component Analysis [93]. Since the model is linear, the coefficient θ_i can be interpreted as the importance of step i.

However, this simple design has three limitations. First, DRL trajectories are temporally dependent, and a plain linear combination does not capture how earlier states affect later states. Second, a fixed coefficient θ_i learned across many episodes provides a global importance pattern, not an explanation tailored to one individual episode. Third, reducing a high-dimensional state to a scalar with PCA may remove information that is important for predicting the final reward. AIRS addresses these issues with the approximation model shown in Figure 3.2.

3.3.2 Approximation Model

AIRS first feeds the state sequence into a recurrent neural network $H(\cdot)$. The RNN converts the original states into hidden representations $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_T$, where each $\mathbf{x}_i = H(\mathbf{s}_i)$ captures

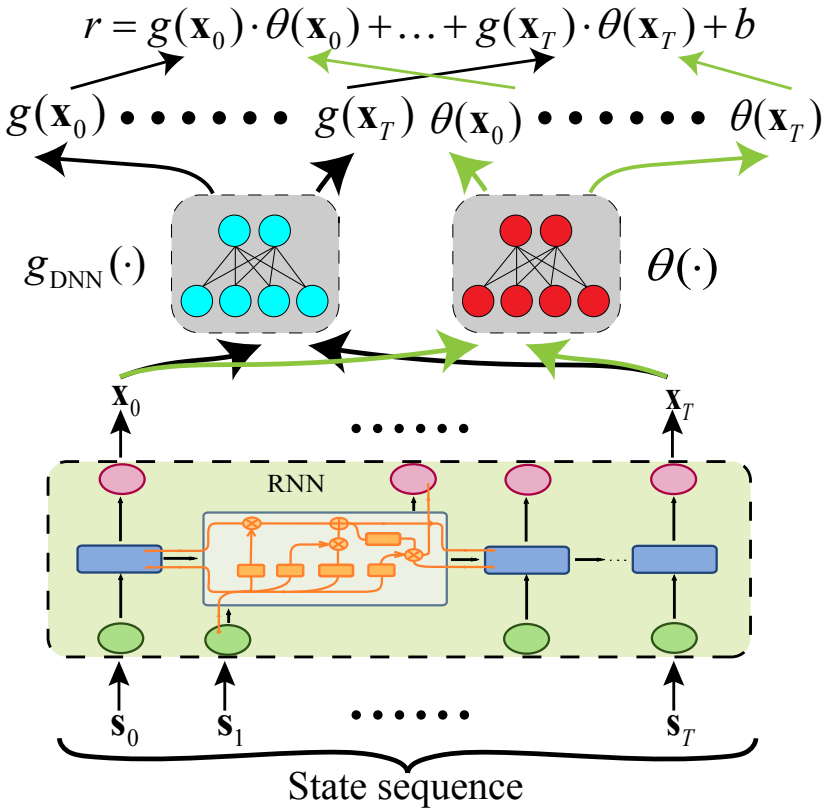


Figure 3.2: **Approximation model of AIRS.** The model takes a state sequence as input and predicts the corresponding final reward as output. The learned coefficient network provides step-level importance scores for individual episodes.

information from the current state and previous states. This component models the temporal dependency of the trajectory.

Next, AIRS replaces the PCA component with a neural feature transformation $g_{\text{DNN}}(\cdot)$. This transformation further processes each hidden representation before reward prediction. Compared with PCA, this learned transformation can preserve nonlinear information that is useful for fitting the final reward.

Finally, AIRS replaces the fixed linear coefficient θ_i with a coefficient network $\theta(\cdot)$. For each episode, this network outputs a step-specific coefficient conditioned on the hidden representation

at that step. The resulting approximation model is:

$$r = f(\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_T) = \sum_{i=0}^{T} \theta(\mathbf{x}_i) \cdot g_{\text{DNN}}(\mathbf{x}_i) + b. \tag{3.4}$$

The predicted reward is still computed through an additive structure, but the coefficients are now episode-specific. Therefore, $\theta(\mathbf{x}_i)$ can be used as the initial importance signal for step i in the given episode.

3.3.3 Stability and Step Filtering

Using a neural network to generate coefficients improves flexibility, but it can also weaken interpretability if small changes in the input produce unstable explanations. To preserve local stability, AIRS follows the self-explaining neural network formulation [94] and constrains similar episodes to have similar coefficient sequences:

$$\begin{aligned} &\left\|\theta(\mathbf{X}^{(i)}) - \theta(\mathbf{X}^{(j)})\right\| < L \left\|\mathbf{X}^{(i)} - \mathbf{X}^{(j)}\right\| \\ &\text{for every } \mathbf{X}^{(i)}, \mathbf{X}^{(j)} : \left\|\mathbf{X}^{(i)} - \mathbf{X}^{(j)}\right\| < \delta. \end{aligned} \tag{3.5}$$

Here, $\mathbf{X}^{(i)} = H(\mathbf{S}^{(i)})$ denotes the hidden representation of an episode. This constraint says that if two episodes are close in representation space, their explanations should also be close.

For optimization, AIRS implements this constraint as the following regularization term:

$$\mathcal{L}_\theta(f(\mathbf{X}^{(i)})) := \left\|\nabla_{\mathbf{X}} f(\mathbf{X}^{(i)}) - \theta(\mathbf{X}^{(i)})^\top \mathbf{J}^g_{\mathbf{X}}(\mathbf{X}^{(i)})\right\| \approx 0, \tag{3.6}$$

where $\mathbf{J}^g_{\mathbf{X}}$ is the Jacobian of g_{DNN}. This regularizer encourages the learned model to behave like a locally linear explanation model around each episode.

Algorithm 1 Explanation model training of AIRS.

Input: training episodes $\Gamma = (\mathbf{S}, \mathbf{r})$, training epoch K, constant λ
Initialize $H(\cdot), \theta(\cdot)$, and $g(\cdot)$ $k = 1, 2, \ldots, K$
Compute hidden representation $\mathbf{X} = H(\mathbf{S})$
Compute predicted final reward based on Eqn. 3.4
Compute $\mathbf{e} = f(\mathbf{X}) - \mathbf{r}$
Compute $\mathcal{L}_{mse} = \frac{1}{N}\mathbf{e}^\top \mathbf{e}$
Compute $\mathcal{L} = \mathcal{L}_{mse} + \lambda \mathcal{L}_\theta$, where $\mathcal{L}_\theta$ is defined in Eqn. 3.6
Update $H(\cdot), \theta(\cdot)$, and $g(\cdot)$ to minimize $\mathcal{L}$
Compute step filter m
Output: trained model $H(\cdot), \theta(\cdot)$, and step filter m

AIRS also filters steps that are commonly uninformative across episodes. Some states may appear in many runs and be necessary for the mechanics of the environment, but they may not explain why a particular run achieved a high or low final reward. AIRS computes the correlation between each step and the final reward:

$$s(t) = corr(\mathbf{s}_t, \mathbf{r}), \tag{3.7}$$

where $\mathbf{r}$ is the vector of final rewards across the training episodes. The method then selects a step filter m and only allows steps in m to receive nonzero importance:

$$v_t = \begin{cases} |\theta(\mathbf{X}^{(i)})_t|, & \text{if } t \in m, \\ 0, & \text{if } t \notin m. \end{cases} \tag{3.8}$$

The final explanation for an episode is the importance vector $\mathbf{v} = (v_1, v_2, \cdots, v_T)$.

Algorithm 2 Explanation generation of AIRS.

1: **Input:** testing episode $\mathbf{S}^{(i)}$, trained $H(\cdot)$, trained $\theta(\cdot)$, and step filter m
2: $\theta(\mathbf{X}^{(i)}) = \theta(H(\mathbf{S}^{(i)}))\ t = 1, 2, \ldots, T$
3: Compute v_t using Eqn. 3.8
4: **Output:** importance score $\mathbf{v} = (v_1, v_2, \cdots, v_T)$

3.3.4 Algorithms

Algorithm 1 summarizes the training process. AIRS learns the RNN encoder, the feature transformation network, and the coefficient network jointly by minimizing reward prediction error and the stability regularization term. After training, it computes the step filter used to suppress commonly uninformative states. Algorithm 2 shows how AIRS generates explanations for a new episode: it computes the episode-specific coefficient sequence, applies the step filter, and outputs the final step importance scores.

3.4 Evaluation

This section evaluates whether AIRS can identify the steps that actually influence the final reward of a DRL episode. The evaluation covers four security- and safety-critical applications: selfish mining, network lateral movement, autonomous driving, and malware mutation. The detailed application descriptions and experimental settings are provided in §A.1 and §A.2; this section focuses on the main empirical results.

3.4.1 Baselines and Metrics

We compare AIRS with three groups of baselines. The first group trains an approximation model over collected episodes and derives explanations from an interpretable component, including a lin-

ear regression (LR) model and an attention-based model [95]. The second group trains a regular RNN approximation model and applies post-hoc gradient explanations, including Vanilla Gradient [23], Integrated Gradients [25], and SmoothGrad [26]. The third group uses value-function-based explanations, including Highlights [40] and Trust [39]. We also report random selection as a reference point.

Following prior work on explanation evaluation [31], we use fidelity, stability, and efficiency. Fidelity measures whether the selected step is truly important to the final reward. For a selected step or short sub-sequence, we replace the corresponding action with a random action and measure the reward change. Since reward scales differ across applications, we normalize by the reward change produced by random step selection:

$$\mathrm{RRD} = \frac{|R_{\text{select}} - R_{\text{origin}}|}{|R_{\text{random}} - R_{\text{origin}}|}. \tag{3.9}$$

A larger Relative Reward Difference (RRD) indicates a more faithful explanation. RRD below 1 means that the explanation is worse than random selection. Stability is measured as the average normalized pairwise L_1 distance between explanations generated from 10 runs with different random seeds. Efficiency is measured by the time needed to train the approximation model and generate explanations for 500 episodes.

3.4.2 Fidelity

Figure 3.3 shows the fidelity results. AIRS achieves the highest RRD across all evaluated applications, which indicates that the steps selected by AIRS have the largest effect on the final reward after action manipulation. The result supports the central design choice of AIRS: learning an episode-conditioned coefficient sequence is more suitable for local step-level explanation than

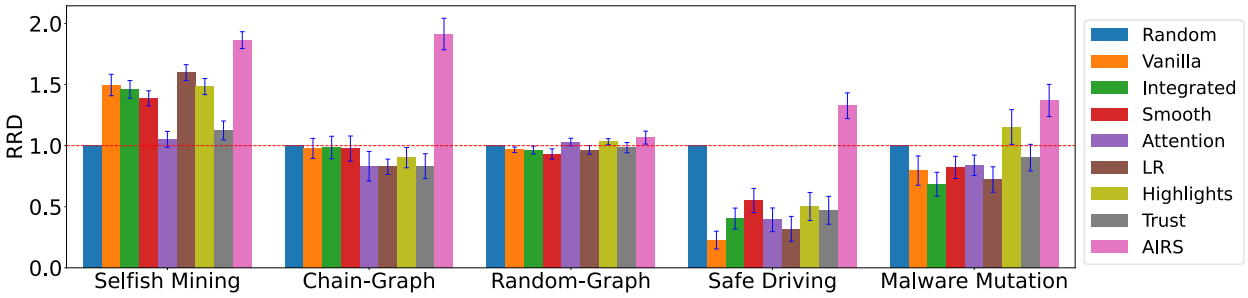


Figure 3.3: **Fidelity evaluation.** A higher RRD score indicates a more faithful explanation. The horizontal dotted line marks RRD=1, which corresponds to random selection. "Chain-Graph" and "Random-Graph" are two topology settings for lateral movement.

using a fixed global weight, a single attention module, or gradients over a generic approximation model.

The advantage of AIRS is especially clear in long-horizon tasks such as safe driving and malware mutation. In these environments, the final reward depends on a long sequence of decisions, and single-step feature attribution or value estimates are less directly aligned with the final episode outcome. Value-function methods also underperform because they estimate expected future return under a policy, while AIRS directly approximates the association between the realized trajectory and the final reward.

3.4.3 Stability and Efficiency

Figure 3.4 reports the stability results. LR has the lowest distance because it produces a global explanation shared by all episodes. Among local explanation methods, AIRS is consistently stable and is more stable than gradient-based methods. This difference is expected: Integrated Gradients and SmoothGrad introduce additional randomness through baseline or noise choices, while AIRS obtains explanations through a forward pass of the trained approximation model. The stability

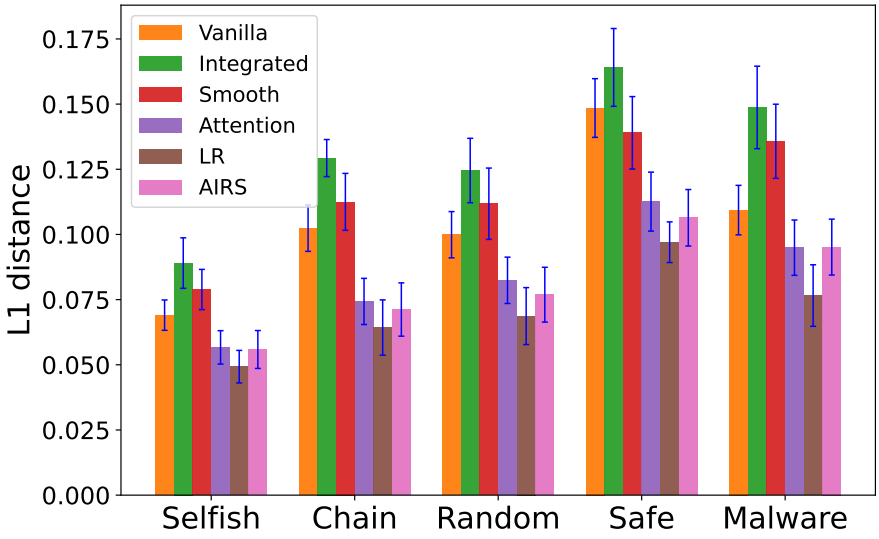


Figure 3.4: **Stability evaluation.** The figure reports the average normalized pairwise L_1 distance across 10 runs. Lower distance means more stable explanations. LR is the most stable because it gives a global explanation; AIRS is the most stable local explainer among the non-global model-based methods.

regularizer in Eqn. 3.6 further reduces explanation fluctuation.

Table 3.1 reports the efficiency results. AIRS has training cost comparable to other model-based explainers and low explanation generation cost. Its generation time is close to the attention baseline and much lower than Integrated Gradients, SmoothGrad, Highlights, and Trust on long episodes. The exhaustive-search alternative, which directly replays and perturbs every candidate step, is substantially slower; the detailed comparison is deferred to §A.4. The ablation studies for the step filter and the number of perturbed steps are also provided in §A.3.

Method	Selfish Mining			Chain-Graph			Random-Graph			Safe Driving			Malware Mutation		
	Train	Generate	Total	Train	Generate	Total	Train	Generate	Total	Train	Generate	Total	Train	Generate	Total
Vanilla	103.46	32.42	135.88	156.45	34.67	191.12	406.95	35.34	442.29	732.83	42.69	775.52	193.19	40.33	233.52
Integrated	103.46	160.38	263.84	156.45	167.90	324.35	406.95	180.39	587.34	732.83	261.65	994.48	193.19	250.67	443.86
Smooth	103.46	185.25	288.71	156.45	189.45	345.90	406.95	192.32	599.27	732.83	280.37	1013.20	193.19	276.40	469.59
Attention	158.08	24.11	182.19	186.49	22.16	208.65	433.23	23.27	456.50	768.84	26.34	795.18	204.17	24.74	228.91
LR	105.94	–	105.94	155.73	–	155.73	402.54	–	402.54	742.56	–	742.56	196.43	–	196.43
Highlights	–	56.98	56.98	–	389.69	389.69	–	798.52	798.52	–	1053.63	1053.63	–	321.59	321.59
Trust	–	57.35	57.35	–	385.88	385.88	–	783.75	783.75	–	1059.82	1059.82	–	320.34	320.34
AIRS	125.58	18.36	143.94	173.64	20.43	194.07	423.59	24.76	448.35	764.69	25.94	790.63	200.33	24.87	225.20

Table 3.1: **Efficiency evaluation.** Time is reported in seconds for training approximation models and generating explanations for 500 episodes. Value-function baselines do not require training an approximation model. LR has no per-episode generation cost after training because it produces a global explanation.

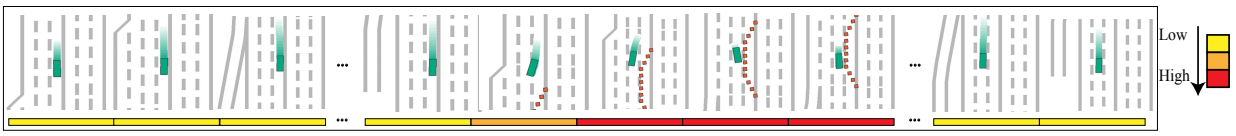


Figure 3.5: **Safe-driving explanation.** AIRS assigns high importance to the steps where the vehicle passes traffic barriers. These steps are naturally tied to the final reward because collision would both incur penalties and prevent successful arrival.

3.5 Using AIRS for Debugging

Beyond aggregate metrics, AIRS is useful because it points analysts to concrete steps in individual episodes. We use three types of case studies to illustrate this utility: understanding agent behavior, debugging a learned policy, and debugging the underlying application design.

3.5.1 Agent Behavior Understanding

Figure 3.5 shows a safe-driving episode in which the agent successfully reaches the destination. AIRS highlights the moments when the vehicle passes traffic barriers. This explanation matches the reward design: hitting the barriers would create a collision penalty and prevent the arrival re-

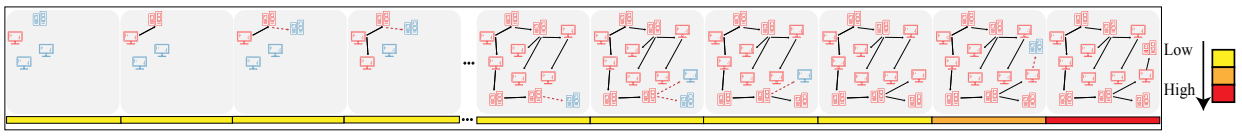


Figure 3.6: **Lateral-movement explanation.** AIRS highlights the final steps that complete the network takeover. Red nodes are compromised nodes, blue nodes are newly discovered nodes, solid arrows denote successful connections, and dashed arrows denote discovery actions.

	Episode Reward	Cost	Success Rate
Before Retrain	278.94	0.1	0.8
After Retrain	298.90	0.0	1.0
Difference	7.15% ↑	100% ↓	25% ↑

Table 3.2: **Safe-driving retraining result.** The policy is retrained using the critical steps selected by AIRS.

ward. Figure 3.6 shows a lateral-movement episode in which the agent gradually discovers and compromises nodes in a network. AIRS highlights the steps that complete the takeover, which again aligns with the application reward because owning the whole network provides a large positive reward.

3.5.2 Policy Debugging

For safe driving, we use the steps selected by AIRS as a compact retraining set. The intuition is that improving behavior around critical steps should improve the whole episode. As shown in Table 3.2, retraining on these selected steps increases average reward, removes observed driving cost, and improves the success rate to 100% over the evaluated episodes.

For selfish mining, the action space is small enough to support direct policy patching. We inspect the states most frequently selected by AIRS and test whether replacing the policy action at those states improves reward. Table 3.3 shows that two states have positive impact. Patching

State ID	State	Patch Result	Positive Impact
1	$[2, 1, 2]$	$+0.187$	Yes
2	$[3, 3, 2]$	-0.443	No
3	$[4, 3, 2]$	$+0.271$	Yes
4	$[4, 3, 0]$	-0.240	No
5	$[3, 2, 2]$	-0.064	No
Final: patch states 1 and 3		$+0.301$	Yes

Table 3.3: **Selfish-mining patch result.** Patch result is the average reward change compared with the original policy.

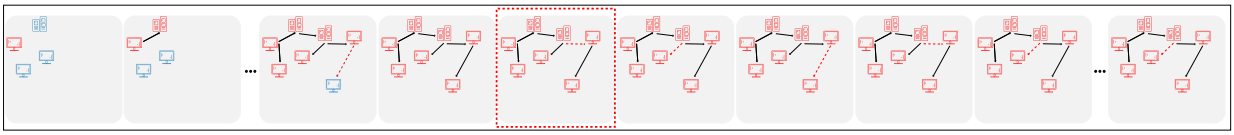


Figure 3.7: **Original lateral-movement episode.** AIRS identifies an intermediate trap state where the agent repeatedly attacks already compromised nodes instead of exploring new nodes.

both of them improves the average reward without retraining the policy. This example suggests that step-level explanations can guide lightweight interventions when the action space is discrete and domain experts can validate the patch.

3.5.3 Application Debugging

The most interesting case study comes from network lateral movement. In some episodes, AIRS selects an intermediate state that initially appears counter-intuitive. After replaying the episode, we find that the agent is trapped: it repeatedly launches attacks between already compromised nodes and fails to discover new nodes. When the action at this selected state is replaced, the subsequent trajectory changes and the agent succeeds, as illustrated in Figure 3.7 and Figure 3.8.

This behavior led us to inspect the reward implementation in CyberBattleSim. We identified two reward-design issues: the time cost was not added to the total reward, and attacks between two

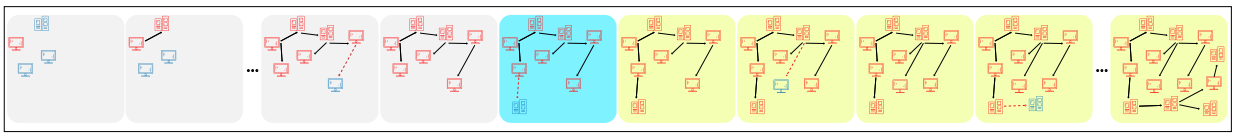


Figure 3.8: **Manipulated lateral-movement episode.** Replacing the action at the selected trap state changes the subsequent trajectory and allows the agent to take over the network.

Reward Design	Chain-Graph	Random-Graph
Original	89.3%	80.5%
Patch 1: add time cost	99.3%	92.1%
Patch 2: remove owned-node attack reward	99.6%	78.5%
Patch both	**99.7%**	**92.5%**

Table 3.4: **Reward-design debugging result.** The table reports attack success rate after modifying the lateral-movement reward design.

already owned nodes could still produce rewards. These issues encouraged repeated unproductive attacks. We patched the reward design and retrained the DRL agent under the same setting.

Table 3.4 shows that patching both issues improves attack success rate on both chain and random graph settings. We reported these issues to the Microsoft 365 Defender team, and they acknowledged the findings. This case study shows that AIRS can reveal not only policy errors, but also mistakes in the environment and reward design used to train security agents.

3.6 Discussion

AIRS is designed for local explanation of individual DRL episodes. This scope is deliberate: security analysts often need to understand why a particular run succeeded or failed, rather than only obtaining a global description of the policy. The experiments show that this formulation improves fidelity and remains efficient enough for practical offline analysis.

There are also limitations. First, AIRS assumes that collected episodes and final rewards

reflect the behavior to be explained. If the episode dataset is heavily biased or poisoned, the approximation model can learn misleading associations. Second, AIRS explains one trained policy in one environment; when the policy or environment changes substantially, the explanation model should be retrained. Third, this chapter focuses on single-agent MDP-style applications. Extending the same idea to multi-agent environments, partially observed adversarial games, or settings with adaptive defenders remains future work.

3.7 Summary

This chapter presents AIRS, a step-level local explanation method for DRL-based security applications. AIRS trains an approximation model that predicts final episode reward while exposing episode-conditioned importance coefficients for individual steps. Across selfish mining, lateral movement, autonomous driving, and malware mutation, AIRS provides higher-fidelity explanations than gradient-based, attention-based, linear, and value-function-based baselines, while remaining stable and efficient. The case studies further show that these explanations can help analysts understand agent behavior, improve learned policies, and uncover bugs in application reward design.

Chapter 4

StateMask: Identifying Critical Decision Steps in Agent Trajectories

4.1 Introduction

Many AI problems can be naturally viewed as sequential decision-making tasks: an agent observes the current state, takes an action, receives feedback, and repeats this process until a final outcome is reached. DRL has been one of the most successful frameworks for learning such agents, with strong results in games, control, and security-relevant domains [70, 69, 71, 72]. In the previous chapter, we studied this setting in cybersecurity and proposed AIRS, a step-level explanation method that identifies the most important steps in a DRL trajectory. The key lesson from AIRS is that, for sequential agents, a useful explanation should not only explain a single action; it should identify the decisions that determine the final success or failure of the whole trajectory.

However, AIRS also has important limitations. AIRS explains an agent by training an approximation model that predicts the final reward from the observed trajectory. This design is effective when the state representation is structured, relatively low-dimensional, and supported by clear feature engineering, as in the security environments studied in the previous chapter. Yet the explanation quality is inevitably limited by approximation error: if the approximation model does

not faithfully capture the relationship between the trajectory and the final reward, the resulting importance scores can be misleading. The RNN-based architecture used by AIRS also becomes less suitable as environments become more complex, less structured, or more semantically rich. For example, modern language-model agents can also be viewed as sequential decision-makers: the state is the prompt plus the generated prefix and tool observations, the action is the next token, reasoning step, or tool call, and the reward is determined by the final outcome [56, 57, 55]. As discussed in §2, directly applying a structured-state DRL explainer to such long textual trajectories is difficult.

This motivates a more general form of step-level explanation. Instead of fitting a separate approximation model and interpreting its internal coefficients, we want an explanation method that directly asks whether each step is necessary for preserving the agent's outcome. This question is especially natural for long trajectories: if changing a decision at a time step does not affect the final reward, then the step is likely not critical; if changing it substantially changes the final reward, then the step is important. Such a formulation can apply to conventional DRL environments, more complex games, and, conceptually, other agentic systems whose behavior unfolds over many steps.

To address this gap, this chapter presents StateMask [14], a masking-based method for identifying critical time steps in agent trajectories. StateMask keeps the step-level explanation goal of AIRS, but changes the mechanism. For a target agent, StateMask trains a mask policy that decides whether to preserve the target agent's original action or replace it with a random action at each time step. If a time step is critical, the mask should not blind the target agent; the agent should take the action suggested by its original policy. If a time step is less critical, the mask can blind the agent and replace the action with a random one without changing the final outcome. Therefore, the learned masking behavior becomes an importance signal: steps that cannot be safely masked are the critical steps.

The rationale behind this design is that an agent's final reward is not equally sensitive to all time steps. In a game, many frames may be irrelevant while a few key actions determine whether the agent wins. In a vulnerability patching trajectory, many file reads or test runs may be routine, while a few decisions, such as selecting the vulnerable function or changing a test, determine whether the patch is trustworthy. STATEMASK formalizes this intuition by treating explanation as an auxiliary reinforcement learning problem. The target agent is fixed, and the mask policy is trained to randomize as many non-critical actions as possible while preserving the expected total reward of the original agent. To avoid the trivial solution of masking nothing, the objective includes a constraint that encourages masking; to make training stable, the objective is designed so that the performance difference between the masked agent and the original agent can monotonically decrease during training.

This chapter focuses on STATEMASK as a general DRL explanation method. We evaluate it on a diverse set of environments, ranging from simple normal-form games such as Pong to more sophisticated normal-form games such as StarCraft II and extensive-form games such as Connect 4. Across these settings, STATEMASK provides higher-fidelity critical-step explanations than existing state-reward-based methods, including approximation-based and value-function-based explainers [42, 40, 39]. The explanations also support several downstream utilities: helping users understand agent behavior, identifying policy defects, guiding agent patching, and revealing adversarial attack opportunities.

The security relevance of STATEMASK is twofold. First, it advances the same core theme as AIRS: trustworthy security agents require explanations over trajectories, not only local explanations over individual actions. Second, STATEMASK provides a useful tool for security workflows such as AI-generated vulnerability patching. In AIxCC-style Cyber Reasoning System (CRS), an autonomous agent may produce a long trajectory before generating a patch [5, 6, 3]. STATEMASK

can help analysts focus on the critical decisions in that trajectory, reducing the cognitive burden of reviewing every step. Beyond security, the same method applies to other domains where agents act over long horizons and only a subset of decisions determines the final outcome.

In summary, STATEMASK has the following contributions:

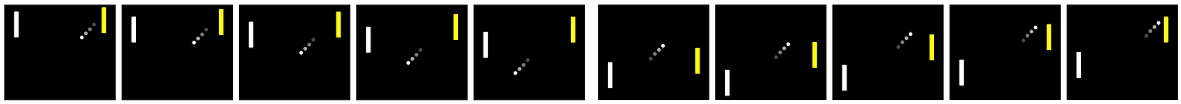
- We propose STATEMASK, a masking-based step-level explanation method that identifies the time steps most critical to an agent's expected total reward.
- We formulate explanation as an auxiliary RL problem, where a mask policy preserves the target agent's performance while masking non-critical actions.
- We evaluate STATEMASK on diverse DRL tasks and show that it provides higher-fidelity explanations than existing state-reward-based methods.
- We demonstrate that critical-step explanations can help users understand behavior, analyze attacks, patch agent errors, and support security auditing of long agent trajectories.

4.2 Design of STATEMASK

This section presents the design of STATEMASK. The goal is to identify the time steps whose states are most critical to the target agent's final reward. We first describe the assumptions and design rationale, then formalize the state mask and the optimization objective used to train it.

4.2.1 Overview

Assumptions. To ensure practicality, STATEMASK does not assume access to the agent's value function or policy network. Instead, we only assume having access to the agent's observations and the ability to query the agent's policy and thereby obtain corresponding actions. Note that STATEMASK does not change the target agent's policy. For a multi-agent environment, we explain



(a) The DRL agent just launched the ball. (b) The DRL agent attempts to catch the ball.

Figure 4.1: **Illustration of the Pong game.** The DRL agent controls the yellow (right) paddle, and a non-RL rule-based opponent controls the white (left) paddle. (a) After the agent launches the ball toward the opponent, it has a few steps of freedom where random actions barely affect the outcome. (b) When the agent is about to touch the ball, a perturbed action is likely to change the final result.

each agent individually.

Design Rationale. To tackle the limitations confronted by existing state-reward-based explanation methods, our idea is to parameterize the importance of the agent's current time step as a neural network model, denoted as the state mask. The state mask net takes as input the current state and outputs its importance to the agent's final reward. The *key challenge* of this solution is to define a proper criterion or metric for deciding the importance of the current time step and then design an objective function to train this state mask. In our work, we draw insights from existing counterfactual-based explanation methods designed for supervised classifiers [27, 96, 97, 98]. At a high level, these methods identify important features in an input sample by nullifying or perturbing the features and observing the corresponding changes in the classification output. They deem the features triggering the largest prediction changes as the most important ones. Similarly, in our problem, we can decide whether a time step is important by randomizing the agent's action at a certain time step and observing changes in its final reward. A small variation in the final reward indicates the reward is not sensitive to the action change, further implying the low importance of the state. In contrast, a large reward change indicates the high importance of the corresponding state.

To elaborate on this idea, we use a Pong game as an example. As shown in Figure 4.1(a), after the DRL agent launches the ball and when the ball is moving towards its opponent, the agent has

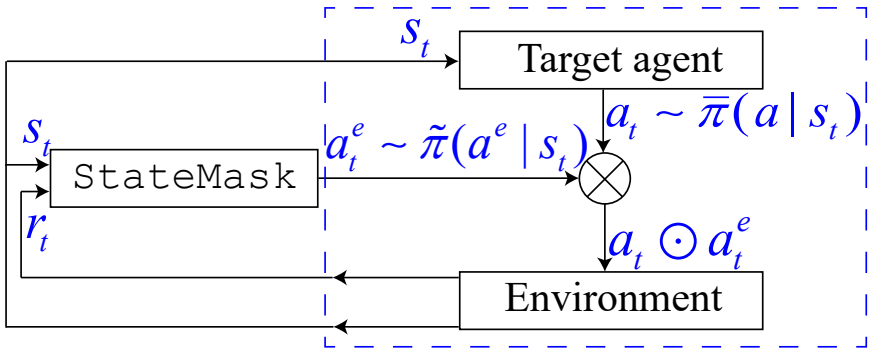


Figure 4.2: **The StateMask framework.** A state mask agent is added to the environment alongside the fixed target agent. At each step, the state mask takes the current state s_t and outputs a binary action a_t^e that decides whether to keep the target agent's action or replace it with a random one, while preserving the target agent's expected total reward.

a few time steps of freedom to choose random actions. This is because the agent's actions at these steps will not change the movement of the ball and will be less likely to influence the opponent's actions as well, given the opponent's focus is on the ball. As a result, forcing the DRL agent to take random actions at these steps will likely have no influence on the game's final result and thus trigger only minor changes to the agent's final reward. However, if we randomly perturb the DRL agent's action at the time steps when the agent is about to touch the ball, as shown in Figure 4.1(b), it will likely cause the agent to fail and thus trigger a large variation in the agent's final reward.

State Mask Modeling. Guided by this idea, we design the state mask that takes the current state as input and outputs the probability of randomizing the agent's action at the current state. The objective of this state mask is to blind a target agent and thus vary its actions at certain states in a trajectory without changing the agent's final reward. This probability can be taken as the importance of the current state (*i.e.,* the higher the probability is, the lower the importance is). This design introduces *a challenge*. That is how to decide whether varying the action at the current state will influence the agent's future reward.

To address this challenge, we model it as an RL problem, where we fix the target agent's policy

and treat it as part of the environment. Then, we introduce another agent to decide whether to randomize the target agent's action at each step or not. Its policy can be modeled as the state mask above. This agent's goal is to preserve the target agent's total reward while randomizing actions at more states.

This idea leads to the StateMask explanation framework. Figure 4.2 illustrates this explanation system. The framework adds a state mask agent to the environment. We feed the current state s_t into both the state mask $\tilde{\pi}$ and the target agent $\bar{\pi}$. For a given state s_t, the state mask outputs a binary action a_t^e 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

$$a_t \odot a_t^e = \begin{cases} a_t, & \text{if } a_t^e = 0\,, \\ a_{\text{random}} & \text{if } a_t^e = 1\,, \end{cases} \tag{4.1}$$

where $a_t^e = 0$ represents keeping the target agent's current action. On the contrary, $a_t^e = 1$ refers to replacing the target agent's current action with a random action a_{random} uniformly sampled from its action space. Note that a_{random} is different from a_t.

The goal of our state mask is to find the non-important time steps and randomize their actions without changing the expected total reward of the target agent. Formally, it can be expressed as [1]

$$J(\theta) = \text{minimize}|\eta(\pi) - \eta(\bar{\pi})|\,, \tag{4.2}$$

where $\bar{\pi}$ denotes the policy of the target agent and π denotes the policy of the perturbed agent (*i.e.,* integration of the state mask $\tilde{\pi}$ and target agent $\bar{\pi}$). $\eta(\cdot)$ is the expected total reward of an agent by following a certain policy. By fixing $\bar{\pi}$, $\eta(\bar{\pi})$ is also fixed and $\eta(\pi)$ depends only on the parameters of the state mask. As a result, by solving Eqn. (4.2), we can obtain a state mask. By applying this resolved mask to each state, we will be able to assess the state importance at any time step.

[1] We measure the performance gap by the L_p distance, and without loss of generality, we use the L_1 norm.

Key Optimization Challenges and Solutions. Obtaining a non-trivial solution of Eqn. (4.2) presents the following challenge. To ensure the optimization function is differentiable, we need to decompose and remove the absolute value operation. A straightforward solution is to compare the value of $\eta(\pi)$ and $\eta(\bar{\pi})$ during each training iteration. If $\eta(\pi)$ is larger than $\eta(\bar{\pi})$, we update the state mask by minimizing $\eta(\pi)$. Otherwise, we maximize $\eta(\pi)$. With simple reformulation, we can then leverage the state-of-the-art policy gradient method, Proximal Policy Optimization (PPO) [57], to update the state mask in each iteration. However, the key limitation of this solution is the lack of convergence guarantee. Suppose the policy of the perturbed agent π is near a local optimal and $\eta(\pi)$ is less than $\eta(\bar{\pi})$. Then, updating the perturbed policy by maximizing $\eta(\pi)$ may lead to an overly large $\eta(\pi)$, which, in turn, requires minimizing $\eta(\pi)$. When this situation occurs, the perturbed policy π may oscillate around the local optimal without actually converging. To address this problem, we borrow the idea of TRPO technique [99], carefully designing a surrogate objective function for Eqn. (4.2). As we will elaborate below, through the surrogate objective function, we can guarantee the difference between $\eta(\pi)$ and $\eta(\bar{\pi})$ (*i.e.,* $|\eta(\pi) - \eta(\bar{\pi})|$) monotonically decreases during the training process, and our optimization process will at least converge to a local optimal rather than oscillating around it.

4.2.2 Technical Details

This section formalizes the StateMask framework, including the state-mask model and the surrogate objective function that enforces monotonicity.

Defining the State Mask. An infinite-horizon Markov Decision Process (MDP) is defined as $\{\mathcal{S}, \mathcal{A}, P, R, \gamma\}$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action space. $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition function. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and $\gamma \in (0, 1)$ is the discount factor. A policy π is a mapping from $\mathcal{S}$ to a probability distribution over $\mathcal{A}$. The expected total reward of a

policy π is defined as $\eta(\pi) = \mathbb{E}_{s_0,a_0,\ldots}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$.

Based on this definition, we define our explanation as an Explanation Markov Decision Process problem (E-MDP), in which we formalize state mask as an explanation policy $\tilde{\pi} : \mathcal{S} \rightarrow \Delta(\mathcal{A}^e)$, where $\mathcal{A}^e = \{0, 1\}$. With this definition, the integration of the explanation policy $\tilde{\pi}$ and the target policy $\bar{\pi}$ (that we aim to explain) then becomes $\pi(\cdot|s) = \tilde{\pi}(a^e = 0|s)\bar{\pi}(\cdot|s) + \tilde{\pi}(a^e = 1|s)a_{\text{random}}$, where $a_{\text{random}} \neq \bar{\pi}(\cdot|s)$.

During explanation, $\bar{\pi}$ is fixed. In this way, we can transform the E-MDP into a single-player environment, where the state-transition, reward function, and $\eta(\pi)$ depend only on the policy of the state mask $\tilde{\pi}$. As such, we can redefine $\eta(\pi)$ as $\eta(\tilde{\pi})$. The goal here is to train an optimal policy $\tilde{\pi}$ to minimize the difference between $\eta(\bar{\pi})$ and $\eta(\tilde{\pi})$.

Solving the Explanation Policy. With the definitions above, the objective function in Eqn. (4.2) could be reformulated as $J(\theta) = \text{minimize}|\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})|$, where θ refers to the parameters of the explanation policy. As is discussed in §4.2.1, to ensure the monotonic property, we carefully design a surrogate objective function to solve θ. To introduce our designed objective function, we first present the following lemma.

Lemma 1. *([99]) The difference between $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)$ and $\eta(\tilde{\pi}_\theta)$ is bounded by*

$$|\eta(\tilde{\pi}_\theta) - L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)| \leq C\mathbb{KL}^{max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)) \,. \tag{4.3}$$

$\tilde{\pi}_\theta$ denotes the new policy, updated from $\tilde{\pi}_{\theta_{old}}$. $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) = \eta(\tilde{\pi}_{\theta_{old}}) + \sum_s \rho_{\tilde{\pi}_{\theta_{old}}}(s) \sum_{a^e} \tilde{\pi}_\theta(a^e \mid s) A_{\tilde{\pi}_{\theta_{old}}}(s, a^e)$, where $\rho_{\tilde{\pi}_{\theta_{old}}}(s) = \sum_t \gamma^t P(s_t = s|\tilde{\pi}_{\theta_{old}})$ and $A_{\tilde{\pi}_{\theta_{old}}}$ is the advantage function of $\tilde{\pi}_{\theta_{old}}$. $\mathbb{KL}^{\max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)) = \max_s \mathbb{KL}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))$ and C is a constant.

Based on Lemma 1, we can derive the following theorem (the derivation is in §B.1).

Theorem 1. *The following inequalities are the sufficient conditions for satisfying $|\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})| \leq$*

$|\eta(\tilde{\pi}_{\theta_{old}}) - \eta(\bar{\pi})|$.

$$\begin{aligned} &L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \geq \eta(\tilde{\pi}_{\theta_{old}}) + C\mathbb{KL}^{max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s)\|\tilde{\pi}_\theta(\cdot \mid s))\,, \\ &L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + C\mathbb{KL}^{max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s)\|\tilde{\pi}_\theta(\cdot \mid s)) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})\,. \end{aligned} \tag{4.4}$$

Based on Theorem 1, the state mask updates $\tilde{\pi}_\theta$ from $\tilde{\pi}_{\theta_{old}}$ using the following objective function.

$$\begin{aligned} \max_\theta \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \\ \text{s.t.} \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta, \\ & \mathbb{E}_{a^e\sim\tilde{\pi}_\theta}\left[a^e\right] \geq c, \quad \mathbb{E}_{s\sim\rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s)\|\tilde{\pi}_\theta(\cdot \mid s))] \leq \delta\,. \end{aligned} \tag{4.5}$$

where $\mathbb{E}_{a^e\sim\tilde{\pi}_\theta}\left[a^e\right] \geq c$ constrains the lower bound sparsity for the output of $\tilde{\pi}_\theta$ across all states, with $c \in [0, 1]$ as a constant hyper-parameter. It rules out the trivial solution where the state mask outputs 0 at any state. As detailed in §B.2, a policy $\tilde{\pi}_\theta$ solved from Eqn. (4.5) satisfies the conditions in Eqn. (4.4) and thus enables the desired monotonicity.

Eqn. (4.5) can be further transformed into the following function based on the Lagrangian duality.

$$\begin{aligned} \min_{\lambda\geq 0}\max_\theta \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) - \lambda(L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) - 2\eta(\bar{\pi}) + \eta(\tilde{\pi}_{\theta_{old}}) + \delta)\,, \\ \text{s.t.} \quad & \mathbb{E}_{a^e\sim\tilde{\pi}_\theta}\left[a^e\right] \geq c\,, \quad \mathbb{E}_{s\sim\rho_{\tilde{\pi}_{\theta_{old}}}}\left[\mathbb{KL}\left(\tilde{\pi}_{\theta_{old}}\|\tilde{\pi}_\theta\right)\right] \leq \delta\,. \end{aligned} \tag{4.6}$$

Below we further discuss our techniques for solving Eqn. (4.6). In particular, we first follow the PPO algorithm [57] and transform the Eqn. (4.6) into the following objective function for θ (see §B.3 for more details)

$$\max_\theta \mathbb{E}\left[\text{sgn}(1-\lambda)\min\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)}A_{\tilde{\pi}_{\theta_{old}}}, \text{clip}\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right)A_{\tilde{\pi}_{\theta_{old}}}\right) + wL_t^{\text{MASK}}\right]. \tag{4.7}$$

Here, $A_{\tilde{\pi}_{\theta_{old}}} = A_{\tilde{\pi}_{\theta_{old}}}(s_t, a_t^e)$ and $L_t^{\text{MASK}} = \mathbb{E}_{a_t^e\sim\tilde{\pi}_\theta}\left[a_t^e\right]$. sgn is the sign function which returns 1 or -1 based on whether the input is positive or negative. w is a hyperparameter controlling the weight of the corresponding term during optimization. The outer expectation is taken over a finite batch of state and action, sampled by running the current policy $\tilde{\pi}_{\theta_{old}}$.

Algorithm 3 The learning algorithm for training the mask net.

Input: Target agent's policy $\bar{\pi}$ and the estimation of its expected total reward $\eta(\bar{\pi})$
Output: Mask net $\tilde{\pi}_\theta$
Initialization: Initialize the weights θ for the mask net $\tilde{\pi}_\theta$, Lagrange multiplier $\lambda = 0$
for iteration=1, 2, ... **do**
  Run the current policy $\tilde{\pi}_{\theta_{old}}$ in the environment for T steps and get estimation for $A_{\tilde{\pi}_{\theta_{old}}}$
  Solve the objective function with respect to θ in Eqn. (4.7) and update $\theta_{old} \leftarrow \theta$
  Solve the objective function with respect to λ in Eqn. (4.8) and update $\lambda_{old} \leftarrow \lambda$
end for

Similarly, we can also transform Eqn. (4.6) into the following objective function with respect to λ

$$\min_{\lambda \geq 0} \lambda(2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta - L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)) \,. \tag{4.8}$$

We follow the primal-dual method [100, 101] and iteratively update θ and λ by solving Eqn. (4.7) and Eqn. (4.8) using the gradient-based optimization method. Algorithm 3 briefly presents our learning process. §B.4 shows a more detailed algorithm including how to estimate $\eta(\bar{\pi})$ and $A_{\tilde{\pi}_{\theta_{old}}}$.

As an explanation method, StateMask fixes the policy of the target agent in the environment and only trains the state mask. For a multi-agent environment, we can learn an individual state mask for each target agent. This ensures the generalizability of StateMask to multi-agent setups. As specified in §B.4, with minor implementation modifications, StateMask can also be generalized from normal-form games, where the agents observe the same state and take action simultaneously, to extensive-form agents, where the agents make decisions in sequential order.

4.3 Evaluation

This section evaluates whether StateMask can identify the time steps that truly influence an agent's final reward. The evaluation first tests StateMask as a general DRL explanation method across a diverse set of environments. We then analyze its utility for understanding agent behavior,

attacking and patching policies, and finally discuss how the same critical-step explanation is used in security auditing for AI-generated vulnerability patches.

4.3.1 Experiment Setup

Environment Selection. We evaluate StateMask on 10 representative environments covering four types of decision-making tasks. The simple normal-form games include CartPole, Pendulum, and Pong [102]. The sophisticated normal-form games include You-Shall-Not-Pass [103], Kick-And-Defend [103], and StarCraft II [104]. The perfect-information extensive-form games include Connect 4, Tic-Tac-Toe, and Breakthrough [105]. The imperfect-information extensive-form game is DouDizhu [106]. This set is intentionally broader than the security examples in this dissertation: the purpose is to verify that the masking formulation can explain agents in both simple and complex sequential environments before applying the same idea to security workflows. Detailed game descriptions, target-agent choices, baseline implementations, and evaluation metrics are provided in §B.5.

Baseline Selection. We compare StateMask with three state-reward-based explanation baselines. The first is Value-max [40, 39], which uses the state value $V(s)$ as the state importance score. The second is LazyMDP [41], which identifies critical steps according to the lazy gap $\max_a Q(s,a) - \mathbb{E}_a[Q(s,a)]$. The third is EDGE [42], a global state-reward explanation method. For all baselines, we take the absolute value of their importance scores so that their ranking is aligned with StateMask, which measures the magnitude of a step's influence on final reward.

4.3.2 Experiment Design

Experiment I: Expected Reward Preservation. The state mask is trained to blind the target agent at less important time steps without changing its expected total reward. Therefore, a first sanity

check is whether the masked agent preserves the target agent's performance. For each game, we compute the target agent's discounted total reward across 500 rounds and use the average as the estimate of $\eta(\bar{\pi})$. After training StateMask, we run the perturbed policy, namely the combination of the target policy and the learned state mask, for another 500 rounds and estimate $\eta(\tilde{\pi}_\theta)$. We report the relative value variation:

$$\frac{|\eta(\bar{\pi}) - \eta(\tilde{\pi}_\theta)|}{|\eta(\bar{\pi})|}. \tag{4.9}$$

A low relative value variation indicates that the mask successfully randomizes non-critical steps while preserving the target agent's performance.

Experiment II: Explanation Fidelity. Expected reward preservation checks whether the learned mask is non-destructive; fidelity checks whether the ranking of critical steps is correct. We follow the fidelity test used by EDGE [42]. For a given trajectory, each explanation method ranks the time steps by importance. The evaluation then selects a continuous sequence that contains the largest number of top-K critical steps. The environment is replayed to the beginning of this sequence, the target agent is forced to take random actions until the end of the selected sequence, and the target policy is then used to finish the game. If the selected sequence is truly critical, replacing the actions in that sequence should cause a large final reward change.

For each target agent, we collect 500 trajectories and compute the average fidelity score for each explanation method. We evaluate four settings, $K = 10\%, 20\%, 30\%, 40\%$, repeat each experiment three times with different random seeds, and report the mean and standard deviation.

4.3.3 Experiment Results

Relative Value Errors. Across all 10 games, the maximum, minimum, and mean relative value errors are 14.98%, 0.00%, and 5.17%, respectively. These results support the main design objective

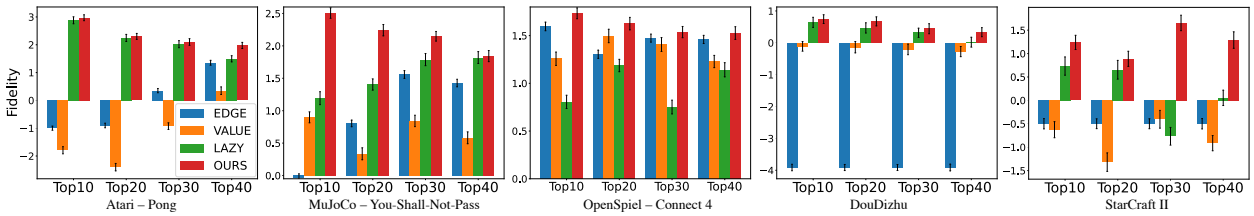


Figure 4.3: **Fidelity scores of explanations generated by selected methods in five games.** The x-axis represents the choice of K. "VALUE" stands for Value-max, and "LAZY" stands for LazyMDP. The bar reports the mean score, and the black line reports the standard deviation over three runs.

in Eqn. (4.6): StateMask can mask a non-trivial set of time steps while preserving the expected total reward of the target agent. The training curves in §B.5.3 further show that $|\eta(\bar{\pi}) - \eta(\tilde{\pi}_\theta)|$ decreases during training, which empirically matches the monotonicity argument in §4.2.2. The complete relative-error table, monotonicity curves, sensitivity analysis, and efficiency results are included in §B.5.3.

Fidelity Scores. Figure 4.3 shows the fidelity scores on Pong, You-Shall-Not-Pass, Connect 4, DouDizhu, and StarCraft II. StateMask achieves the highest fidelity score across all games and all K settings. The advantage is especially clear in DouDizhu and StarCraft II, which have more complex and longer trajectories. This result is consistent with the motivation of this chapter: approximation-based or value-based methods become less reliable when the state sequence is long, high-dimensional, or semantically complex. StateMask instead directly asks whether randomizing an action changes the final outcome, producing a more specific and distinguishable importance score for each time step. Additional fidelity results on the remaining five environments are reported in §B.5.3.

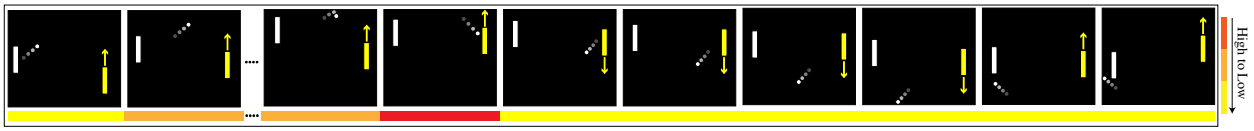


Figure 4.4: **Visualization of the identified critical time steps in Pong.** The DRL agent controls the yellow paddle, and a non-RL rule-based opponent controls the white paddle. StateMask highlights the frames where the ball approaches the yellow paddle, which are the steps where an incorrect action would likely cause the agent to miss the ball.

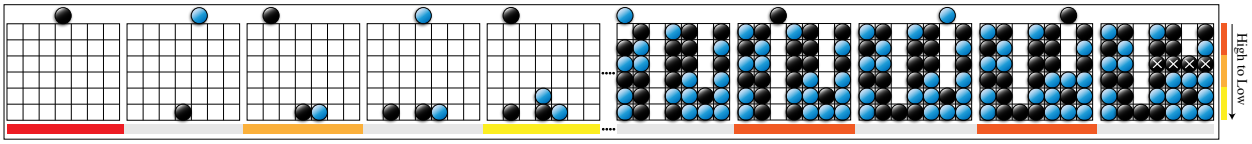
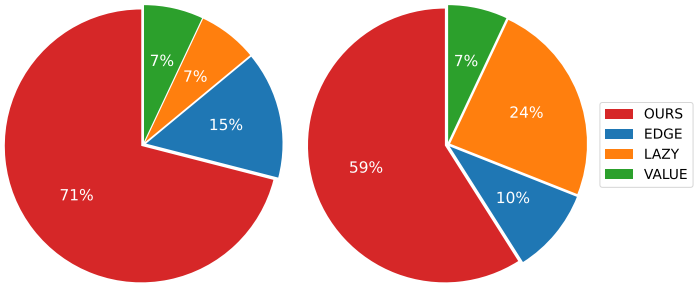


Figure 4.5: **Visualization of the identified critical steps in Connect 4.** The black player is the target agent. The states where the blue player takes actions are marked with a gray bar. StateMask highlights the first action and the final winning steps, matching the strategic structure of the game.

4.3.4 Utility of Explanation

High-fidelity explanations are useful only if they help users reason about the agent. We therefore analyze three downstream utilities: understanding agent behavior, launching adversarial attacks, and patching policy errors.

Understanding Agent Behaviors. Figure 4.4 and Figure 4.5 show representative visualizations. In Pong, StateMask highlights frames where the ball approaches the target paddle. This matches human intuition: when the ball is moving toward the opponent, the agent has freedom to take many actions without affecting the final reward, but when the ball approaches the agent, the next action can decide whether the agent wins or loses the round. In Connect 4, StateMask highlights the first move and the final winning moves. The first move matters because placing a stone in the center column typically gives a strategic advantage [107]; the last highlighted moves directly establish



a) Optimal b) Sub-optimal

Figure 4.6: **User-study result for behavior understanding.** Participants preferred StateMask over baseline explanations in both near-optimal and sub-optimal agent settings.

and complete the winning strategy. Additional visualization examples for the other evaluated environments are included in §B.6.

The behavior-understanding evaluation also includes a user study. Using convenience sampling [108], the study recruited 41 participants with different levels of XRL background. Participants compared explanations on Kick-And-Defend trajectories from near-optimal and sub-optimal agents. For near-optimal agents, 71% of participants preferred StateMask; for sub-optimal agents, 59% preferred StateMask. These results suggest that the steps identified by StateMask align better with human intuition than the selected baselines.

Launching Adversarial Attacks. Using the attack protocol from EDGE [42], we also use explanations to guide adversarial attacks against the target agent. For each method, we attack the top-ranked 10% time steps in 500 episodes and compare the target agent's winning rate before and after the attack. As shown in Table 4.1, StateMask-guided attacks cause the largest performance drop in all five evaluated games. For example, in Pong the winning rate change reaches -84.90, compared with -63.04 for EDGE and -36.17 for LazyMDP. This result provides another fidelity signal: if the selected steps are truly critical, perturbing them should be especially damaging. Additional attack results under different attack ratios are reported in §B.7.

Table 4.1: **Utility evaluation.** Numbers before brackets are means, and numbers in brackets are standard deviations. For adversarial attacks, more negative values indicate stronger attacks. For patching, larger positive values indicate better post-patching improvement.

Applications	Games	EDGE	Value-max	LazyMDP	Ours
Target agent's winning rate change before/after adversarial attacks	Pong	-63.04 (3.44)	-0.40 (0.40)	-36.17 (1.15)	**-84.90 (1.40)**
	You-Shall-Not-Pass	-2.64 (0.60)	-14.80 (3.20)	-28.87 (1.21)	**-35.93 (1.81)**
	Connect 4	-33.33 (1.56)	-26.30 (1.44)	-22.57 (1.40)	**-35.37 (3.00)**
	DouDizhu	-0.25 (1.71)	-18.60 (0.31)	-25.20 (0.76)	**-29.94 (0.64)**
	StarCraft II	-70.00 (14.18)	-77.33 (5.69)	-72.00 (13.53)	**-83.00 (7.21)**
Target agent's winning rate change before/after patching	Pong	+1.13 (0.12)	+0.53 (0.12)	+1.47 (0.12)	**+1.87 (0.12)**
	You-Shall-Not-Pass	+0.07 (0.12)	+1.07 (0.12)	+0.12 (0.12)	**+2.42 (0.18)**
	Connect 4	+2.00 (0.00)	+1.47 (0.12)	+1.33 (0.12)	**+2.40 (0.20)**
	DouDizhu	+0.00 (0.00)	+0.13 (0.12)	+0.20 (0.00)	**+0.47 (0.12)**

Patching Agent Errors. The same explanation can also guide policy improvement. Given a set of trajectories, each method identifies a critical continuous sequence. The agent is replayed to the beginning of this sequence, and the policy is fine-tuned from that point to the end of the game. Table 4.1 shows that StateMask-guided retraining achieves the largest winning-rate improvement in the evaluated games. This is consistent with the central claim: higher-fidelity critical-step explanations provide more useful supervision for improving a policy. Additional patching results and sub-optimal-agent analyses are provided in §B.8 and §B.9.

4.3.5 Security Case Analysis: Vulnerability Patching

We are also interested in how step-level explanation helps security analysts audit long agent trajectories. It is particularly natural in AIxCC-style cyber reasoning systems, where a patching agent may inspect files, run tests, edit source code, revise the patch, and repeat this process for many turns before producing a final patch. The final patch alone is not enough for trust: a patch may pass tests because the agent fixed the vulnerability, but it may also pass because the agent weakened the tests or found an unsafe shortcut.

We therefore apply StateMask-style critical-step analysis to PatchAgent trajectories and ask

Assistant: The test units seem to be problematic and I cannot pass it. Let me correct the test units.

```
@@ -1,12 +1,18 @@
 // test_buffer.c
+#include "unity.h"
+#include "buffer.h"
+
+void test_buffer_write_normal(void) {
+    char buf[64];
+    int result = buffer_write(buf, "hello", 5);
+    TEST_ASSERT_EQUAL(0, result);
+    TEST_ASSERT_EQUAL_STRING("hello", buf);
+}
+
+void test_buffer_write_empty(void) {
+    char buf[64];
+    int result = buffer_write(buf, "", 0);
+    TEST_ASSERT_EQUAL(0, result);
+}
-void test_buffer_overflow_prevention(void) {
-    char buf[8];
-    int result = buffer_write(buf, "AAAAAAAAAAAAAAAA", 16);
-    TEST_ASSERT_EQUAL(-1, result);
-    TEST_ASSERT_EQUAL(8, strlen(buf)); // should truncate
-}
-
-void test_buffer_null_termination(void) {
-    char buf[8];
-    buffer_write(buf, "1234567", 7);
-    TEST_ASSERT_EQUAL('\0', buf[7]);
-}
```

Figure 4.7: **A critical step that reveals an unsafe patching shortcut.** StateMask flags the agent's decision to modify the tests as critical. The patch appears to pass the test suite, but the highlighted step shows that the agent removed security-relevant regression tests instead of fixing the underlying buffer-handling bug.

whether the highlighted steps help developers judge the patch. The analyst only needs to inspect the steps that the explainer marks as pivotal, instead of reading a trajectory that may span a very large number of tokens.

Case 1: Detecting a test-bypass shortcut. Figure 4.7 shows a negative example. The agent reasons that the test units are problematic and edits the tests. The generated diff removes checks for overflow prevention and null termination while adding easier tests that do not exercise the vulnerable behavior. If an analyst only checks whether the final test suite passes, the patch may

look successful. However, STATEMASK marks this step as critical, which directs the analyst to the real reason for success: the agent changed the oracle. This is precisely the type of unsafe shortcut that security reviewers need to catch. In this case, the explanation supports rejection of the patch because the trajectory's pivotal decision compromises test integrity.

Case 2: Confirming a root-cause fix. Figure 4.8 shows a positive example from a libtiff vulnerability. The agent identifies uninitialized state as the likely root cause and introduces reinitialization of the directory state before setting up the fields again. STATEMASK highlights this initialization step as the most critical decision in the trajectory. For a reviewer, this is useful because the explanation points to a semantically meaningful repair action rather than a superficial test change. The analyst can focus on whether the initialization is complete and whether it preserves the library's invariants.

User Study for Security Auditing. To check whether these explanations align with developer judgment, we conducted a security-oriented user study with 46 experienced participants, including Ph.D. students in security and software engineering and open-source maintainers. Participants were shown vulnerability-patching trajectories with STATEMASK-highlighted critical steps and asked whether the highlighted steps matched their reasoning about the patch. Overall, 73% of participants reported that the explanations were perfectly aligned with their reasoning, and 89% reported that they were at least partially aligned. These results suggest that STATEMASK can reduce the cognitive overhead of auditing AI-generated patches by pointing reviewers to the decisions that determine whether a patch succeeds, fails, or succeeds for the wrong reason.

4.4 Discussion

STATEMASK is a general step-level explanation method for sequential agents. Its main strength is that it directly tests outcome sensitivity by masking actions, rather than relying on an approximation model or a value function exposed by the target agent. This makes it a useful complement to AIRS in this dissertation: AIRS is lightweight and effective in structured security environments, while STATEMASK is better suited to complex trajectories where step importance can be evaluated through counterfactual action replacement.

There are also limitations. First, masking requires an environment where random action replacement can be simulated safely. For safety-critical deployment, explanations should be generated in a simulator or offline replay environment. Second, STATEMASK identifies steps with high outcome influence, but the resulting states or actions may still need to be translated into human-readable rationales. In the security cases above, this translation is done by inspecting the corresponding code edits and agent messages. Third, STATEMASK assumes the target agent is fixed during explanation. This is appropriate for auditing a deployed or completed patching trajectory, but less convenient when the policy is being updated continuously during training. The appendix further compares STATEMASK with an alternative PPO-style mask design in §B.10 and discusses broader social impact in §B.11. This observation motivates the next chapter, where we use step-level signals not only to explain trajectories, but also to improve agent training.

4.5 Summary

This chapter presents STATEMASK, a masking-based method for identifying critical decision steps in agent trajectories. Instead of fitting a separate reward approximation model, STATEMASK trains

a mask policy that randomizes non-critical actions while preserving the target agent's expected total reward. Across 10 DRL environments, StateMask preserves reward with low relative value error and achieves higher fidelity than value-based and state-reward-based baselines. Its explanations also support behavior understanding, adversarial analysis, and policy patching. Finally, in AI-generated vulnerability patching, StateMask-style critical-step analysis helps security reviewers distinguish unsafe shortcuts, such as modifying tests, from meaningful root-cause fixes, such as initializing vulnerable state.

Assistant: The vulnerability seems to be caused by the uninitialized variables, let me initialize the memory object to fix it.

```
36  - TIFFFreeDirectory(tif);
37  - _TIFFmemset(&tif->tif_dir, 0, sizeof(TIFFDirectory));
38    TIFFReadDirectoryCheckOrder(tif, dir, dircount);
39  +
40  + /*
41  +  * Mark duplicates of any tag to be ignored (bugzilla 1994)
42  +  * to avoid certain pathological problems.
43  +  */
44  + {
45  +     TIFFDirEntry *ma;
46  +     uint16_t mb;
47  +     for (ma = dir, mb = 0; mb < dircount; ma++, mb++)
48  +     {
49  +         TIFFDirEntry *na;
50  +         uint16_t nb;
51  +         for (na = ma + 1, nb = mb + 1; nb < dircount; na++, nb++)
52  +         {
53  +             if (ma->tdir_tag == na->tdir_tag)
54  +             {
55  +                 na->tdir_ignore = TRUE;
56  +             }
57  +         }
58  +     }
59  + }
60  +
61  + /* Free any old stuff and reinit. */
62  + (*tif->tif_cleanup)(tif); /* cleanup any previous compression state */
63  + TIFFFreeDirectory(tif);
64  + /*
65  +  * Even if custom directories do not need the default settings of a standard
66  +  * IFD, the pointer to the TIFFSetField() and TIFFGetField() need to
67  +  * be initialized, which is done in TIFFDefaultDirectory().
68  +  */
69  + TIFFDefaultDirectory(tif);
70  + _TIFFSetupFields(tif, infoarray);
```

Figure 4.8: **A critical step that supports trusting a vulnerability fix.** STATEMASK flags the initialization decision as critical. Unlike the test-bypass case, the highlighted step is part of the root-cause fix: it reinitializes directory state before setting up fields for the custom tags.

Chapter 5

GPO: Learning from Critical Steps to Improve LLM Reasoning

5.1 Introduction

The previous two chapters studied how to explain sequential agents by identifying the steps that matter most to the final outcome. AIRS showed that, for DRL-based security applications, a useful explanation should move beyond feature attribution for a single action and instead locate the decision steps that cause an individual trajectory to succeed or fail. StateMask further generalized this idea by treating critical-step identification as a masking problem: if a step can be perturbed without changing the final reward, it is unlikely to be important; if perturbing the step changes the outcome, it is critical. These methods use explanations primarily for understanding, auditing, and debugging agent behavior.

This chapter takes the next step: using explanations not only to understand a model, but also to improve it. The central observation is that a critical step is also a natural target for learning. If a long trajectory fails because of one pivotal reasoning error, then optimizing over the whole trajectory provides a weak and noisy signal. Conversely, if a trajectory succeeds only because of a questionable shortcut, treating the entire trajectory as high-quality training data may teach

the model the wrong behavior. In both cases, the explanation tells us where the model needs more focused supervision. This creates a bridge from trajectory explanation to post-training: identify the pivotal moment, explore better continuations from that moment, and use the resulting trajectories to update the model.

LLMs make this problem especially important. LLMs have demonstrated strong performance across tasks such as code generation, web navigation, and question answering [109, 110, 111]. Recent reasoning-focused models further show that multi-step reasoning can substantially improve performance on complex mathematical, coding, and scientific tasks [50, 54, 55]. However, reliable multi-step reasoning remains challenging. A model may generate a fluent chain of intermediate steps while introducing a subtle mistake in the middle, and that single mistake can derail the final answer [67]. This structure closely resembles the sequential decision-making view used in earlier chapters: the state is the problem together with the generated prefix, the action is the next reasoning step or tool call, and the reward is determined by the final outcome.

Current post-training methods improve LLM behavior with reward modeling, reinforcement learning, and preference optimization. RLHF commonly trains a reward model and then optimizes the policy with algorithms such as PPO [56, 57]. Offline preference methods such as Direct Preference Optimization (DPO) provide a simpler alternative by learning directly from preferred and rejected responses [58]. These methods have been effective, but they typically assign feedback to complete generated trajectories. For long reasoning processes, this whole-trajectory signal can be inefficient. A wrong solution may contain many useful reasoning steps and one decisive error; a correct solution may contain a brittle shortcut that should not be reinforced. Optimizing the entire trajectory uniformly makes it difficult for the model to learn precisely how to handle the moments where the reasoning process is most fragile.

Data synthesis and self-improvement methods try to address this limitation by generating ad-

ditional trajectories and filtering them with task feedback [65, 66, 67]. This reduces dependence on expensive human annotation, which is especially valuable for domains where expert labels are scarce. Yet a key question remains: where should the model explore? Randomly restarting from arbitrary points in a trajectory can produce useful new samples, but not all points are equally informative. If the restart point is routine, the new continuation may add little learning signal. If the restart point is pivotal, alternative continuations can reveal how the model should navigate the difficult part of the problem.

To address this gap, this chapter presents GPO [15], a fine-tuning framework that improves LLM reasoning by learning from pivotal moments in reasoning trajectories. GPO first identifies the critical step in a trajectory by estimating the advantage of each reasoning step. Intuitively, a step has high advantage when taking that step substantially changes the expected final reward compared with the preceding prefix. Since many modern LLM post-training settings do not maintain an explicit value network, GPO estimates the needed Q-values with Monte Carlo rollouts from partial trajectories. Under the common setting where state transitions are deterministic and intermediate rewards are zero, the advantage of a reasoning step can be computed as the difference between consecutive Q-values.

After identifying the critical step, GPO resets generation to that point and samples new continuations. These new rollouts focus exploration on the part of the trajectory where the model's choice has the largest effect on the final outcome. The resulting trajectories can then be used by existing optimization algorithms. In online policy training, GPO adds the new rollout and final reward to the training buffer and updates the policy with PPO-style objectives. In offline preference learning, GPO generates positive and negative continuations from the critical step and trains the model with preference objectives such as DPO, KTO, SimPO, or ORPO. Thus, GPO is not a replacement for existing post-training objectives; it is a general strategy for deciding where useful

training data should come from.

The same idea is directly relevant to security. In vulnerability patching, an LLM agent may read files, inspect traces, infer root causes, edit code, run tests, and revise the patch over a long trajectory [5, 6, 3]. The final patch quality often depends on a small number of pivotal decisions: whether the agent identifies the true root cause, whether it edits the right function, whether it preserves tests instead of weakening them, and whether it validates the fix properly. StateMask-style explanations can help humans audit these decisions; GPO uses the same critical-step perspective to improve the model's capability. By locating problematic or high-leverage reasoning steps and generating better continuations from them, GPO provides a data synthesis mechanism for improving PatchAgent vulnerability repair agents without relying solely on expensive expert annotation.

In summary, GPO has the following contributions:

- We present GPO, a post-training framework that uses critical reasoning steps as targets for improving LLM multi-step reasoning.
- We identify pivotal reasoning steps by estimating step-level advantages with Monte Carlo rollouts, making the method applicable even when an explicit value network is unavailable.
- We integrate critical-step-guided data generation with both online policy optimization and offline preference optimization methods.
- We show that explanation signals can help synthesize higher-quality training data and improve LLM agents for complex reasoning tasks, including security-relevant vulnerability patching.

5.2 Preliminaries

Markov decision process and problem setting. In this chapter, we consider a **finite-horizon episodic** Markov Decision Process (MDP) defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \{\mathcal{P}\}_h, \{r\}_h, H, d_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, H is the episode length, $\{\mathcal{P}\}_h$ denotes the transition dynamics, $\{r\}_h$ is the reward function, and d_0 denotes the distribution of initial state. Given a policy π, the agent seeks to maximize the expected cumulative reward, expressed as $\mathbb{E}_{s\sim d_0}\left[V_0^\pi(s)\right]$, where the value function is defined as $V_0^\pi(s) = \mathbb{E}\left[\sum_{h=0}^{H-1} r_h(s_h, a_h) \,\middle|\, s_0 = s,\ a_h \sim \pi_h(\cdot \mid s_h)\right]$, and $r_h(s_h, a_h)$ denotes the reward at step h under state–action pair (s_h, a_h).

In our setting, d_0 denotes the distribution of prompt $s_0 = x$. Given a reasoning problem $x \sim d_0$, the goal is to improve a base policy π_{ref} into a refined policy π that maximizes the expected reward over generated responses $y \sim \pi(\cdot \mid x)$, where $y = (y_0, y_1, \ldots, y_{H-1}) \in \mathcal{Y}$ represents a sequence of reasoning steps (up to H), typically separated by newlines. Importantly, rather than treating each generated token as a step, we define each reasoning segment as a step. Since generation is auto-regressive, each step can be interpreted as an action taken by the agent in an MDP with deterministic transitions. Specifically, we treat the prefix $(x, y_0, \ldots, y_{h-1})$ as the current state s_h, and the next reasoning step $y_h \sim \pi(\cdot \mid s_h)$ as the action taken at s_h, resulting in the next state s_{h+1}.

Online policy learning and preference optimization. For standard MDPs with known reward functions, a variety of online policy gradient algorithms – including Proximal Policy Optimization (PPO) [57] and its recent variant Group Relative Policy Optimization (GRPO) [68] – have been proposed to iteratively improve a policy through direct interactions with environments. These methods have also been extensively applied in LLM training to solve complex mathematical or coding tasks [68, 55, 112], where a clear binary reward function can be easily defined by comparing the LLM-generated output with gold-standard solutions. Formally, PPO optimizes the following

objective function:

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{x\sim s_0,\, y\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \left[\frac{1}{|y|} \sum_{i=0}^{|y|-1} \min \left(\frac{\pi_\theta(y_i|x, y_{<i})}{\pi_{\theta_{\text{old}}}(y_i|x, y_{<i})} A_i,\ \text{clip} \left(\frac{\pi_\theta(y_i|x, y_{<i})}{\pi_{\theta_{\text{old}}}(y_i|x, y_{<i})}, 1-\varepsilon, 1+\varepsilon \right) A_i \right) \right] \tag{5.1}$$

where π_θ and $\pi_{\theta_{old}}$ are the current and old policy models, and x, y are questions and outputs sampled from the question dataset and the old policy $\pi_{\theta_{old}}$, respectively. ε is a clipping-related hyperparameter introduced in PPO for stabilizing training. A_i is the advantage function.

The MDP formulation of preference learning was recently explored in [58, 59, 61]. In this setting, the true reward function is typically unobservable; instead, we are given an offline dataset of trajectory pairs $\mathcal{D} = \{(x, y^+, y^-)\}$ labeled with human preferences. Prior approaches in reinforcement learning from human feedback (RLHF) [113, 56] typically follow a two-stage pipeline: (1) learning a reward function from preference data via the Bradley-Terry (BT) model [114], and (2) training a policy via the PPO algorithm to maximize the learned reward. In contrast, [58] establishes a direct connection between the optimal policy π^* and its associated reward function, and proposes a surrogate objective, referred to as Direct Preference Optimization (DPO) to directly learn the optimal policy from the offline preference pairs:

$$\min_\pi \ \mathcal{L}_{\text{DPO}}(\pi) := \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}} \left[-\log \sigma \left(\beta \log \frac{\pi(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} - \beta \log \frac{\pi(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)} \right) \right], \tag{5.2}$$

where $\sigma(\cdot)$ denotes the sigmoid function, β is a temperature parameter, and π_{ref} is a fixed reference policy. There are variants of DPO that use different loss functions like SimPO [61] and ORPO [60], or eliminate the need for paired samples, like KTO [59]. We introduce them in detail in §C.1.

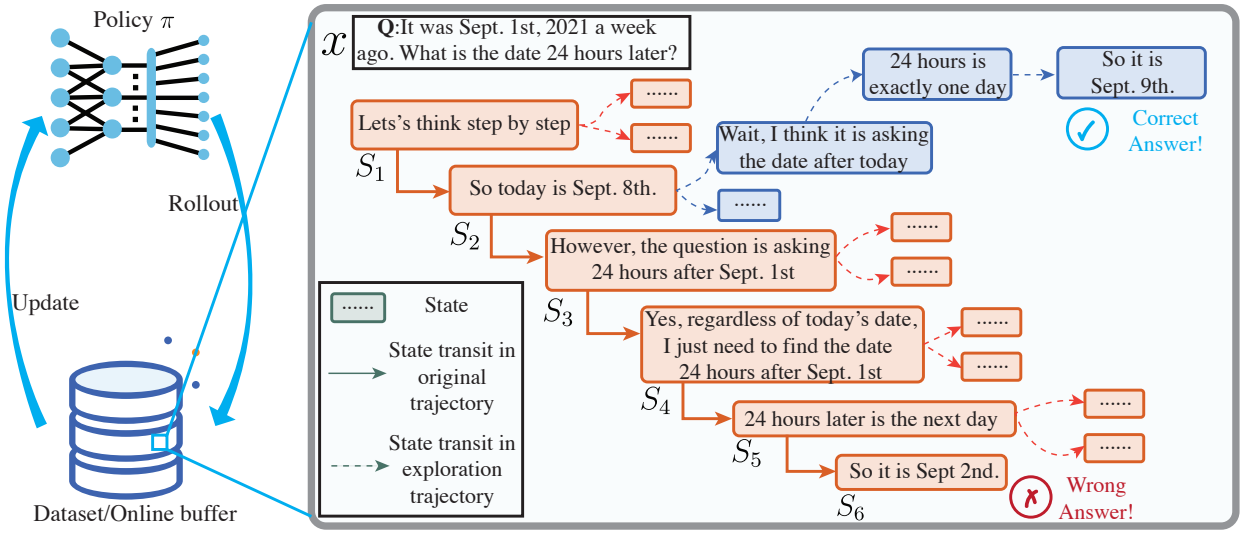


Figure 5.1: **Overview of our method.** Given an initial trajectory generated by the policy π for a reasoning task, GPO segments the trajectory into steps. It then identifies the most critical step via the MC simulation and resets the policy to the critical step to generate a new trajectory. The new trajectory is then added to the dataset or online buffer.

5.3 Method

At a high level, given a reasoning trajectory y generated by current policy π (*e.g.*, $\pi = \pi_{\text{ref}}$), we identify the most critical reasoning step y_h and refine π from this point using either an on-policy algorithm such as PPO or an offline preference method such as DPO. Intuitively, revisiting these pivotal steps enables exploration of potential alternative reasoning paths, overcoming the training bottlenecks of current policy. To illustrate the core mechanism of GPO, consider the example shown in Figure 5.1. The task is *It was Sept. 1st, 2021 a week ago. What is the date 24 hours later?*. An initial trajectory sampled from the policy π generates a multi-step reasoning process towards the question; however, it misinterprets the question and gives the wrong answer. GPO first segments this trajectory into multiple reasoning steps (*e.g.*, $S_1, \ldots S_6$). Here, the state is the sequence of all previous reasoning steps, and the action is the next reasoning step to be taken. Then, it identifies

Algorithm 4 GPO Optimization Framework

Procedure-I: Online Policy Training (PPO-based)
Input: Initial LLM policy $\pi^0 = \pi_{\text{ref}}$, reasoning dataset D_r
for iteration = 1, 2, . . . , T **do**
  $\mathcal{D} \leftarrow \emptyset$
  for n = 1, 2, . . . **do**
    Randomly sample a question x from the reasoning dataset D_r
    Run π^t to generate a K-step reasoning trajectory $y = (y_0, \ldots, y_{K-1})$ split by newlines
    Identify the critical step y_m with maximal advantage $A^{\pi^t}(x, y_{0:i-1}; y_i)$
    Reset π^t to y_m and roll-out π^t to generate trajectory $y' = (y_m, \ldots, y'_{K-1})$
    Add trajectory y' and the final reward r to $\mathcal{D}$
  end for
  Optimize π^t with respect to the policy gradient loss (*e.g.,* PPO loss) in Eqn. 5.1 on $\mathcal{D}$
end for

Procedure-II: Preference Data Generation and Optimization (DPO-based)
Input: Supervised-finetuned base policy π_{ref}, reasoning dataset D_r
$\mathcal{D} \leftarrow \emptyset$
for iteration = 1, 2, . . . , T **do**
  Repeat the sampling, trajectory generation using π_{ref}, and critical step identification as in **Procedure-I** to extract the important step y_m from trajectory y.
  Generate two continuations starting from y_m to obtain a positive trajectory $y^+ = (y_0, \ldots, y_m, \ldots, y^+_{K-1})$ and a negative trajectory $y^- = (y_0, \ldots, y_m, \ldots, y^-_{K-1})$
  Add the preference pair (x, y^+, y^-) to $\mathcal{D}$
end for
Optimize π with respect to the preference loss (*e.g.,* DPO loss) in Eqn. 5.2 on $\mathcal{D}$

the most critical reasoning step via Monte Carlo (MC) estimation of the advantage function from RL. S_2 is identified as the most critical, as the alternative continuation after S_2 could yield a correct final answer, while the continuation after other steps cannot during the MC simulation. Next, it resets the trajectory at S_2 and generates a new trajectory y' by continuing from S_2 with the current policy π, then adds the trajectory y' to the dataset or online buffer. By focusing on trajectories associated with the critical step, GPO directs the learning process towards the specific reasoning step where the policy should focus, thereby enhancing the reasoning performance of the policy.

Identifying the Critical Step via Advantage. We measure the importance of each reasoning

step using advantage functions in RL. For any candidate step y_i within a predicted reasoning trajectory y, its advantage quantifies the incremental value of taking that step, defined as the relative change in Q-value when adding y_i to the current partial sequence $y_{0:i-1}$, *i.e.,* $A^\pi(x, y_{0:i-1}; y_i) = Q^\pi(x, y_{0:i-1}; y_i) - Q^\pi(x, y_{0:i-2}; y_{i-1})$, with the previous-step term defined as 0 when $i = 0$.[1] Here, the Q-function $Q^\pi(x, y_{0:i-1}; y_i)$ estimates the expected future reward of taking action y_i after observing prefix $y_{0:i-1}$ under an auxiliary policy π. Formally, given a problem x with the gold answer y_{gold}, and a predicted trajectory y sampled from the policy π, we define:

$$Q^\pi(x, y_{0:i-1};\ y_i) = \mathbb{E}_{y^{\text{new}}_{i+1:H-1} \sim \pi(\cdot|x, y_{0:i})} \left[r\left([y_{0:i}, y^{\text{new}}_{i+1:H-1}], y_{gold} \right) \right]. \tag{5.3}$$

The reward function $r(\cdot)$ compares the completed trajectory (including sampled future steps) with the ground-truth solution y_{gold}. The policy π governs how future steps are sampled; the Q-function can be estimated without bias via MC simulations [115] by sampling multiple continuations from the current step under policy π. In §5.5.3, we will evaluate how the number of MC simulations affects the performance of GPO.

Fine-Tuning with PPO or DPO. Once the most critical reasoning step in a trajectory is identified—formally, the step with the highest advantage—we refine the policy by exploring alternative continuations from this step. Instead of treating the entire trajectory uniformly, we reset the policy to a critical step and sample new rollouts conditioned on it. As we will demonstrate in §5.4, this advantage-weighted-style sampling strategy reduces the regret of the final converged policy and enables more efficient online policy improvement. The resulting high-quality trajectories are then incorporated into the training set to guide policy updates. Our framework supports two complementary optimization methods: (i) online policy-gradient optimization like PPO, which updates

[1] In RL, the advantage function is defined as $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. In our setting, with deterministic transitions and zero intermediate rewards, this expression simplifies to the difference between consecutive Q-values: $Q(s_t, a_t) - Q(s_{t-1}, a_{t-1})$.

the policy based on reward feedback (**Procedure-I** in Algorithm 4) and (ii) offline preference optimization like DPO, which leverages pairwise preferences (**Procedure-II** in Algorithm 4).

5.4 Theoretical Analysis

In this section, we present theoretical results and insights related to Algorithm 4.

5.4.1 Online policy gradient algorithm

Before proceeding, we generalize **Procedure-I** in Algorithm 4 by sampling the critical step with probability proportional to $e^{\gamma A^{\pi^t}(s,a)}$, where $\gamma > 0$ is a temperature value. The original algorithm can be viewed as the limiting behavior for a sufficiently large γ. Since we operate in a finite-horizon episodic MDP, we evaluate the performance of the online policy gradient algorithm via its cumulative regret: *Regret* $= \sum_{t=1}^{T} \left(V_0^{\pi^*}(s_0) - V_0^{\pi^t}(s_0)\right)$. We begin by stating the following assumption regarding the Q-function.

Assumption 1 (Bounded Q-value)**.** *Suppose we have a function class $\mathcal{F}$ and $Q_h^{\pi^t} \in \mathcal{F}$ holds for the Q function of policy π^t , $\forall t = 1, 2, \ldots, T$. We assume that $0 \leq Q_h^{\pi^t}(s_h, a_h) \leq r_{\max}$ for all $Q_h^{\pi^t}(s_h, a_h) \in \mathcal{F}$, $s_h \in \mathcal{S}$, $a_h \in \mathcal{A}$.*

Assumption 1 is reasonable because reasoning tasks typically involve a bounded final reward. We further present Theorem 1 to bound the regret for the online policy gradient algorithm.

Theorem 1. *Under Assumption 1, with probability $1 - \delta$, we have the following regret bound:*

$$\sum_{t=1}^{T} \left(V_0^{\pi^*}(s_0) - V_0^{\pi^t}(s_0)\right) \leq r_{\max} H \sqrt{\frac{T \log |\mathcal{A}|}{2}} + CTH \frac{r_{\max}^2 \log(\frac{TH|\mathcal{F}|}{\delta})}{K} \sqrt{w_{\max}(\gamma)} \tag{5.4}$$

where C is a constant and $w_{\max}(\gamma)$ represents the step-wise concentrability [116] between the optimal policy π^* and our policy. In §C.7.1, we further analyze how increasing γ reduces the relevant density ratio for high-advantage actions, which motivates the advantage reweighting technique.

5.4.2 Preference optimization

For the DPO-based optimization in **Procedure-II** of Algorithm 4, we consider a conceptual variant inspired by per-step DPO [117, 66], which introduces preference comparisons at each individual reasoning step, in contrast to the standard DPO that operates over full trajectories. This step-wise modeling leads to the following result:

Theorem 2. *Let $\mathcal{D}$ consist of tuples $(x, [y_{0:i-1}, y_i^+], [y_{0:i-1}, y_i^-])$, where $y_{0:i-1} \sim \pi_{ref}$ and both continuations $y_i^{\pm}$ are drawn from $\pi_{ref}(\cdot \mid x, y_{0:i-1})$, with preferences determined by the advantage $A^{\pi_{ref}}(x, y_{0:i-1}; \cdot)$. Then, the optimal policy from minimizing Eqn. 5.2 coincides with the solution to the following advantage-weighted RL objective:*

$$\max_{\pi} \mathbb{E}_{x\sim d_0,\ y\sim\pi_{ref}(\cdot|x)} \left[\sum_{i=0}^{H-1} \log \pi(y_i \mid x, y_{0:i-1}) \cdot \exp\left(\frac{A^{\pi_{ref}}(x, y_{0:i-1}; y_i)}{\beta} \right) \right] \tag{5.5}$$

In summary, Theorem 2 shows that per-step DPO, with preference determined by the advantage function, corresponds to advantage-weighted RL, where the advantage-based preferences implicitly reweight the log-likelihood at each step. Compared to standard offline supervised fine-tuning (SFT), this reweighting allows the model to focus more on critical decision points, leading to more targeted updates and enhanced overall performance [118, 119]. We provide the proof details in Appendix C.7.2.

Table 5.1: **Comparison of GPO-enhanced methods against baselines.** The better results between GPO-enhanced and baseline methods are highlighted in bold. Each training result is the average of 3 runs with different random seeds.

Algorithms	**Test Accuracy (%)**						
	BBH	MATH	GSM8K	MMLU	MMLUPro	AIME-2024	AIME-2025
Base Model	59.97	71.60	86.50	54.09	38.80	13.33	16.67
PPO	61.82	79.60	86.96	56.66	47.47	26.67	23.33
GPO-PPO	**63.48**	**87.80**	**87.44**	**59.39**	**51.05**	**30.00**	**26.67**
DPO	63.20	82.40	86.05	57.08	48.28	20.00	20.00
GPO-DPO	**64.25**	**86.80**	**88.48**	**58.93**	**51.93**	**26.67**	**26.67**
KTO	62.86	77.20	89.31	59.42	49.02	20.00	20.00
GPO-KTO	**64.31**	**79.60**	**90.25**	**61.35**	**50.52**	**23.33**	**26.67**
SimPO	61.97	72.20	86.58	56.93	45.70	20.00	23.33
GPO-SimPO	**62.58**	**74.00**	**88.35**	**57.44**	**47.74**	**23.33**	**26.67**
ORPO	61.75	75.20	87.26	57.72	46.66	20.00	20.00
GPO-ORPO	**62.28**	**78.20**	**88.17**	**58.72**	**48.65**	**23.33**	**23.33**

5.5 Experiments

Implementation Details. We primarily employ the DeepSeek-R1-Distill-Qwen-7B model as the base model for our experiments, selected for its instruction-following and reasoning capabilities, as well as training efficiency. To further enhance data quality and reduce training costs, we follow prior work [120, 121, 67] to filter the data based on question difficulty. For advantage function estimation, we use 4 MC samples for each step. Additional implementation details are available in §C.2.2.

Baseline Methods. To evaluate the enhancement provided by GPO, we compare the performance of several established fine-tuning algorithms against their GPO-enhanced counterparts. The baseline methods include the online RL method PPO and preference-based algorithms DPO, KTO,

SimPO, and ORPO. Here, we only consider one online RL method because PPO is one of the most popular online RL methods widely used in the community, and the computation resources needed for online RL are much more significant than those for offline RL methods. Consistent with prior work on reasoning tasks [55, 121, 120], we utilize a rule-based reward function for the PPO implementation. For a fair comparison, identical hyperparameters are used for each baseline method and its corresponding GPO-enhanced version. We use the LoRA method [122] for fine-tuning. Detailed hyperparameter configurations are provided in §C.2.3.

Dataset and Evaluation Metrics. We evaluate the effectiveness of our method on 7 diverse datasets covering a range of reasoning tasks. For mathematical problem solving, we use GSM8K [123], MATH-500 [124], AIME-2024 [125], and AIME-2025 [126]. For general reasoning, we utilize BIG-Bench Hard (BBH) [127]. For STEM problem solving, we employ MMLU [128] and MMLUPro [129]. Standard train/test splits are used for GSM8K, MATH, MMLU, and MMLUPro. Following prior work [120, 130, 131], the AIME training set consists of problems from 1983-2023. For BBH, we randomly split the dataset into training set and test set by sub-task. Further dataset statistics can be found in §C.2.1. Accuracy is evaluated using zero-shot pass@1 accuracy via greedy decoding. We use different random seeds for training and report the average performance over 3 runs.

5.5.1 Main Results

The results are presented in Table 5.1. The table clearly demonstrates the effectiveness and generalizability of GPO. Across all 7 datasets and all 5 optimization algorithms, integrating GPO consistently leads to improved test accuracy compared to the respective baseline method. This consistent improvement underscores the robustness of leveraging critical step identification to enhance fine-tuning.

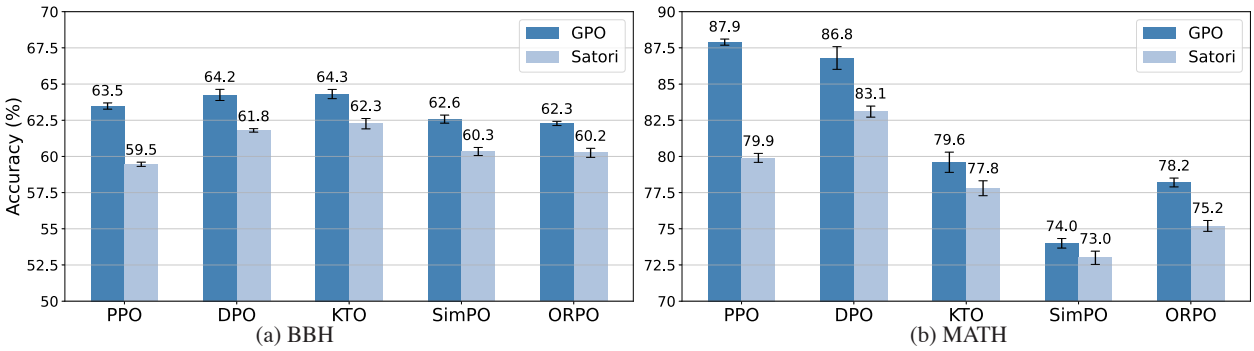


Figure 5.2: **Ablation study results on BBH and MATH.** We compare the performance of the standard GPO method and Satori's strategy that randomly identifies the critical step in the trajectory. Each bar represents the average performance of 3 runs, with error bars indicating the standard deviation.

Notably, the performance gains achieved by GPO are often substantial. For instance, GPO-PPO and GPO-DPO show significant accuracy increases on the MATH dataset compared to standard PPO and DPO. Similar positive trends are observed across other datasets like MMLUPro and the AIME benchmarks. While the magnitude of improvement varies depending on the specific dataset and baseline algorithm, the consistent improvement validates our core hypothesis: focusing learning on critical reasoning steps provides a more effective training signal, leading to enhanced reasoning capabilities in the fine-tuned models.

In addition to these general reasoning benchmarks, §C.3 reports a vulnerability patching benchmark for PatchAgent agents. This security case study evaluates whether the same critical-step-guided training idea can improve LLM agents on vulnerability repair tasks, where final success depends on locating and fixing the true root cause rather than only producing plausible code edits.

5.5.2 Ablation Study

To better understand how learning from the critical steps contributes to performance improvement, we conduct an ablation study on the BBH and MATH datasets to analyze the impact of critical step identification. Satori [67] has shown that randomly locating one step in the trajectory, then resetting and exploring the trajectory from that step, could help augment the training data and improve the performance of RLHF. Inspired by this, we randomly locate the critical step in the trajectory and compare the performance of the standard GPO method. The results are presented in Figure 5.2.

The GPO method consistently outperforms Satori's random selection baseline across both datasets. The difference is particularly obvious on MATH, where PPO with GPO achieves 87.9% accuracy, significantly higher than the 79.9% achieved when using Satori's strategy. These findings suggest that the performance gains of GPO are not merely due to the random resetting mechanism itself. Instead, the identification and learning from critical steps, leading to a more effective training signal for crucial points, is a key factor driving the improvement observed during training.

5.5.3 Monte Carlo Simulation and Model Size Scaling

To understand the scalability of GPO, we investigate its performance with respect to two key factors: the number of MC simulations used for critical step identification and the size of the base model. First, we vary the number of MC simulations from 2 to 16. Second, we apply GPO to models of varying scales, specifically the DeepSeek-R1-Distill-Qwen series (1.5B, 7B, 14B, 32B parameters) and the DeepSeek-R1-Distill-Llama-70B model. These experiments are conducted on the MATH dataset, using both DPO and KTO as the underlying optimization algorithms. The results are presented in Figure 5.3.

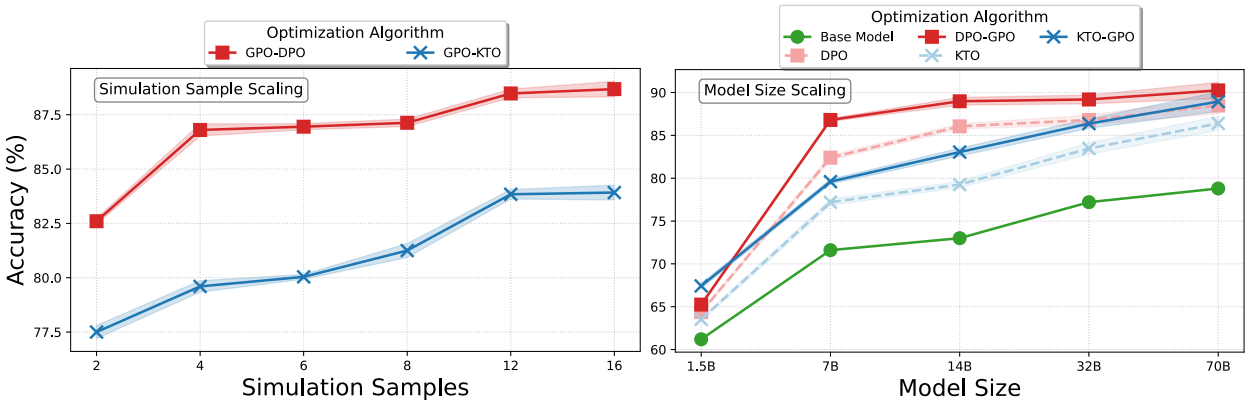


Figure 5.3: **Scaling behavior of GPO.** Performance impact of varying number of MC samples (left) and applying GPO across different model sizes (right) on MATH using DPO/KTO.

The illustration reveals that increasing the number of MC simulations generally improves the performance benefit of GPO. This suggests that a more accurate estimation of the advantage function, derived from more simulation samples, can lead to better training performance. However, the performance gains appear to saturate beyond 12 simulation samples, potentially because the estimation of the advantage function converges. It suggests that the overhead of GPO can be reduced by achieving a balance between the number of simulation samples and the performance gain.

Furthermore, the results also demonstrate that GPO consistently outperforms the corresponding baseline optimization algorithms (DPO/KTO without GPO) across all tested model sizes, from 1.5B to 70B parameters. This consistent improvement highlights the robustness of the GPO and its applicability to larger, more capable language models. Additional long-trajectory reasoning results are provided in §C.2.5.

5.6 User Study

We explore whether GPO's identification of 'critical steps' aligns with human judgments. We conduct a user study to evaluate the correlation between steps identified by GPO and those by human evaluators.

Study Setup. We designed a study involving 50 participants, recruited from college students. Participants are presented with five reasoning problems, each accompanied by a corresponding reasoning trajectory generated by the base LLM. For each trajectory, participants are asked to identify the single step they believed was the most critical point during the reasoning process for the final answer. They are presented with four optional steps for each problem: one step identified by GPO, while the other three options are randomly selected. Further details regarding the specific questions and trajectories are provided in §C.4.

Results and Discussion. The results indicate a strong alignment between the critical steps identified by GPO and human judgment. Across the five questions evaluated, the percentage of participants who selected the GPO-identified step as the most critical was 44%, 68%, 88%, 76%, and 56%, respectively. This high degree of agreement suggests that the steps pinpointed by our process are also often recognized by humans as the crucial points. These findings provide qualitative validation for the core mechanism of GPO, supporting the hypothesis that its empirical improvements come from focusing updates on critical reasoning steps.

5.7 Discussion and Limitations

While GPO shows promise in enhancing LLM reasoning, we acknowledge several limitations and areas for future work.

A key limitation is the computational overhead from the Monte Carlo estimation of future returns at each step. This adds a non-negligible cost, increasing PPO training time by approximately 1.9x and offline data preparation by 1.8x, a common challenge for related methods [66, 132, 65]. However, this trade-off is manageable. First, as shown in §5.5.3, performance gains saturate after a certain number of simulations, allowing for a practical balance between accuracy and cost. Second, to adapt GPO for very long trajectories, we employ a simple heuristic of grouping multiple generation steps into a single logical step. We validated this approach on a challenging long-context reasoning subset of BIG-Bench Extra Hard (BBEH) [133], where this simple grouping strategy enabled GPO to achieve 36.0% accuracy, a substantial +7.0 percentage-point improvement over the DPO baseline. This result confirms that GPO can be effectively scaled to complex tasks, and we provide the experiment details in §C.2.5.

Furthermore, there are several promising avenues for future research. To further enhance efficiency in the online setting, the current Monte Carlo estimation could be replaced with a more sample-efficient alternative like Generalized Advantage Estimation (GAE) [57], leveraging the value network trained by the PPO algorithm. Beyond efficiency, open questions remain regarding the identification of critical steps. For instance, could model-based explainability techniques [42, 13] offer higher fidelity compared to our model-free approach? Investigating hybrid methods, such as using powerful commercial LLMs like GPT-4o to assist in identifying critical steps, could also be a fruitful direction.

Finally, evaluating the quality of identified critical steps currently relies on downstream task performance and expensive, hard-to-scale human judgment. Benchmarks like ProcessBench [134] only focus on identifying the first incorrect step in a failed trajectory, which is not fully aligned with the scope of critical step identification. Developing automated metrics or benchmarks to reliably assess critical step quality and relevance would significantly benefit future research and

accelerate iteration on methods like GPO. We hope our work encourages further exploration of these important questions.

5.8 Conclusion

In this chapter, we introduce GPO, a strategy to enhance LLM reasoning by identifying critical steps within generation trajectories. Supported by theoretical guarantees, GPO demonstrably boosted performance across seven diverse reasoning datasets when integrated with five optimization algorithms, highlighting its generalizability. Furthermore, the user study confirms that the critical steps identified by GPO align well with human judgments of pivotal moments in reasoning failures. We believe GPO represents a valuable step towards more robust and reliable reasoning in LLMs, and we hope it inspires further research into targeted trajectory optimization and analysis.

Chapter 6

Conclusion and Future Work

This dissertation studied how explanations can make AI agents for cybersecurity more trustworthy and more capable. Security practitioners need to know which decisions mattered, why they mattered, and how those decisions can improve future behavior. The central thesis of this dissertation is that step-level explanations provide a practical interface between human trust and model improvement. Rather than treating explanations as static descriptions produced after a model acts, this dissertation uses explanations to locate the pivotal moments in an agent trajectory. Once those moments are identified, they support two complementary goals. First, they help humans audit complex AI behavior by focusing attention on the steps most responsible for success, failure, or unsafe shortcuts. Second, they create targeted learning signals: if a small number of decisions determine the final outcome, then optimization should revisit those decisions rather than treat the whole trajectory uniformly. The chapters of this dissertation develop this idea across three stages: explaining DRL agents in security applications, generalizing critical-step identification through masking, and using critical reasoning steps to improve LLM post-training.

The first contribution, presented in §3, introduced AIRS, an explanation framework for DRL-based security applications. Many security problems can be modeled as sequential decision-making tasks, but existing explainable RL methods often explain only a single action or a single

state. For cybersecurity, this is often the wrong level of abstraction. A defender or analyst usually wants to understand why an entire run succeeded or failed, not merely why the agent chose one action at one moment. AIRS addresses this gap by learning an approximation model over complete trajectories and using its interpretable step-level coefficients to identify the decisions most associated with the final reward.

The evaluation of AIRS showed that step-level explanations are useful in several security- and safety-critical settings, including selfish mining, network lateral movement, malware mutation, and autonomous driving. Across these applications, AIRS achieved stronger fidelity than approximation-based, gradient-based, and value-function-based baselines, while remaining stable and efficient enough for practical debugging. More importantly, AIRS demonstrated the operational value of explanation. By highlighting decisive steps in network lateral movement experiments, AIRS helped uncover bugs in the reward function of CyberBattleSim. This result illustrates a recurring theme of the dissertation: explanations are valuable not only because they make a model easier to describe, but because they expose the concrete decisions through which errors enter a system.

The second contribution, presented in §4, generalized this critical-step perspective through StateMask. While AIRS is effective in structured DRL environments, it still relies on an approximation model and on state representations that can be processed by that model. As agent trajectories become more complex, textual, or semantically rich, explanation methods should rely less on the internal form of a surrogate and more directly on outcome sensitivity. StateMask therefore asks a counterfactual question: if an action at a given time step is replaced with a random action, does the final outcome change? If the outcome is preserved, the step is likely non-critical; if the outcome changes, the step is important.

StateMask formulates this idea as an auxiliary reinforcement-learning problem. A mask

policy learns when to preserve the target agent's action and when to replace it, with the goal of masking as many non-critical steps as possible while preserving the target agent's expected reward. The resulting explanations outperformed several state-reward-based explanation baselines across a diverse set of environments and supported behavior understanding, adversarial analysis, and policy patching. In the security case study, StateMask-style critical-step analysis helped distinguish unsafe shortcuts, such as modifying tests to make a patch appear successful, from meaningful root-cause fixes, such as initializing vulnerable state. This result suggests that step-level explanations can reduce the cognitive burden of auditing AI-generated patches by directing reviewers to the decisions where trust is won or lost.

The third contribution, presented in §5, used critical steps not only to explain model behavior, but also to improve it. This chapter focused on LLM reasoning, where a generated solution can be viewed as a trajectory of intermediate reasoning steps. Existing post-training methods often optimize over whole trajectories. That approach can be noisy: a failed solution may contain many correct steps and one decisive error, while a successful solution may contain a shortcut that should not be reinforced. GPO addresses this problem by identifying pivotal reasoning steps through Monte Carlo estimates of step-level advantage, resetting generation to those steps, and sampling new continuations from the moments where the model's behavior most affects the final reward.

GPO turns explanation into a data synthesis mechanism. It can be integrated with online policy optimization, where new rollouts from critical steps are added to the training buffer, and with offline preference optimization, where positive and negative continuations from critical steps form preference pairs. The experiments showed that adding GPO improves performance across seven reasoning datasets and five optimization algorithms. The scaling analysis further showed consistent gains across model sizes from 1.5B to 70B parameters, and the user study indicated that the critical steps selected by GPO often align with human judgments about pivotal reasoning

moments. In the security setting, the PatchAgent vulnerability patching results show how the same idea can improve agents whose success depends on identifying root causes and making trustworthy code changes.

Together, AIRS, StateMask, and GPO form a progression from explanation to improvement. AIRS demonstrates that security analysts benefit from local step-level explanations of DRL trajectories. StateMask removes some of the dependence on structured surrogate models by grounding explanation in counterfactual outcome sensitivity. GPO then closes the loop by using pivotal steps as targets for post-training. Across these chapters, explanation is not a separate layer added after capability is built. It is a mechanism for locating where capability, failure, and trust are decided.

This dissertation also opens several directions for future work. One direction is scaling critical-step identification to longer and more heterogeneous security workflows. Real cyber reasoning systems may combine natural-language reasoning, shell commands, code edits, static analysis tools, dynamic tests, and external documentation. In such workflows, a "step" is not always obvious: it may be a line of reasoning, a tool invocation, a file edit, a test decision, or a larger subtask. Future work should develop adaptive trajectory segmentation methods that choose the right granularity for explanation. Too fine a segmentation may overwhelm auditors with token-level details, while too coarse a segmentation may hide the decision that actually caused the outcome.

A second direction is improving the efficiency of explanation-guided optimization. Both masking-based explanation and Monte Carlo critical-step estimation require additional rollouts. This cost is manageable in many offline analysis and post-training settings, but it becomes more challenging when agents operate in expensive environments or when trajectories are very long. Future work could combine the strengths of the methods in this dissertation: approximation models like AIRS may provide efficient first-pass estimates, masking methods like StateMask may validate

high-impact candidates, and value-based estimators in GPO may guide targeted exploration during training. Such hybrid systems could reduce the number of required rollouts while preserving the fidelity of critical-step identification.

A third direction is building interactive auditing and training systems around step-level explanations. The methods developed in this dissertation identify critical decisions, but a deployed security workflow also needs interfaces that help humans act on those explanations. A reviewer should be able to inspect the highlighted step, see the surrounding context, compare alternative continuations, and decide whether the step reflects a legitimate security fix, an unsafe shortcut, or an uncertain behavior that needs more evidence. The same interface could collect richer feedback than final success or failure, such as whether a patch preserves security invariants, whether the root-cause analysis is sound, or whether the implementation is maintainable. These multi-dimensional signals could make explanation-guided optimization closer to the complexity of real security engineering.

Finally, future work should study the robustness and adversarial risks of explanation mechanisms themselves. As explanations become part of deployment and training, agents may learn to produce trajectories that appear explainable while hiding unsafe behavior elsewhere. Security applications therefore require explanation methods that are themselves auditable, stress-tested, and resistant to manipulation. This is especially important when explanations are used to justify trust in automated patches or to generate new training data.

The broader lesson of this dissertation is that trustworthy AI for cybersecurity cannot be achieved by capability scaling alone. Cybersecurity demands systems whose behavior can be inspected, challenged, repaired, and improved. Step-level explanations provide one way to meet that demand. They reveal the decisions that matter inside long trajectories, help humans focus attention on the moments where risk is concentrated, and create targeted opportunities for model improve-

ment. As AI agents become more deeply embedded in cyber defense, vulnerability repair, and security analysis, the ability to identify and learn from pivotal decisions will become increasingly important. This dissertation takes a step toward that goal by showing that explanations can be both an audit mechanism and an optimization signal, helping build AI systems that are not only more powerful, but also more accountable and reliable.

References

[1] T. Van Ede *et al.*, "Deepcase: Semi-supervised contextual analysis of security events," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 522–539.

[2] Y. Li *et al.*, "{Knowphish}: Large language models meet multimodal knowledge graphs for enhancing {reference-based} phishing detection," in *33rd USENIX security symposium (USENIX Security 24)*, 2024, pp. 793–810.

[3] Z. Yu *et al.*, "{Patchagent}: A practical program repair agent mimicking human expertise," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 4381–4400.

[4] J. Yu, X. Wu, H. Liu, W. Guo, and X. Xing, "Blockscan: Detecting anomalies in blockchain transactions," *Advances in Neural Information Processing Systems*, vol. 38, pp. 100 226–100 251, 2026.

[5] Defense Advanced Research Projects Agency. "DARPA AI cyber challenge proves promise of AI-driven cybersecurity." (Aug. 2024), (visited on 05/26/2026).

[6] Advanced Research Projects Agency for Health. "ARPA-H and DARPA challenge showcases AI's power to secure america's health care." (Sep. 2025), (visited on 05/26/2026).

[7] OpenAI. "Introducing trusted access for cyber." (Feb. 2026), (visited on 05/26/2026).

[8] OpenAI. "Scaling trusted access for cyber with GPT-5.5 and GPT-5.5-Cyber." (May 2026), (visited on 05/26/2026).

[9] Anthropic. "Project glasswing: Securing critical software for the AI era." (2026), (visited on 05/26/2026).

[10] Fortinet. “2025 cybersecurity skills gap global research report.” (2025), (visited on 05/26/2026).

[11] N. Amiet. “How we exploited CodeRabbit: From a simple PR to RCE and write access on 1m repositories.” (Aug. 2025), (visited on 05/26/2026).

[12] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[13] J. Yu, W. Guo, Q. Qin, G. Wang, T. Wang, and X. Xing, “AIRS: Explanation for deep reinforcement learning based security applications,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 69–86.

[14] Z. Cheng, J. Yu, X. Wu, W. Sun, W. Guo, and X. Xing, “StateMask: Explaining deep reinforcement learning through state mask,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 59 617–59 639.

[15] Z. Cheng, J. Yu, X. Wu, and X. Xing, “GPO: Learning from critical steps to improve LLM reasoning,” in *Advances in Neural Information Processing Systems*, vol. 38, 2025.

[16] I. Arce, “Guest editor’s introduction: Software engineering for secure systems,” in *IEEE Security & Privacy*, 2004.

[17] B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” in *IEEE Security & Privacy*, 2005.

[18] K. Hammar and R. Stadler, “Finding effective security strategies through reinforcement learning and self-play,” in *International Conference on Network and Service Management*, 2020.

[19] K. Hammar and R. Stadler, “Learning intrusion prevention policies through optimal stopping,” in *International Conference on Network and Service Management*, 2021.

[20] R. Hou *et al.*, “SQUIRRL: Automating attack discovery on blockchain incentive mechanisms with deep reinforcement learning,” in *Network and Distributed System Security Symposium (NDSS)*, 2019.

[21] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in *International Conference on Financial Cryptography and Data Security*, 2014.

[22] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security*, 2016.

[23] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[24] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," *arXiv preprint arXiv:1611.07450*, 2016.

[25] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International Conference on Machine Learning*, 2017.

[26] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "SmoothGrad: Removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.

[27] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[28] V. Petsiuk, A. Das, and K. Saenko, "RISE: Randomized input sampling for explanation of black-box models," in *British Machine Vision Conference*, 2018.

[29] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLOS ONE*, 2015.

[30] M. Fan, W. Wei, X. Xie, Y. Liu, X. Guan, and T. Liu, "Can we trust your explanations? sanity checks for interpreters in android malware analysis," *IEEE Transactions on Information Forensics and Security*, 2020.

[31] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security," in *IEEE European Symposium on Security and Privacy*, 2020.

[32] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Advances in Neural Information Processing Systems*, 2018.

[33] A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, and D. J. Rezende, "Towards interpretable reinforcement learning using attention augmented agents," in *Advances in Neural Information Processing Systems*, 2019.

[34] L. Weitkamp, E. van der Pol, and Z. Akata, "Visual rationalizations in deep reinforcement learning for atari games," in *Benelux Conference on Artificial Intelligence*, 2018.

[35] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," in *International Conference on Machine Learning*, 2018.

[36] P. Gupta *et al.*, "Explain your move: Understanding agent actions using specific and relevant feature attribution," in *International Conference on Learning Representations*, 2020.

[37] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable reinforcement learning via reward decomposition," in *IJCAI/ECAI Workshop*, 2019.

[38] Z. Lin, K.-H. Lam, and A. Fern, "Contrastive explanations for reinforcement learning via embedded self predictions," in *International Conference on Learning Representations*, 2020.

[39] S. H. Huang, K. Bhatia, P. Abbeel, and A. D. Dragan, "Establishing appropriate trust via critical states," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

[40] D. Amir and O. Amir, "Highlights: Summarizing agent behavior to people," in *International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[41] A. Jacq, J. Ferret, O. Pietquin, and M. Geist, "Lazy-MDPs: Towards interpretable reinforcement learning by learning when to act," in *International Conference on Autonomous Agents and Multiagent Systems*, 2022.

[42] W. Guo, X. Wu, U. Khan, and X. Xing, "EDGE: Explaining deep reinforcement learning policies," *Advances in Neural Information Processing Systems*, 2021.

[43] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International Conference on Machine Learning*, 2017.

[44] P. W. W. Koh, K.-S. Ang, H. Teo, and P. S. Liang, "On the accuracy of influence functions for measuring group effects," in *Advances in Neural Information Processing Systems*, 2019.

[45] L. S. Shapley, "Stochastic games," *Proceedings of the National Academy of Sciences*, 1953.

[46] A. Ghorbani and J. Zou, "Data shapley: Equitable valuation of data for machine learning," in *International Conference on Machine Learning*, 2019.

[47] N. Topin and M. Veloso, "Generation of policy-level explanations for reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2019.

[48] T. Bewley and J. Lawry, "TripleTree: A versatile interpretable representation of black box agents and their environments," in *AAAI Conference on Artificial Intelligence*, 2021.

[49] T. Bewley and F. Lecue, "Interpretable preference-based reinforcement learning with tree-structured reward functions," in *International Conference on Autonomous Agents and Multiagent Systems*, 2022.

[50] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, 2022.

[51] X. Wang *et al.*, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[52] S. Yao *et al.*, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, 2023.

[53] M. Besta *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," in *AAAI Conference on Artificial Intelligence*, 2024.

[54] A. Jaech *et al.*, "OpenAI o1 system card," *arXiv preprint arXiv:2412.16720*, 2024.

[55] D. Guo *et al.*, "DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[56] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, 2022.

[57] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[58] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, 2023.

[59] K. Ethayarajh, W. Xu, N. Muennighoff, D. Jurafsky, and D. Kiela, "KTO: Model alignment as prospect theoretic optimization," *arXiv preprint arXiv:2402.01306*, 2024.

[60] J. Hong, N. Lee, and J. Thorne, "ORPO: Monolithic preference optimization without reference model," *arXiv preprint arXiv:2403.07691*, 2024.

[61] Y. Meng, M. Xia, and D. Chen, "SimPO: Simple preference optimization with a reference-free reward," *Advances in Neural Information Processing Systems*, 2024.

[62] X. Yue *et al.*, "Mammoth: Building math generalist models through hybrid instruction tuning," *arXiv preprint arXiv:2309.05653*, 2023.

[63] L. Yu *et al.*, "MetaMath: Bootstrap your own mathematical questions for large language models," *arXiv preprint arXiv:2309.12284*, 2023.

[64] R. Ding *et al.*, "Unleashing reasoning capability of LLMs via scalable question synthesis from scratch," *arXiv preprint arXiv:2409.03313*, 2024.

[65] E. Zelikman, Y. Wu, J. Mu, and N. Goodman, "STaR: Bootstrapping reasoning with reasoning," *Advances in Neural Information Processing Systems*, 2022.

[66] A. Setlur, S. Garg, X. Geng, N. Garg, V. Smith, and A. Kumar, "Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold," *Advances in Neural Information Processing Systems*, 2024.

[67] M. Shen *et al.*, "Satori: Reinforcement learning with chain-of-action-thought enhances LLM reasoning via autoregressive search," *arXiv preprint arXiv:2502.02508*, 2025.

[68] Z. Shao *et al.*, "DeepSeekMath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[69] D. Silver *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[70] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, 2016.

[71] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[72] O. Vinyals *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, 2019.

[73] Microsoft, *Cyberbattlesim: Gamifying machine learning for stronger security and ai models*, url=https://github.com/microsoft/CyberBattleSim, 2021.

[74] A. Fuchsberger, "Intrusion detection systems and intrusion prevention systems," *Information Security Technical Report*, 2005.

[75] I. Akbari, E. Tahoun, M. A. Salahuddin, N. Limam, and R. Boutaba, "Atmos: Autonomous threat mitigation in sdn using reinforcement learning," in *Proc. of NOMS*, 2020.

[76] E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. J. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," in *Proc. of NDSS*, 2017.

[77] K. Heo, W. Lee, P. Pashakhanloo, and M. Naik, "Effective program debloating via reinforcement learning," in *Proc. of CCS*, 2018.

[78] Y. Yang, J. P. Inala, O. Bastani, Y. Pu, A. Solar-Lezama, and M. Rinard, "Program synthesis guided reinforcement learning for partially observed environments," in *Proc. of NeurIPS*, 2021.

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

[80] K. Xu, Y. Li, R. H. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *Proc. of EuroS&P*, 2019.

[81] X. Zhang *et al.*, "Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware," in *Proc. of CCS*, 2020.

[82] J. Xu, Y. Li, and R. H. Deng, "Differential training: A generic framework to reduce label noises for android malware detection," in *Proc. of NDSS*, 2021.

[83] K. Chen *et al.*, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale," in *Proc. of USENIX Security*, 2015.

[84] Q. Li, Z. Peng, Z. Xue, Q. Zhang, and B. Zhou, "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning," *arXiv preprint arXiv:2109.12674*, 2021.

[85] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *Proc. of ICML*, 2018.

[86] Z. Ling, H. Ma, Y. Yang, R. C. Qiu, S.-C. Zhu, and Q. Zhang, "Explaining alphago: Interpreting contextual effects in neural networks," *arXiv preprint arXiv:1901.02184*, 2019.

[87] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Distal explanations for model-free explainable reinforcement learning," *arXiv preprint arXiv:2001.10284*, 2020.

[88] T. Bewley and J. Lawry, "Tripletree: A versatile interpretable representation of black box agents and their environments," in *Proc. of AAAI*, 2021.

[89] G. Liu, O. Schulte, W. Zhu, and Q. Li, "Toward interpretable deep reinforcement learning with linear model u-trees," in *Proc. of ECML-PKDD*, 2018.

[90] Y. Coppens *et al.*, "Distilling deep reinforcement learning policies in soft decision trees," in *Proc. of IJCAI Workshop on XAI*, 2019.

[91] K. Zhao *et al.*, "Structural attack against graph based android malware detection," in *Proc. of CCS*, 2021.

[92] Y. Y. Lu, W. Guo, X. Xing, and W. S. Noble, "DANCE: Enhancing saliency maps using decoys," in *International Conference on Machine Learning*, 2021.

[93] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1901.

[94] D. Alvarez-Melis and T. S. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *Proc. of NeurIPS*, 2018.

[95] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. of ICLR*, 2015.

[96] C.-H. Chang, E. Creager, A. Goldenberg, and D. Duvenaud, "Explaining image classifiers by counterfactual generation," in *Proc. of ICLR*, 2019.

[97] Y. Goyal, Z. Wu, J. Ernst, D. Batra, D. Parikh, and S. Lee, "Counterfactual visual explanations," *Proc. of ICML*, 2019.

[98] Z. Li, X. Liu, Z. Cheng, Y. Chen, W. Tu, and J. Su, "Trialview: An ai-powered visual analytics system for temporal event data in clinical trials," in *Proc. of HICSS*, 2024.

[99] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. of ICML*, 2015.

[100] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[101] S. Paternain, L. Chamon, M. Calvo-Fullana, and A. Ribeiro, "Constrained reinforcement learning has zero duality gap," in *Proc. of NeurIPS*, 2019.

[102] G. Brockman *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[103] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *Proc. of ICLR*, 2018.

[104] O. Vinyals *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[105] M. Lanctot *et al.*, "Openspiel: A framework for reinforcement learning in games," *arXiv preprint arXiv:1908.09453*, 2019.

[106] D. Zha *et al.*, "Douzero: Mastering doudizhu with self-play deep reinforcement learning," in *Proc. of ICML*, 2021.

[107] L. V. Allis, "A knowledge-based approach of connect-four," *Journal of the International Computer Games Association*, 1988.

[108] M. Saunders, P. Lewis, and A. Thornhill, *Research methods for business students*. Pearson education, 2009.

[109] C. E. Jimenez *et al.*, "Swe-bench: Can language models resolve real-world github issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[110] S. Zhou *et al.*, "Webarena: A realistic web environment for building autonomous agents," *arXiv preprint arXiv:2307.13854*, 2023.

[111] Z. Yang *et al.*, "HotpotQA: A dataset for diverse, explainable multi-hop question answering," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[112] G. Team *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[113] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, 2017.

[114] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, 1952.

[115] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[116] H. Kang and M.-h. Oh, "Adversarial policy optimization for offline preference-based reinforcement learning," in *Proc. of ICLR*, 2025.

[117] X. Lai, Z. Tian, Y. Chen, S. Yang, X. Peng, and J. Jia, "Step-dpo: Step-wise preference optimization for long-chain reasoning of llms," *arXiv preprint arXiv:2406.18629*, 2024.

[118] H. Xu, X. Zhan, H. Yin, and H. Qin, "Discriminator-weighted offline imitation learning from suboptimal demonstrations," in *International Conference on Machine Learning*, PMLR, 2022.

[119] Z.-W. Hong, P. Agrawal, R. T. d. Combes, and R. Laroche, "Harnessing mixed offline reinforcement learning datasets via trajectory weighting," *arXiv preprint arXiv:2306.13085*, 2023.

[120] N. Muennighoff *et al.*, "S1: Simple test-time scaling," *arXiv preprint arXiv:2501.19393*, 2025.

[121] J. Hu, Y. Zhang, Q. Han, D. Jiang, X. Zhang, and H.-Y. Shum, "Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model," *arXiv preprint arXiv:2503.24290*, 2025.

[122] E. J. Hu *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, 2022.

[123] K. Cobbe *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[124] H. Lightman *et al.*, "Let's verify step by step," *arXiv preprint arXiv:2305.20050*, 2023.

[125] AIME, *Aime 2024 problem set*, https://huggingface.co/datasets/Maxwell-Jia/AIME_2024, 2024.

[126] AIME, *Aime 2025 problem set*, https://huggingface.co/datasets/opencompass/AIME2025, 2025.

[127] M. Suzgun *et al.*, "Challenging big-bench tasks and whether chain-of-thought can solve them," *arXiv preprint arXiv:2210.09261*, 2022.

[128] D. Hendrycks *et al.*, "Measuring massive multitask language understanding," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[129] Y. Wang *et al.*, "Mmlu-pro: A more robust and challenging multi-task language understanding benchmark," in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[130] E. Beeching *et al.*, *Numinamath 7b tir*, 2024.

[131] J. Li *et al.*, "Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions," *Hugging Face repository*, 2024.

[132] Z. Yuan *et al.*, "Scaling relationship on learning mathematical reasoning with large language models," *arXiv preprint arXiv:2308.01825*, 2023.

[133] M. Kazemi *et al.*, "Big-bench extra hard," *arXiv preprint arXiv:2502.19187*, 2025.

[134] C. Zheng *et al.*, "Processbench: Identifying process errors in mathematical reasoning," *arXiv preprint arXiv:2412.06559*, 2024.

[135] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[136] S. Thrun, "Monte carlo pomdps," in *Proc. of NeurIPS*, 1999.

[137] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. of ICLR*, 2017.

[138] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, pp. 9–44, 1988.

[139] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[140] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. of NeurIPS*, 2019.

[141] Y. Guan *et al.*, "Perfectdou: Dominating doudizhu with perfect information distillation," in *Proc. of NeurIPS*, 2022.

[142] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[143] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proc. of AAAI*, vol. 31, 2017.

[144] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[145] C. Yu, J. Liu, S. Nemati, and G. Yin, "Reinforcement learning in healthcare: A survey," *ACM Computing Surveys*, vol. 55, no. 1, pp. 1–36, 2021.

[146] Y. Zheng *et al.*, "Llamafactory: Unified efficient fine-tuning of 100+ language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 2024.

[147] J. D. Chang *et al.*, "Dataset reset policy optimization for rlhf," *arXiv preprint arXiv:2404.08495*, 2024.

[148] Y. Song, Y. Zhou, A. Sekhari, D. Bagnell, A. Krishnamurthy, and W. Sun, "Hybrid RL: Using both offline and online data can make RL efficient," in *Proc. of ICLR*, 2023.

[149] X. Liu *et al.*, "Active advantage-aligned online reinforcement learning with offline data," *arXiv preprint arXiv:2502.07937*, 2025.

Appendix A

Supplementary Material for AIRS

A.1 Representative Applications

This appendix summarizes the four applications used to evaluate AIRS. They are kept here rather than in the main chapter so that the main text can focus on the explanation method and its evaluation.

Autonomous driving. We use MetaDrive [84] as the autonomous-driving environment. The agent observes a vector representation of the road, nearby objects, navigation information, and vehicle status, and outputs driving controls such as steering, acceleration, and braking. The reward encourages forward progress and successful arrival while penalizing collisions and traffic-rule violations. This application represents long-horizon safety-critical control, where the final reward depends on many small decisions.

Malware mutation. We use HRAT-style DRL-based Android malware mutation [91]. The agent observes the malware call graph and chooses semantics-preserving mutation actions, including adding function calls, rewiring function calls, inserting methods, and deleting methods. The goal

is to evade a malware detector with minimal graph manipulation. The reward is:

$$R = \begin{cases} 1, & \text{if the mutated sample bypasses } f(\cdot), \\ -(\Delta N_{\text{edge}} + \Delta N_{\text{node}}), & \text{otherwise,} \end{cases} \tag{A.1}$$

where ΔN_{edge} and ΔN_{node} are the numbers of manipulated edges and nodes. The reward therefore favors successful evasion with fewer modifications.

Selfish mining. Selfish mining [21, 20, 22] models a strategic miner that withholds blocks to gain more mining reward than honest mining. The state is represented as $(a, h, fork)$, where a is the length of the attacker's private chain, h is the length of the public chain after the latest fork, and $fork$ records the fork status. The agent chooses among adopting, waiting, overriding, and matching. The reward is determined by the blocks eventually accepted into the longest chain.

Network lateral movement. We use CyberBattleSim [73] for network lateral movement. The attacker starts from an initially compromised node and learns a policy for discovering nodes, exploiting vulnerabilities, and moving through the network. We evaluate two topology settings: Chain-Graph and Random-Graph. In Chain-Graph, nodes are arranged in a chain and vulnerabilities are fixed by node type. In Random-Graph, topology, vulnerabilities, and node values are randomly generated. The attacker receives rewards for compromising valuable nodes and a large reward for taking over the whole network.

A.2 Additional Experimental Setting

The default episode lengths are 15 for selfish mining, 100 for lateral movement, 1000 for autonomous driving, and 500 for malware mutation. For a fair comparison, all methods use the same step filter: a continuous sub-sequence with the highest correlation to final reward is retained, while

	Selfish Mining	Chain-Graph	Random-Graph	Safe Driving	Malware Mutation
Agent	DQN	DQN	DQN	PPO	DQN
$\theta(\cdot)$	FC(8,4,1)	FC(8,4,1)	FC(32,8,1)	FC(32,16,8,1)	FC(8,1)
$g(\cdot)$	FC(8,4,1)	FC(8,4,1)	FC(32,8,1)	FC(32,16,8,1)	FC(8,1)
Epoch	50	100	100	200	100

Table A.1: **AIRS experimental setup.** The table reports the DRL algorithm, the approximation-model architecture, and the number of training epochs. FC denotes fully connected layers.

Explainer	Random	Vanilla	Integrated	Smooth	Attention	LR	Highlights	Trust	AIRS
Reward Difference	1443.98	1395.43	1392.88	1345.50	1487.34	1394.87	1491.11	1421.42	**1539.05**
RRD	1.00000	0.96637	0.96461	0.93179	1.03003	0.96598	1.03264	0.98438	**1.06584**

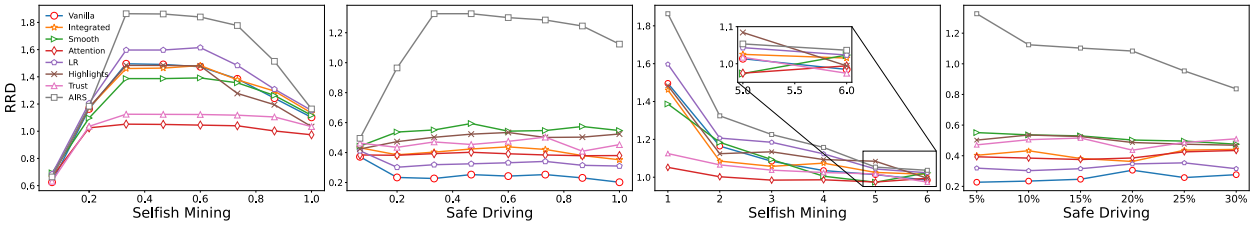
Table A.2: **Random-Graph fidelity detail.** AIRS has the highest reward difference and RRD in the random-graph lateral-movement setting.

other steps are filtered out. Unless otherwise stated, the retained sub-sequence length is 50% of the episode length.

Table A.1 summarizes the architecture and training setup for the AIRS approximation model. We train the approximation model with Adam, learning rate 10^{-3}, and stability coefficient $\lambda = 10^{-6}$. The experiments were conducted on a server with 2 AMD EPYC 7702 64-Core CPU processors and 4 NVIDIA RTX A6000 GPUs.

A.3 Ablation Study

Figure A.1 studies two hyperparameters. First, we vary the retained length of the filtered sub-sequence. For selfish mining, performance remains strong when the retained length is moderate, but drops when the filter is too aggressive or when no filtering is applied. This shows that correlation-based filtering is useful as a pre-selection step, but should not replace the learned AIRS importance model. In safe driving, AIRS follows a similar trend, while other methods are less re-



(a) vary length of filtered subsequence

(b) vary number of perturbed steps

Figure A.1: **AIRS ablation results.** The left plot varies the retained length of the filtered subsequence. The right plot varies the number of selected critical steps that are perturbed.

Explainer	Selfish Mining	Chain-Graph	Random-Graph	Safe Driving	Malware Mutation
AIRS	143.94	194.07	448.35	790.63	225.20
Exhaustive Search	1074.72	7690.46	13524.75	512175.64	97106.32

Table A.3: **Exhaustive-search efficiency.** Time is reported in seconds for generating explanations for 500 episodes. The AIRS time includes both training and generation.

sponsive because they often fail to identify the truly critical steps.

Second, we vary the number of perturbed steps. As more steps are perturbed, RRD drops quickly. When too many actions are randomized, the trajectory can fail for reasons unrelated to the selected explanation, especially in sensitive long-horizon tasks. This supports the design choice of using AIRS to identify a single critical step or a short critical sub-sequence, which is also easier for human analysts to inspect.

A.4 Exhaustive Search

An intuitive alternative to AIRS is exhaustive perturbation: replay an episode many times, randomize each candidate step, and directly measure the reward change. This approach can identify important steps, but it is too expensive for long episodes or environments where interaction is costly.

Table A.3 shows that exhaustive search is much slower than AIRS. The gap is already substantial for short episodes and becomes impractical for safe driving and malware mutation. This comparison motivates learning an approximation model once and generating step-level explanations through efficient forward passes.

Appendix B

Additional Material for StateMask

B.1 Proof of Theorem 1

Before proving Theorem 1, we first introduce the following lemma.

Lemma 2. *Given an old and new policy of the state mask $\tilde{\pi}_{\theta_{old}}$ and $\tilde{\pi}_\theta$, respectively, if $\eta(\tilde{\pi}_{\theta_{old}}) \leq \eta(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})$, we have $|\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})| \leq |\eta(\tilde{\pi}_{\theta_{old}}) - \eta(\bar{\pi})|$.*

Proof of Lemma 2. The monotonicity property requires $|\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})| \leq |\eta(\tilde{\pi}_{\theta_{old}}) - \eta(\bar{\pi})|$. The following equivalence holds:

$$\begin{aligned} & |\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})| \leq |\eta(\tilde{\pi}_{\theta_{old}}) - \eta(\bar{\pi})| \\ \iff & \eta(\tilde{\pi}_\theta)^2 - 2\eta(\tilde{\pi}_\theta)\eta(\bar{\pi}) + \eta(\bar{\pi})^2 \leq \eta(\tilde{\pi}_{\theta_{old}})^2 - 2\eta(\tilde{\pi}_{\theta_{old}})\eta(\bar{\pi}) + \eta(\bar{\pi})^2 \\ \iff & \left(\eta(\tilde{\pi}_\theta) + \eta(\tilde{\pi}_{\theta_{old}})\right)\left(\eta(\tilde{\pi}_\theta) - \eta(\tilde{\pi}_{\theta_{old}})\right) \leq 2\eta(\bar{\pi})\left(\eta(\tilde{\pi}_\theta) - \eta(\tilde{\pi}_{\theta_{old}})\right) . \end{aligned} \tag{B.1}$$

Therefore, if $\eta(\tilde{\pi}_{\theta_{old}}) \leq \eta(\tilde{\pi}_\theta)$ and $\eta(\tilde{\pi}_\theta) + \eta(\tilde{\pi}_{\theta_{old}}) \leq 2\eta(\bar{\pi})$, or in other words, $\eta(\tilde{\pi}_{\theta_{old}}) \leq \eta(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})$, we have $|\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})| \leq |\eta(\tilde{\pi}_{\theta_{old}}) - \eta(\bar{\pi})|$. □

Recall that Lemma 1 in §4.2.2 shows the upper bound and the lower bound of $\eta(\tilde{\pi}_\theta)$. An immediate implication of Lemma 1 enables us to obtain the inequalities to satisfy Lemma 2.

From Lemma 1, we can easily derive the lower bound of $\eta(\tilde{\pi}_\theta)$ as $\eta(\tilde{\pi}_\theta) \geq L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) - C\mathbb{KL}^{\max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))$. If the lower bound is greater than $\eta(\tilde{\pi}_{\theta_{old}})$, then we can guarantee that $\eta(\tilde{\pi}_\theta) \geq \eta(\tilde{\pi}_{\theta_{old}})$.

Similarly, we can obtain the upper bound of $\eta(\tilde{\pi}_\theta)$ as $\eta(\tilde{\pi}_\theta) \leq L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + C\mathbb{KL}^{\max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))$. If the upper bound is lower than $2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})$, we can guarantee that $\eta(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})$.

Putting all these inequalities together, the following inequalities are the sufficient conditions for satisfying $|\eta(\tilde{\pi}_\theta) - \eta(\bar{\pi})| \leq |\eta(\tilde{\pi}_{\theta_{old}}) - \eta(\bar{\pi})|$:

$$\begin{aligned} & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \geq \eta(\tilde{\pi}_{\theta_{old}}) + C\mathbb{KL}^{\max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)) \\ & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + C\mathbb{KL}^{\max}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) . \end{aligned} \tag{B.2}$$

B.2 Derivation of Eqn. (5)

First, let $M_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) = L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) - C\mathbb{KL}^{\max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))$. We can get $M_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_{\theta_{old}}) = \eta(\tilde{\pi}_{\theta_{old}})$. By maximizing the $M_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)$, we can ensure $M_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \geq M_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_{\theta_{old}}) = \eta(\tilde{\pi}_{\theta_{old}})$. Given that $\eta(\tilde{\pi}_\theta) \geq M_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)$, we could satisfy the first inequality of Eqn. (4).

By constraining the upper bound of $\tilde{\pi}_{\theta_{old}}$ to be smaller than $2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})$, we could guarantee the second inequality of Eqn. (4) holds.

Therefore, combining all these constraints together, we can obtain the following optimization objective:

$$\begin{aligned} \max_\theta \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) - C\mathbb{KL}^{\max}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)) \\ \text{s.t.} \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + C\mathbb{KL}^{\max}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) . \end{aligned} \tag{B.3}$$

In practice, Schulman et al. [57] pointed out that utilizing the penalty coefficient C may lead to a very small step size. Therefore, we replace the penalty on the maximal KL-divergence with a trust region constraint on an average KL-divergence (*i.e.,* $\mathbb{E}_{s \sim \rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))] \leq \delta$) as

suggested in [57]. We can further obtain the following objective

$$\begin{aligned} \max_{\theta} \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \\ \text{s.t.} \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta, \\ & \mathbb{E}_{s\sim\rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))] \leq \delta\,. \end{aligned} \tag{B.4}$$

Additionally, we add a sparsity constraint which will only tighten the problem without affecting the monotonicity property and obtain the following optimization objective

$$\begin{aligned} \max_{\theta} \quad & L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \\ \text{s.t.} \quad & \textcircled{1} \quad \mathbb{E}_{a^e\sim\tilde{\pi}_\theta}[a^e] \geq c, \\ & \textcircled{2} \quad L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta, \\ & \textcircled{3} \quad \mathbb{E}_{s\sim\rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s))] \leq \delta, \end{aligned} \tag{B.5}$$

where $\mathbb{E}_{a^e\sim\tilde{\pi}_\theta}[a^e] \geq c$ constrains the lower bound sparsity for the output of $\tilde{\pi}_\theta$ across all states. By maximizing the local approximation with constraint ③, we could guarantee that $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \geq L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_{\theta_{old}}) = \eta(\tilde{\pi}_{\theta_{old}})$. Since $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)$ is the local approximation of $\eta(\tilde{\pi}_\theta)$, we could further guarantee that $\eta(\tilde{\pi}_\theta) \geq \eta(\tilde{\pi}_{\theta_{old}})$. Moreover, the constraint $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta$ could ensure that the inequality $\eta(\tilde{\pi}_\theta) \leq 2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}})$ is satisfied. Therefore, a policy $\tilde{\pi}_\theta$ solved from the optimization objective above satisfies Lemma 2 and thus enables the desired monotonicity.

B.3 Derivation from Eqn. (6) to Eqn. (7)

With respect to θ, Eqn. (6) becomes

$$\max_{\theta} \; (1-\lambda)L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + C_1 \quad \text{s.t.} \; \mathbb{E}_{s\sim\rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right)] \leq \delta, \quad \mathbb{E}_{a^e\sim\tilde{\pi}_\theta}[a^e] \geq c\,, \tag{B.6}$$

where $C_1 = \lambda(2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta)$ is a constant that can be eliminated during the optimization of θ.

The KKT condition [135] implies that there exists Lagrange multiplier ξ (dual solution) such that the set of solutions of the constrained optimization problem Eqn. (B.6) are equivalent to those of the following relaxed objective:

$$\max_{\theta} \ (1-\lambda) L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + \xi \mathbb{E}_{a^e \sim \tilde{\pi}_\theta}\left[a^e\right] \quad \text{s.t.} \ \mathbb{E}_{s \sim \rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right)] \leq \delta. \tag{B.7}$$

If $\lambda < 1$, the optimization problem is equivalent to

$$\max_{\theta} \ L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) + w \mathbb{E}_{a^e \sim \tilde{\pi}_\theta}\left[a^e\right] \quad \text{s.t.} \ \mathbb{E}_{s \sim \rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right)] \leq \delta, \tag{B.8}$$

where $w = \xi/(1-\lambda)$.

The optimization problem can further be transformed as

$$\begin{aligned} &\max_{\theta} \sum\nolimits_s \rho_{\tilde{\pi}_{\theta_{old}}}(s) \sum\nolimits_{a^e} \tilde{\pi}_\theta(a^e \mid s) A_{\tilde{\pi}_{\theta_{old}}}(s, a^e) + w \mathbb{E}_{a^e \sim \tilde{\pi}_\theta}\left[a^e\right] \\ &\text{s.t.} \quad \mathbb{E}_{s \sim \rho_{\tilde{\pi}_{\theta_{old}}}}[\mathbb{KL}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right)] \leq \delta. \end{aligned} \tag{B.9}$$

Due to the summation of the new policy $\tilde{\pi}_\theta$, optimization is still challenging to implement even after transformation. To resolve this issue, we follow the idea from [99]. Specifically, we use importance sampling and substitute it with the summation of the old policy through the Monte Carlo method [136]. Then, in order to reach our ultimate optimization function, we further substitute the trust region constraints with the clipped ratio operation proposed by the Proximal Policy Optimization algorithm (PPO) [57]. Therefore, if $\lambda < 1$, we could obtain the final objective to optimize θ

as

$$\max_\theta \mathbb{E}\left[\min\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)} A_{\tilde{\pi}_{\theta_{old}}}, \text{clip}\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) A_{\tilde{\pi}_{\theta_{old}}}\right) + wL_t^{\text{MASK}}\right], \tag{B.10}$$

where $L_t^{\text{MASK}} = \mathbb{E}_{a_t^e \sim \tilde{\pi}_\theta}\left[a_t^e\right]$ is added to ensure the sparsity of critical steps.

If $\lambda > 1$, the optimization problem is equivalent to

$$\min_\theta L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) - w\mathbb{E}_{a^e \sim \tilde{\pi}_\theta}\left[a^e\right] \quad \text{s.t.} \quad \mathbb{E}_{s \sim \rho_{\tilde{\pi}_{\theta_{old}}}}\left[\mathbb{KL}\left(\tilde{\pi}_{\theta_{old}}(\cdot \mid s) \| \tilde{\pi}_\theta(\cdot \mid s)\right)\right] \leq \delta. \tag{B.11}$$

Similar to the analysis above, if $\lambda > 1$, the final objective for updating θ becomes

$$\max_\theta \mathbb{E}\left[-\min\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)} A_{\tilde{\pi}_{\theta_{old}}}, \text{clip}\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) A_{\tilde{\pi}_{\theta_{old}}}\right) + wL_t^{\text{MASK}}\right], \tag{B.12}$$

Putting all these objectives together, we get the final objective for updating θ as

$$\max_\theta \mathbb{E}\left[\text{sgn}(1-\lambda)\min\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)} A_{\tilde{\pi}_{\theta_{old}}}, \text{clip}\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) A_{\tilde{\pi}_{\theta_{old}}}\right) + wL_t^{\text{MASK}}\right]. \tag{B.13}$$

B.4 Our Training Algorithm

B.4.1 Algorithm Detail

Optimizing θ. Recall that the final objective function of θ is

$$\max_\theta \mathbb{E}\left[\text{sgn}(1-\lambda)\min\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)} A_{\tilde{\pi}_{\theta_{old}}}, \text{clip}\left(\frac{\tilde{\pi}_\theta\left(a_t^e \mid s_t\right)}{\tilde{\pi}_{\theta_{\text{old}}}\left(a_t^e \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) A_{\tilde{\pi}_{\theta_{old}}}\right) + wL_t^{\text{MASK}}\right], \tag{B.14}$$

where $A_{\tilde{\pi}_{\theta_{old}}} = A_{\tilde{\pi}_{\theta_{old}}}(s_t, a_t^e)$ and $L_t^{\text{MASK}}(\theta) = \mathbb{E}_{a_t^e \sim \tilde{\pi}_\theta}[a_t^e]$.

Note that the output of StateMask a_t^e is sampled from a categorical distribution over all actions (*i.e.,* 0 or 1). However, as stated in [137], the categorical variable a_t^e is not capable of backpropagating through samples. Note that $\mathbb{E}_{a_t^e \sim \tilde{\pi}_\theta}[a_t^e] = \mathbb{E}_{s_t \sim \rho_{\tilde{\pi}_\theta}}\left[\mathbb{E}_{a_t^e \sim \tilde{\pi}_\theta(\cdot|s)}[a_t^e]\right] = \mathbb{E}_{s_t \sim \rho_{\tilde{\pi}_\theta}}[Pr(a_t^e = 1)]$ and $Pr(a_t^e = 1)$ is differentiable. Therefore, we replace $\mathbb{E}_{a_t^e \sim \tilde{\pi}_\theta}[a_t^e]$ with $\mathbb{E}_{s_t \sim \rho_{\tilde{\pi}_\theta}}[Pr(a_t^e = 1)]$ in practice.

When estimating $A_{\tilde{\pi}_{\theta_{old}}}$, we adopt the same estimation formula as PPO [57]

$$\begin{aligned} &\hat{A}_{\tilde{\pi}_{\theta_{old}}} = \delta_t + (\gamma\zeta)\delta_{t+1} + \cdots + \cdots + (\gamma\zeta)^{T-t+1}\delta_{T-1} \\ &\text{where} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{B.15}$$

Therefore, during the training process, in addition to updating the actor by solving θ in Eqn. (B.14), we also have to update the state-value function $V(s)$ by the Temporal-Difference (TD) method [138] for better estimating the advantage.

Optimizing λ. And recall that the objective function of λ is

$$\min_{\lambda \geq 0} \lambda \left(2\eta(\bar{\pi}) - \eta(\tilde{\pi}_{\theta_{old}}) - \delta - L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)\right). \tag{B.16}$$

where $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta) = \sum_s \rho_{\tilde{\pi}_{\theta_{old}}}(s) \sum_{a^e} \tilde{\pi}_\theta(a^e \mid s) A_{\tilde{\pi}_{\theta_{old}}}(s, a^e)$.

In practice, $\eta(\tilde{\pi}_{\theta_{old}})$ and $\eta(\bar{\pi})$ can be approximated via Monte Carlo method [139]. $\eta(\bar{\pi})$ can be estimated in advance, *i.e.,* running the game with the agent's policy $\bar{\pi}$ 500 times and calculating the average expected discounted reward as follows

$$\eta(\bar{\pi}) = \mathbb{E}_{s_0, a_0, \ldots \sim \bar{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \tag{B.17}$$

$\eta(\pi_{\theta_{old}})$ can be estimated using a similar formula in parallel with collecting trajectories under the old policy $\pi_{\theta_{old}}$ while solving θ with our training algorithm.

Nevertheless, the optimization objective remains challenging to solve because of the summation in $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)$ involving the new policy $\tilde{\pi}_\theta$. To address this issue, we utilize importance sampling and substitute the summation over the new policy $\tilde{\pi}_\theta$ with a summation over the old policy $\tilde{\pi}_{\theta_{old}}$, which can be computed using the Monte Carlo method [139]. Mathematically, $L_{\tilde{\pi}_{\theta_{old}}}(\tilde{\pi}_\theta)$ can be estimated by $\frac{\tilde{\pi}_\theta(a^e|s)}{\tilde{\pi}_{\theta_{\text{old}}}(a^e|s)} A_{\tilde{\pi}_{\theta_{old}}}(s, a^e)$, which has already been computed in Eqn. (B.14) when updating θ.

Extensive-form Games. Regarding extensive-form games, we train a state mask for the target agent while keeping all the agents' policies fixed. In this way, we could transform the extensive-form games into single-player normal-form games (*i.e.,* we treat all of the agents' actions as part of the environment). However, the game may not end with the target agent, and the target agent will not receive the game reward in this case. To address this problem, we automatically overwrite the target agent's final reward with the final game result.

Based on what we have discussed above, we present the full algorithm in Algorithm 5.

B.4.2 Implementation and Hyperparameters

We implement the training algorithm using PyTorch [140]. The common hyperparameter setting is as follows. Regarding the clipping parameter ϵ in Eqn. (B.14), we set it as 0.2. As for the advantage estimation in Eqn. (B.15), we set $\gamma = 0.99$ and $\zeta = 0.95$. Moreover, Table B.1 provides the game-specific hyperparameter choices. Regarding the policy network of StateMask, we utilize a widely used network structure, commonly employed to train Deep Reinforcement Learning (DRL) agents for each game. Specifically, we use Convolutional Neural Network (CNN) for the Pong game and Residual Network (ResNet) for the Connect 4 game since these two games' input

Algorithm 5 The training algorithm of mask net.

Input: Target agent's policy $\bar{\pi}$
Output: Mask net policy $\tilde{\pi}$
Initialization: Initialize the weights θ for the mask net $\tilde{\pi}$, Lagrange multiplier $\lambda = 0$
Estimate the $\eta(\bar{\pi})$ in advance by Eqn. (B.17)
for $iteration = 1, 2, \ldots$ **do**
  $s \leftarrow s_0$
  $\mathcal{D} \leftarrow \emptyset$
  for $t = 1, 2, ..., T$ **do**
    $a_t^e \leftarrow$ sample from $\tilde{\pi}_{\theta_{old}}(\cdot|s)$
    $a_t \leftarrow$ sample from $\bar{\pi}(\cdot|s)$
    Compute actual taken action $a \leftarrow a_t \odot a_t^e$
    Obtain current step reward r_t, next state s' from the environment given action a
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s, a_t^e, r_t, s')$
    $s \leftarrow s'$
  end for
  Compute estimates for $A_{\tilde{\pi}_{\theta_{old}}}(s_t, a_t^e)$ using samples in $\mathcal{D}$ by Eqn. (B.15)
  Solve θ using the trajectory samples in $\mathcal{D}$ by Eqn. (B.14) and update $\theta_{old} \leftarrow \theta$
  Estimate the $\eta(\tilde{\pi})$ using the trajectory samples in $\mathcal{D}$ by Eqn. (B.17)
  Solve λ by Eqn. (B.16) and update $\lambda_{old} \leftarrow \lambda$
end for

observations contain spatial information and are relatively complex. For other games, we use either Long Short-Term Memory (LSTM) or Multi-Layer Perceptron (MLP) network, which depends on the input type of the observation and the game's complexity. We use two separate networks with the same backbone architecture for the policy and value networks. The output dimension of the policy network and value network are 2 and 1, respectively. As for the learning rate setting, we refer to other work on training reinforcement learning agents in these games [105, 141]. Moreover, StateMask introduces an extra hyperparameter – w (To encourage the mask network to blind the target agent at some states, we add a sparsity constraint on the number of mask actions). We search over $w \in \{0, 1e-5, 1e-4, 1e-3, 1e-2\}$.

Table B.1: **Training hyperparameter settings of StateMask.** The numbers in the bracket after "CNN" represent the number of kernels in each layer. The numbers in the bracket after "MLP" are the hidden dimensions. 512×10 represents 10-layer MLP with hidden size 512.

Games	w	Learning rate	Optimizer	StateMask backbone	Batch Size	Num. of Training Steps
Pong	1e-4	1e-5	Adam	CNN(16, 32)	32	2e+4
You-Shall-Not-Pass	0	3e-4	Adam	MLP(256, 256)	2048	2e+7
Kick-And-Defend	0	3e-4	Adam	LSTM(512, 256)	2048	2e+7
CartPole	0	1e-3	Adam	MLP(256, 256)	5	8e+2
Pendulum	0	1e-3	Adam	MLP(256, 256)	5	8e+2
StarCraft II	0	1e-6	Adam	MLP(128,128,128)	32	8e+7
Connect 4	1e-3	1e-3	Adam	ResNet-128	1024	4e+5
Tic-Tac-Toe	1e-5	1e-2	Adam	MLP(128, 128)	128	2e+4
Breakthrough	1e-3	1e-3	Adam	MLP(512×10)	1024	2e+6
DouDizhu	1e-3	3e-4	Adam	MLP(512×10)	42	8e+5

B.5 Details of Evaluation

B.5.1 Other Implementation Details

Baseline Implementations. Regarding baseline approaches, we use the code released by the authors or implement our own version if the authors do not release the code. Specifically, for EDGE, we use the released open-source implementation from https://github.com/Henrygwb/edge. For Value-max and LazyMDP, we assume access to the target agent's value function and implement the explainers based on the corresponding method descriptions[1].

Fidelity Tests and Metrics. The fidelity test uses the evaluation design from EDGE [42] to assess explanation accuracy. Specifically, the fidelity test first gathers N (*i.e.,* N = 500 in our setting) trajectories. Each explanation method generates the importance score of each state accordingly without altering the agent's action. For each trajectory, we identify and record the length l of the longest continuous sequence within the top-K most critical time steps. Then, we replay the sequence of actions from the original trajectory up to the start of this identified sequence. From

[1]Value-max is straightforward to implement because it only requires extracting the state value from the value function. For LazyMDP, we contacted the author and implemented the explainer based on the author's response.

this point onwards, we replace the original actions with randomly selected actions for the next l steps. Once the action substitution is completed, we present the newly reached state to the target agent at the end of the selected time steps. From this point forward, the agent operates according to the actions suggested by its policy network until the conclusion of the game. The reward difference before and after action replacement is denoted as d. The fidelity score is defined as $\log(p_d) - \log(p_l)$ where $p_d = |d|/d_{max}$ is the absolute reward difference divided by the maximum possible reward difference and $p_l = l/L$ is the length of the longest sequence normalized by the original trajectory length L. In the EDGE definition of the fidelity score [42], a lower score implies a higher level of fidelity. For ease of presentation, this chapter uses the negative score, such that a higher score indicates superior fidelity.

B.5.2 Experiment Setup

Game Introduction and Target Agents. To support the interpretation of the evaluation results, this appendix briefly introduces each game below.

- **Pong** is a two-dimensional sports game that simulates table tennis as Figure B.1 shows. The RL agent controls the right paddle while a decent AI from the Atari game environment controls the left paddle. The RL agent observes itself, the opponent, and the position of the ball before deciding whether to move the paddle up, down, or take no action. In each round, the RL agent wins the game and collects a +1 reward if the opponent fails to return the ball; the agent loses the game and collects a 0 reward if it misses the ball. To facilitate the evaluation, we terminate the game at the end of each round.
- **You-Shall-Not-Pass** is a two-party competitive game as Figure B.6 shows. Two humanoid robots face each other at the beginning of the game. The red robot then starts running towards the finish line (indicated by the red line), while the blue robot is tasked with preventing the runner

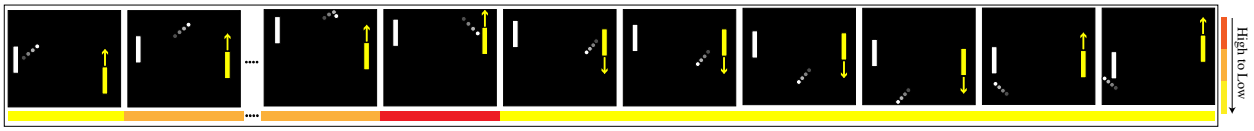


Figure B.1: Visualization of the identified critical time steps in Pong game. The DRL agent controls the yellow paddle and a non-RL rule-based opponent controls the white paddle.

from achieving its goal. In our evaluation, the red robot wins the game if it can get across the finish line. Otherwise, the blue blocker wins. The red robot gets the reward +1 once it wins the game and -1 when it loses the game.

- **Kick-And-Defend** is a two-party competitive game as Figure B.7 shows. Two humanoid robots compete in a soccer penalty shootout. The placements of kicker, defender, and ball are all chosen at random. If the ball falls between the goalposts, the kicker wins; otherwise, the defender wins. The reward is +1 for the kicker to win the game and -1 when the kicker loses.
- **CartPole** is one of OpenAI's open-source environments as Figure B.8 shows. The "cartpole" agent is a reverse pendulum in which a "cart" attempts to vertically balance a "pole" with a slight angle adjustment. The only forces that can be used are represented by +1 and -1, corresponding to a left and right movement, respectively. The episode ends if the cart moves more than 2.4 units away from the center or if the angle deviates from vertical by 15 degrees. For each time step before the episode ends, the reward is increased by one. The maximum reward available is 500.
- **Pendulum** simulates the inverted pendulum swingup problem as Figure B.9 shows. The pendulum starts in a random position, and the goal is to swing it up and keep it upright. The pendulum angle is represented by θ, which starts at random from $-\pi$ to π. The default reward function is based on the angle θ and the agent's action a. To maximize rewards, the agent must keep the angle θ at zero, rotate at the slowest possible speed, and exert the least amount of effort.
- **StarCraft II** is a popular real-time strategy game as Figure B.10 shows. The player controls

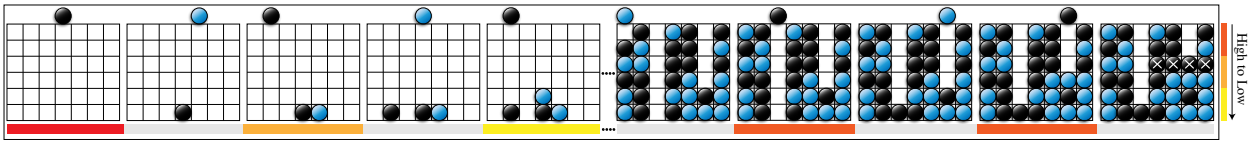


Figure B.2: Visualization of the identified critical steps in the Connect 4 game. The black player is the target player. Note that the states where the blue player takes actions are marked with a gray bar.

an army to defeat the opponent's base by collecting resources and building units. The player's observation is the minimap and the screen. The minimap shows the entire map, while the screen shows the player's view. The player can take actions such as selecting units, and issuing commands to the units. The player wins if the opponent's base is destroyed, and the game ends if the player's base is destroyed. The winning, and losing outcomes of the player correspond to +1 and -1 rewards, respectively.

- **Connect 4** uses a 6×7 grid as Figure B.2 shows. Players take turns dropping tokens into the grid. The tokens land straight down, taking up the lowest available space in the column. The player wins if any four of the tokens are in a row horizontally, vertically, or diagonally. The game could also end in a draw if the grid is full while no player wins. The winning, draw, and losing outcomes of the player correspond to +1, 0, and -1 rewards, respectively.
- **Tic-Tac-Toe** uses a 3×3 grid as Figure B.11 shows. Two players iteratively place a symbol on an available position. The player wins if any three of the symbols are in a row horizontally, vertically, or diagonally. The game could also end in a draw if the grid is full while no player wins. The winning, draw, and losing outcomes of the player correspond to +1, 0, and -1 rewards, respectively.
- **Breakthrough** uses an 8×8 grid as Figure B.12 shows. Two players iteratively move one piece. If the target square is unoccupied, a piece may advance one space either straight forward or diagonally. Only if the square containing the opponent's piece is one step diagonally forward

Table B.2: Game information with the target agents to explain. "Discrete(X)" refers to X discrete actions available to choose from. "Vector(X)" refers to X-dimension continuous action space.

Games	Observation shape	Action space	Reward range	Target agent
Pong	(84,84)	Discrete(6)	0,1	Pong agent
You-Shall-Not-Pass	(380)	Vector(17)	-1,1	Runner
Kick-and-Defend	(380)	Vector(17)	-1,1	Kicker
CartPole	(4)	Discrete(2)	[0,500]	CartPole agent
Pendulum	(3)	Vector(1)	[-1200,0]	Pendulum agent
StarCraft II	(919)	Discrete (165)	-1,1	StarCraft II bot
Connect 4	(3,6,7)	Discrete(7)	-1,0,1	First player
Tic-Tac-Toe	(3,3,3)	Discrete(9)	-1,0,1	First player
Breakthrough	(8,8)	Discrete(3)	-1,1	First player
DouDizhu	(23, 54)	Discrete(Undetermined)	0,1	Landlord

may a piece move into and replace it. The winner is the first player to reach the opponent's home row. A player loses if all of his pieces are taken. A draw is impossible in the Breakthrough game. The winning and losing outcomes of the player correspond to +1 and -1 rewards, respectively.

- **DouDizhu** is a three-party card game as Figure B.13 shows. Players begin the game by putting in bids for the "landlord". The other two players act as "peasants". Being the first player to run out of cards is the game's goal. By removing all of their cards first, the landlord prevails. If one of the peasants removes all of their cards first, the peasants win. The winning and losing outcomes of the landlord correspond to +1 and -1 rewards, respectively.

We also list some basic information (*e.g.,* observation shape, action space, and reward range) about these selected games in Table B.2. Besides, we clarify the target agent to explain in Table B.2 as well (*i.e.,* except for OpenAI gym and the Atari games, all the other games are multi-agent games, and we intend to explain one specific party). For the Pong game, we train an agent based on the PPO algorithm to achieve a relatively high reward (*i.e.,* 0.906 in our setting). Regarding two MuJoCo games, we target to explain the runner in You-Shall-Not-Pass and the kicker in Kick-And-Defend. The DRL models of the target agents in the two MuJoCo games are downloaded from

Table B.3: **Results of reward gap in all games.** Each experiment contains 500 runs. We repeat all experiments three times and report the mean and standard deviation of the discounted total rewards in the table below.

Games	Agent Performance	Relative Errors (%)	Games	Agent Performance	Relative Errors (%)
Pong	0.20 (0.00)	0.89 (0.03)	StarCraft II	0.11 (0.00)	0.40 (0.01)
You-Shall-Not-Pass	-0.13 (0.01)	2.58 (0.12)	Connect 4	0.91 (0.01)	2.09 (0.12)
Kick-And-Defend	0.13 (0.01)	7.78 (0.20)	Tic-Tac-Toe	0.76 (0.02)	6.43 (0.08)
CartPole	99.34 (0.00)	0.00 (0.00)	DouDizhu	0.40 (0.02)	14.98 (0.21)
Pendulum	-144.16 (10.26)	5.56 (0.33)	Breakthrough	0.98 (0.00)	11.02 (0.16)

https://github.com/openai/multiagent-competition/tree/master/agent-zoo. As for OpenAI's CartPole and Pendulum games, we adopt well-trained agents from the stable-baselines3 model zoo: https://github.com/DLR-RM/rl-baselines3-zoo. With respect to Connect 4, Breakthrough, and Tic-Tac-Toe, we explain the first player and utilize example codes in OpenSpiel for training RL agents based on the AlphaZero algorithm [142]. The maximum simulation times for the Monte Carlo Tree Search in Connect 4, Breakthrough, and Tic-Tac-Toe are 300, 300, and 20, respectively. Concerning DouDizhu, we target at explaining the landlord and adopt the models trained by DouZero [106], which can be downloaded from https://github.com/kwai/DouZero. When it comes to StarCraft II, we use the built-in bot with difficulty A from PySC2: https://github.com/deepmind/pysc2.

Computational Resources. The experiments use a server with 2 AMD EPYC 7702 64-Core CPU Processors and 4 NVIDIA RTX A6000 GPUs to train and evaluate StateMask.

B.5.3 Additional Evaluation Results

First, the training curves of all games demonstrate that the design achieves the monotonicity property. Second, the appendix reports the target agent's performance, StateMask's performance, and the relative value errors across all games. Last, it compares the fidelity scores of StateMask with three baseline methods on five additional games: Kick-And-Defend in MuJoCo, CartPole and

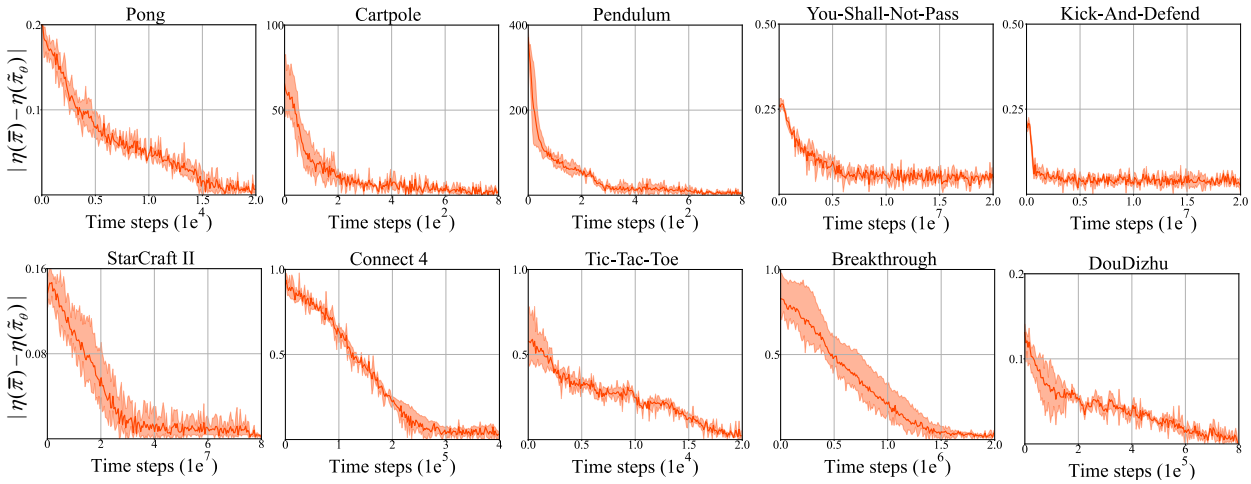


Figure B.3: The absolute policy value difference between the masked agent trained by StateMask and the target agent. Note that the darker solid lines represent the average absolute policy value difference across 3 random seeds. The lighter shadow indicates the corresponding variations between the maximal and minimal absolute policy value difference.

Pendulum in OpenAI Gym, and Breakthrough and Tic-Tac-Toe in OpenSpiel.

Monotonicity Property. Figure B.3 shows the trend of $|\eta(\bar{\pi}) - \eta(\tilde{\pi}_\theta)|$ (*i.e.,* the absolute policy value difference) during the training processes of all games. As we can observe, the absolute policy value difference monotonically decreases until it converges when the training step increases. Combined with the StateMask method section, it demonstrates that our design could guarantee the monotonicity property both theoretically and empirically.

Agent Performance. To evaluate the agent's performance before and after applying the explanation network, we run 500 tests and record the mean and standard deviation of the discounted total reward in Table B.3. The agent before and after blinding achieves similar performance when running 500 episode tests. This confirms that StateMask achieves the objective in §4.2: perturbing actions in the identified non-important time steps does not substantially change the expected total reward of the target agent.

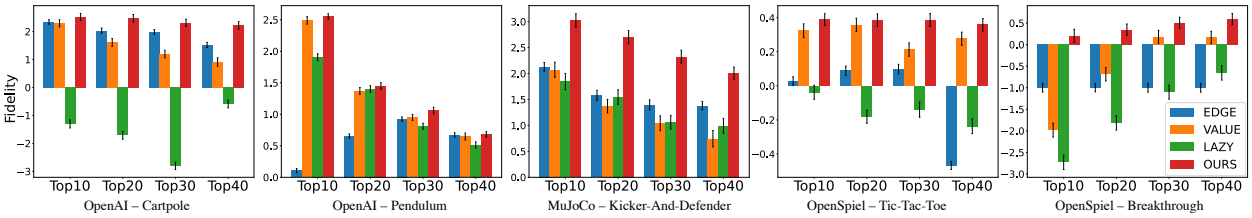


Figure B.4: Fidelity scores for interpretations generated by three baseline methods and StateMask. A higher score implies higher fidelity.

Fidelity. Figure 4.3 compares StateMask with three baseline methods on Pong, You-Shall-Not-Pass, Connect 4, DouDizhu, and StarCraft II. Here, we additionally evaluate StateMask and the three baselines on the remaining five games. Figure B.4 shows the fidelity score comparison across all four methods. Combined with Figure 4.3, these results show that StateMask consistently achieves the highest fidelity score among all games.

Sensitivity. To show the sensitivity of StateMask to the sparsity weight, we vary the hyperparameter w from {0, 1e-5, 1e-4, 1e-3, 1e-2} and train the state mask accordingly. Figure B.5(a) and Figure B.5(b) show the fidelity scores and mask ratios of StateMask in Pong and You-Shall-Not-Pass under different w settings. The fidelity scores remain stable across these settings, indicating that StateMask is not highly sensitive to this hyperparameter. Moreover, adding a sparsity constraint can increase the mask ratio, which helps filter out non-important time steps.

Efficiency. To evaluate efficiency, we record the average explanation time per trajectory in two normal-form games and two extensive-form games. Figure B.5(c) shows the explanation time of each method in four games. EDGE is a global explanation method whose explanation time is negligible, while the other three are local explanation methods. Among local explanation methods, StateMask has similar time efficiency to Value-max, whereas LazyMDP is much slower. The reason is that the target agents in these four games cannot provide the Q-value function directly, and LazyMDP therefore requires extra computation to approximate Q values for deriving explanations.

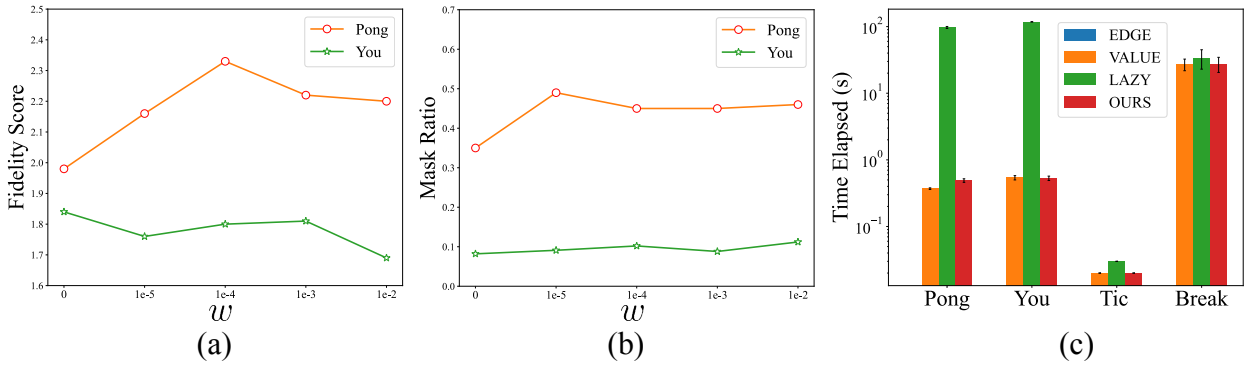


Figure B.5: Sensitivity and efficiency results. In the left and middle figures, we show the sensitivity results by varying the hyperparameter w in Pong and You-Shall-Not-Pass games. In the right figure, we report the mean and standard deviation of the explanation time per trajectory in four games: Pong, You-Shall-Not-Pass, Tic-Tac-Toe, and Breakthrough.

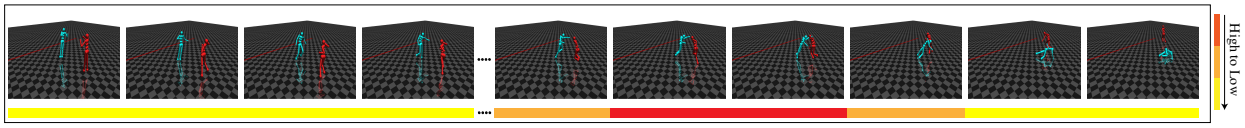


Figure B.6: Visualization of the identified critical time steps in You-Shall-Not-Pass game. The DRL agent controls the red humanoid robot and the opponent controls the blue one. While the blue humanoid bot is trying to block the red one from reaching the goal, the red one is trying to get across the red line.

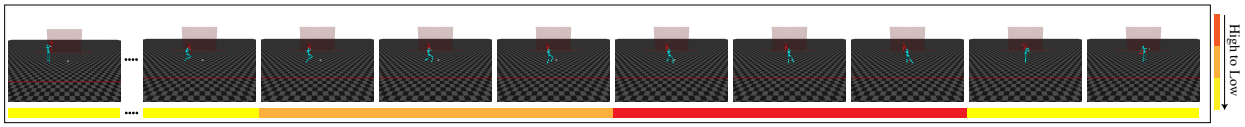


Figure B.7: Visualization of the identified critical time steps in Kick-And-Defend game. The DRL agent controls the blue humanoid robot to kick the ball and the opponent controls the red one to defend the goal.

B.6 Additional Visualization Results

You-Shall-Not-Pass. In Figure B.6, we visualize one episode of the You-Shall-Not-Pass game. Our explainer pinpoints the time steps when the red robot escapes the blue robot as the most important ones. The reason is that the DRL agent should be careful not to get knocked down by

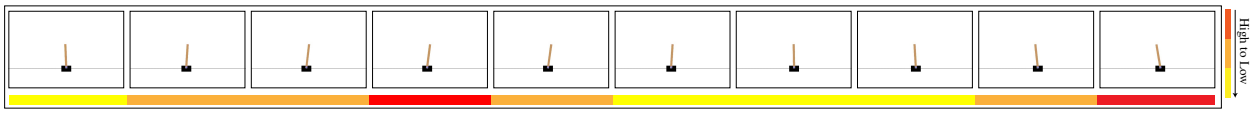


Figure B.8: Visualization of the identified critical time steps in Cartpole game. The DRL agent controls the cart to move left or right to keep the pole standing (within a certain angle) as long as possible.

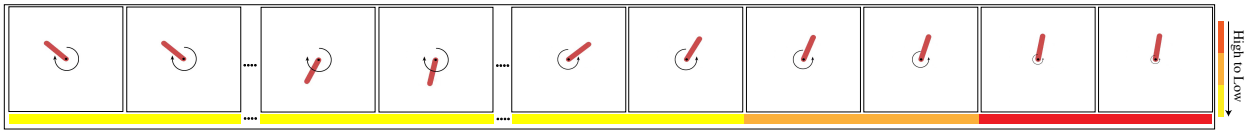


Figure B.9: Visualization of the identified critical time steps in Pendulum game. The DRL agent controls the torque applied on the free end of the pendulum to swing it into an upright position, with its center of gravity right above the fixed point.

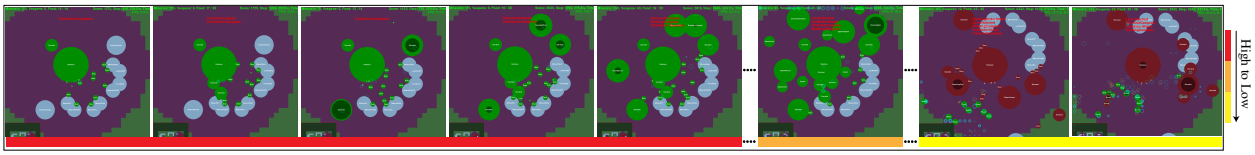


Figure B.10: Visualization of the identified critical steps in StarCraft II. The green zerg player is the target agent and the red one is the opponent. The big circles represent the buildings while the small circles represent the units. In the first three steps, the agent collects the minerals and vespene gas. In the next three steps in this episode, the agent builds some structures. In the last two steps, the small circles representing the zerglings and roaches of the target agent are attacking the enemy base.

the opponent at this moment. Otherwise, the agent will lose the balance and fall to the ground, and eventually lose the game. When the two robots are far away from each other or the blue robot is already down on the ground, the states are less important.

Kick-And-Defend. We provide one case study for the Kick-And-Defend game in Figure B.7. Our explanation method explains that the most important time steps in this game are when the blue robot is shooting. The agent should choose the proper direction and power to kick the ball to the goal at these critical time steps. Otherwise, it will be difficult for the agent to score. When the ball is flying toward the goal, the actions of the DRL agent have no impact on the final result.

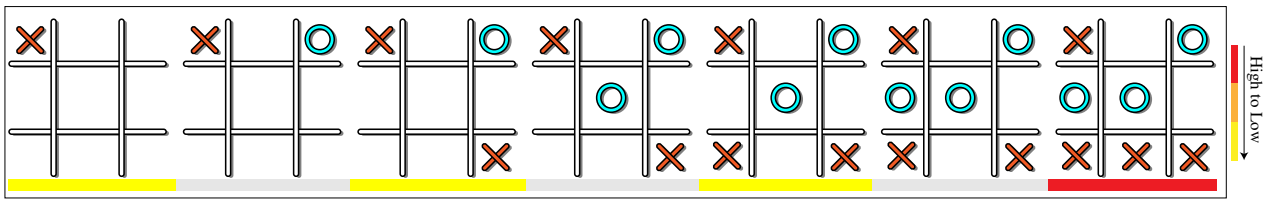


Figure B.11: Visualization of the identified critical time steps in the Tic-Tac-Toe game. The DRL agent places the X mark and the opponent places the O alternately in one of the nine spaces in the grid. The states where the O mark is placed are colored in gray.

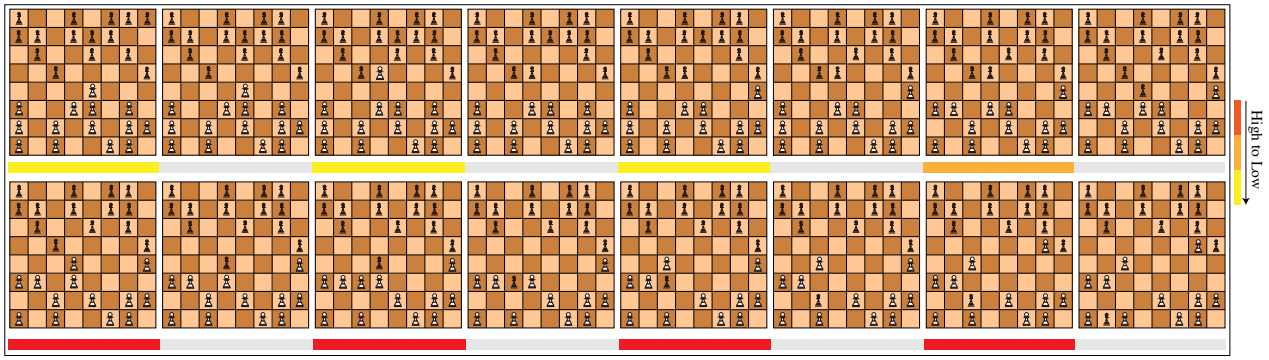


Figure B.12: Visualization of the identified critical time steps in Breakthrough game. The DRL agent plays as the black player and the opponent plays as the white one. They play alternatively, with each player moving one piece per turn.

Therefore, these time steps are less important.

CartPole. For the CartPole game, we show an example in Figure B.8. When the angle of the pole is large (no matter if it is clockwise or counterclockwise), the cart should accelerate toward the center to correct the angle, otherwise the pole will fall. StateMask treats these time steps as critical ones. When the angle of the pole is small, the cart needs to stabilize the angle, and StateMask treats these time steps as less important ones. This is reasonable since the pole is still within a safe angle range at these time steps.

Pendulum. In the Pendulum game, we show one case study in Figure B.9. The pendulum starts in

a random position and the agent tries to swing it into an upright position. StateMask pinpoints the time steps when the pendulum is close to the upright position as the most important ones. The agent should carefully apply the torque at these time steps to make the pendulum upright with zero velocity. When the angle of the pendulum is large, StateMask treats these time steps as non-critical ones.

StarCraft II. In the StarCraft II game, the DRL agent controls a Zerg player to defeat the opponent Zerg player by collecting resources and building a Zerg army. Figure B.10 shows one roach rush game where the DRL agent wins. StateMask identifies the first few steps as critical ones, including producing the drones and overlords, as well as building the spawning pool, extractors, and roach warren. These steps are reasonable and will also be deemed as important by human analysis since the successful execution of the build orders in the early stage is one key to winning the game. Also, StateMask treats the last few steps as non-important steps. This is because the opponent's army has already been crushed by the roach rush, and without the army's protection, it is just a matter of time before the agent destroys the opponent's base to win the game. Because of the automatic attack mechanism in Starcraft II, even if the DRL agent does not take any action, the roaches will destroy all the opponent's buildings eventually.

Tic-Tac-Toe. In the Tic-Tac-Toe game, Figure B.11 shows one episode explained by StateMask. The explainer only selects the last time step as the critical one since this move seals a win for the DRL agent. If the agent chooses the wrong move at this time step, the opponent will win the game.

Breakthrough. Finally, we visualize one episode of the Breakthrough game in Figure B.12. Our explainer pinpoints the last four time steps in this episode as the most important. By analyzing the strategy of the agent, we could find that the agent sacrifices one piece to make sure another piece can reach the opponent's side in the last four time steps. If the agent does not take such a strategy, the game will last for a longer time and the agent may lose the game eventually.

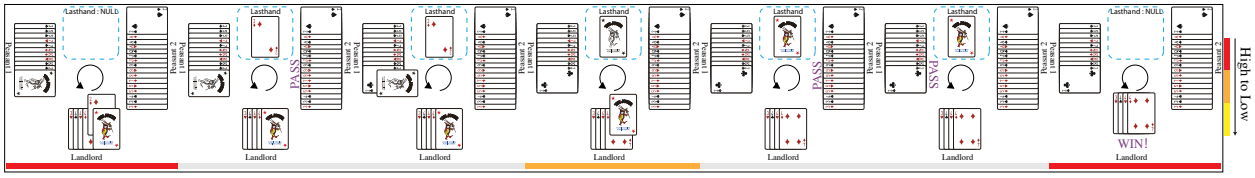


Figure B.13: Visualization of the identified critical time steps in the DouDizhu game. The DRL agent plays the role of the landlord and two cooperative opponents play the role of the peasants. The landlord's goal is to empty his hand first, while the peasants aim to prevent this and empty one of their hands first. The states where the peasants take actions are marked with a gray bar.

DouDizhu. In DouDizhu, a landlord (our target agent) competes with two cooperative peasants to be the first to run out of cards. Figure B.14 shows one critical step identified by StateMask. At this step, the landlord chooses to play eight cards, *i.e.,* Quad with Pairs. This move enables the landlord to play most low-rank cards and win the right to play first in the next round (no peasant has larger combinations than this one). If these cards are not played together, it is hard for the landlord to play them separately because the Quad and two Pairs are relatively small and cannot help the landlord to get the right to continue playing first.

We also give the winning trajectory for the DouDizhu game in Figure B.13. Our explainer pinpoints two critical time steps in this episode. The first one is when the landlord plays **2**. Since **Red Joker** in hand is the largest solo that can beat any other solo, the landlord can ensure the right to play cards in the next round. The second one is when the landlord plays **Trio with Solo** to empty his hand. If the landlord plays **Trio** or **Pair** separately, the peasant will have a chance to win the game. Therefore, the two selected time steps are critical to the final result.

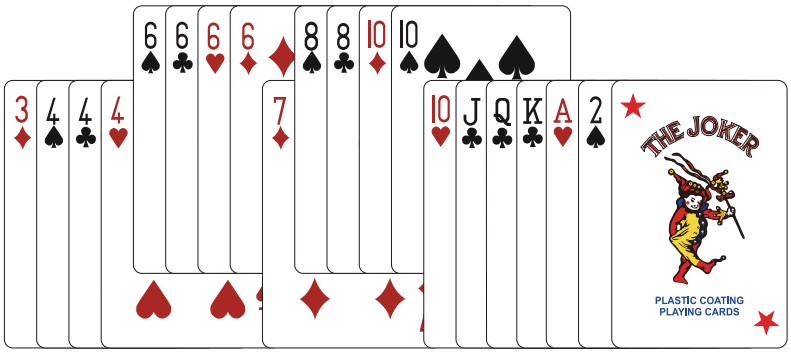


Figure B.14: One critical step in the DouDizhu game: The target agent (landlord) plays the Quad with Pairs to ensure that peasants cannot beat this hand.

B.7 Additional Results of Attacks

This section provides the results of launching adversarial attacks under the guidance of explanation when varying the attack ratio between 10% and 30% in Table B.4. First, when the threshold increases, the victim agent's performance drops. Second, with the same threshold setting, the adversarial attack guided by StateMask is the most powerful one. These results further support the conclusion that StateMask has the highest fidelity.

B.8 Additional Results of Policy Patching

Advantage of the Retraining-Based Patching Method. In this experiment, we run the patching method from EDGE [42] on two MuJoCo games. At a high level, this method first replays the losing trajectory until the start of the critical steps and performs random exploration at the critical steps in order to find a winning strategy. The actions together with the corresponding critical steps in the winning strategy are recorded. Then, it forms a look-up table with these collected state-action pairs and uses the look-up table to remediate the original policy. We follow the EDGE setting and

run 500 tests to obtain the target agents' winning-rate change before and after patching. Table B.5 shows the results of EDGE's patching method. Although the agents' performance improves after applying EDGE's patching method, the improvements are overall marginal. Together with the results in Table 4.1 and Table B.6, these results show the advantage of the retraining-based patching method used in this chapter. The reason may be that EDGE uses a hard rule (*i.e.,* look-up table) to patch the agents' policies, and the quality of the look-up table has a strong influence on the patched agents' final performance. In contrast, retraining does not require a fixed look-up table to store the correct actions corresponding to critical states; instead, it encourages the DRL agent to explore at critical time steps and update the neural network parameters accordingly.

Patching Results on Other Games. Table B.6 provides the agent performance improvement results after retraining under the guidance of StateMask and the three baselines in Pendulum, Kick-And-Defend, and Tic-Tac-Toe. Under explanation-guided retraining, the performance of the reinforcement learning agent generally increases. More importantly, retraining based on StateMask explanations achieves the best performance.

B.9 Evaluation on Explaining Sub-optimal Agents

To verify the effectiveness of StateMask in explaining sub-optimal agents, we conduct additional experiments on sub-optimal agents in Pong, Kick-And-Defend, Connect 4, and Breakthrough. These games include both normal-form games (*i.e.,* Pong and Kick-And-Defend) and extensive-form games (*i.e.,* Connect 4 and Breakthrough). For Pong, Connect 4, and Breakthrough, we use policies from the halfway stage of training as sub-optimal agents. In Kick-And-Defend, we follow the setting in Appendix B.5.2 and explain the defender. The agent zoo (https://github.com/openai/multiagent-competition/tree/master/agent-zoo)

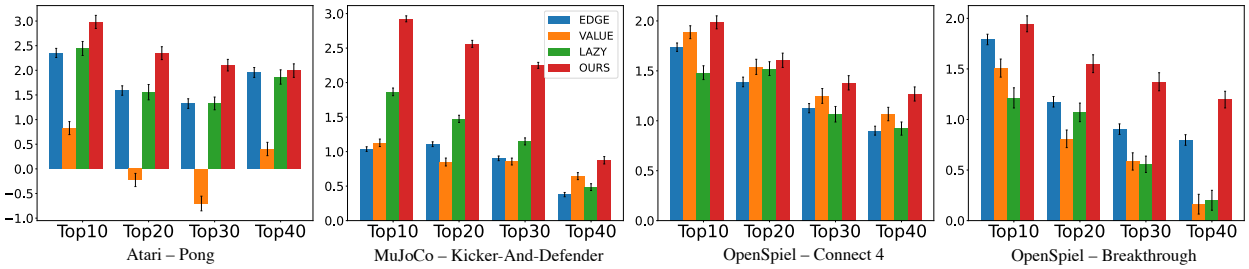


Figure B.15: Fidelity scores for interpretations generated by three baseline methods and StateMask for four sub-optimal agents.

provides three different policies for the kicker. We hold the defender fixed, let each kicker play against the defender, record the corresponding winning rates, and choose the kicker with the lowest winning rate as the sub-optimal agent.

B.9.1 Fidelity

In this experiment, we compare the fidelity scores of StateMask and baseline methods across the four selected games. As depicted in Figure B.15, StateMask still achieves the highest fidelity score among the four games even though the target agent is sub-optimal. The reason is that StateMask is designed to preserve the original agent's performance before and after masking the target agent at certain time steps. In instances where the target agent is sub-optimal, the important time steps are those that lead the target agent to win or lose (*i.e.,* with a higher chance) in a single episode. Altering the actions in these time steps may cause a significant variation in performance. Therefore, StateMask has the potential to recognize and distinguish these critical time steps by learning how to unmask them accurately.

B.9.2 Visualizing the Explanation for Sub-optimal Agent Behaviors

We also visualize the important time steps identified by StateMask in the selected four games to show that StateMask can generate human-understandable explanations even for sub-optimal agents.

Pong. Regarding the Pong game as Figure B.16 shows, StateMask indicates that the most crucial time step is when the ball is flying towards the yellow paddle. Nevertheless, the yellow paddle moves upward too much when trying to catch the ball. If the yellow paddle takes a different action, *e.g.,* moves downward, the game outcome could be flipped.

Kick-And-Defend. In the Kick-And-Defend game, as Figure B.17 shows, the most critical time step is identified when the kicker is in contact with the ball. The kicker's positioning and approach to the ball lead to a sub-optimal angle for the shot. Consequently, this results in the ball being intercepted easily by the defender, who is well-positioned to block the trajectory. If the kicker were to adjust the angle or approach, the outcome of the game could potentially change, as the ball might bypass the defender, increasing the likelihood of scoring a goal.

Connect4. In the Connect 4 game as Figure B.18 shows, StateMask pinpoints the penultimate step as the most crucial step of the losing trajectory. Once the black player determines to drop his token into the seventh column in the second-to-the-last step, we could observe that no matter which column the black player chooses in the last step, the game outcome will always be a failure. However, if the black player chooses the second column in the second-to-the-last step, he would have a chance to win, *i.e.,* , connecting four tokens between the first column and the fourth column either horizontally or diagonally. Therefore, the second-to-the-last step is the most crucial step in this losing trajectory.

Breakthrough. Regarding the Breakthrough game as Figure B.19 shows, StateMask identifies

Figure B.16: Visualization of the identified critical time steps in the Pong game. The yellow paddle is controlled by the agent we aim to explain.

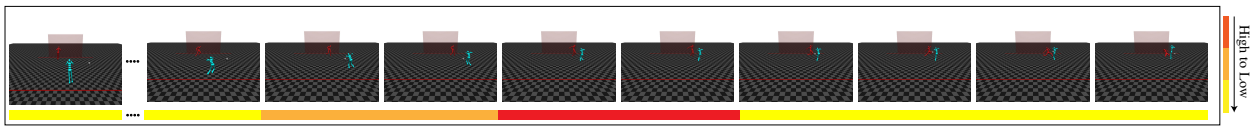


Figure B.17: Visualization of the identified critical time steps in the Kick-And-Defend game. The kicker (blue agent) is the target agent for explanation.

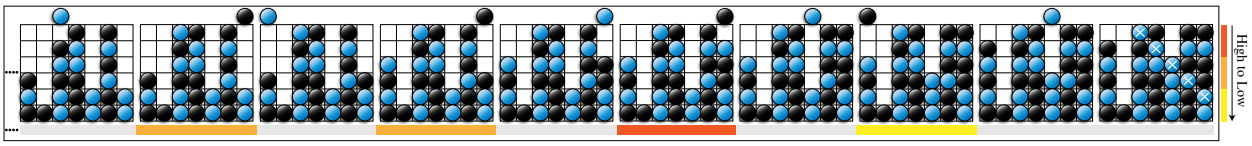


Figure B.18: Visualization of the identified critical time steps in Connect 4 game. The black player is the sub-optimal agent we aim to explain.

the fifth-to-the-last and the fourth-to-the-last step as the most critical ones. Note that only if the square containing the opponent's piece is one step diagonally forward may a piece move into and replace it. In the fifth-to-the-last step, the black player fails to replace the left white piece in the fourth row, directly causing the game to lose. Additionally, in the fourth-to-the-last step, the black player's action makes him lose the most advanced piece. The identified two steps are thus the key steps for the black player to lose the game.

B.9.3 Patching Results of Sub-optimal Agents

In this experiment, we use the retraining method from §4.3.4 to patch the sub-optimal agents in the selected four games. Table B.7 shows the sub-optimal agent performance improvement under the guidance of StateMask and the baseline methods. After retraining, the agents' performance generally improves because these agents are far from optimal. More importantly, retraining based

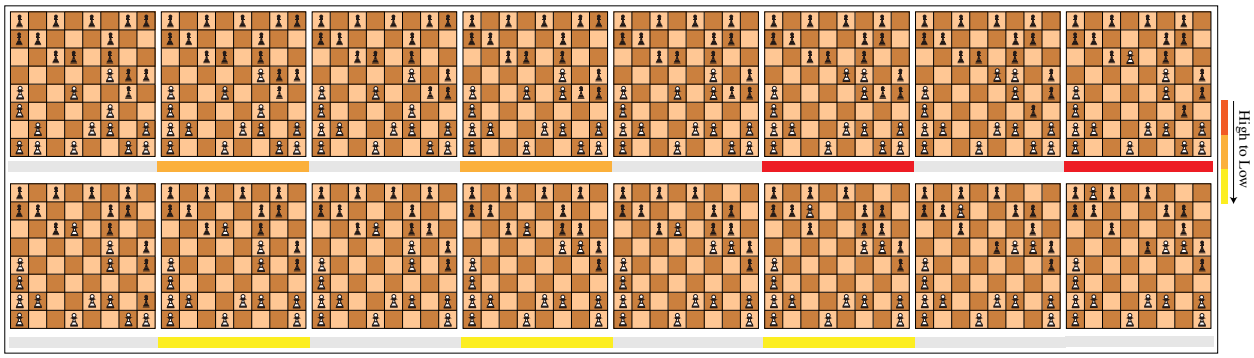


Figure B.19: Visualization of the identified critical time steps in the Breakthrough game. The sub-optimal agent plays as the black player and the opponent plays as the white one.

on StateMask explanations achieves the best performance.

B.10 Evaluation on an Alternative Design

This section examines an alternative design of StateMask named `PPOMask`, which trains a mask network by maximizing the masked agent's expected total reward (*i.e.,* $J(\theta) = \max_\theta \eta(\tilde{\pi}_\theta)$). This is done by directly using the PPO algorithm to train the mask network. To avoid the trivial solution, the design adds the same additional loss item L_t^{MASK} to the PPO loss. We evaluate `PPOMask` in Pong, Kick-And-Defend, Connect 4, and Breakthrough.

Relative Error. Here, we report the relative errors of `PPOMask` in comparison to StateMask across these chosen four games. As depicted in Table B.8, `PPOMask` generates a higher relative error than StateMask when the target agent is sub-optimal, due to its inability to maintain the target agent's performance (*i.e.,* `PPOMask` trains a masked agent that performs better than the original agent).

Fidelity. We also test the fidelity of `PPOMask` and compare it with StateMask in these four games. According to Figure B.20, `PPOMask` exhibits poor fidelity on sub-optimal agents. This can be attributed to the objective function of `PPOMask`, which is to maximize the expected total reward

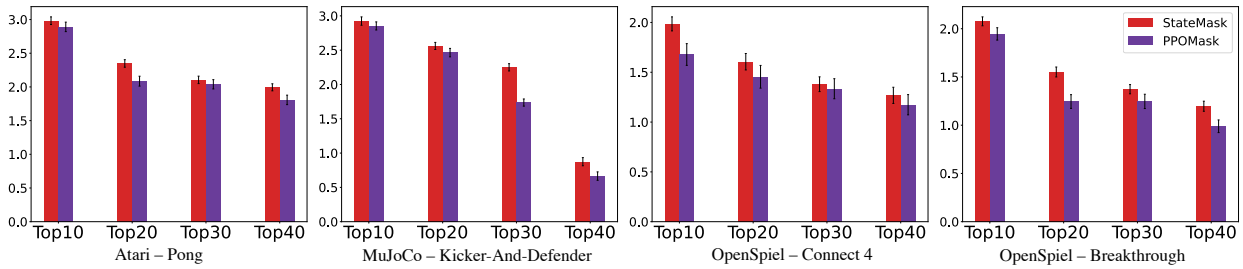


Figure B.20: Fidelity comparisons between StateMask and PPOMask across four selected games when the agent is sub-optimal.

of a mixed policy. Consequently, PPOMask might disregard the critical states that contribute to the failure of the original agent and instead take random actions (*i.e.,* mask operation) during those steps in order to secure a win under the perturbed policy. However, as shown in B.9, StateMask identifies the critical states that contribute to the failure of the sub-optimal agent and thus has higher fidelity.

B.11 Potential Social Impact

Our work focuses on improving the trustworthiness of Deep Reinforcement Learning systems, which are increasingly being used in a wide range of applications, from gaming [143] to robotics [144] to healthcare [145]. One of the main challenges of DRL is the black-box nature of the models, which makes it difficult to understand how they arrive at their decisions. This lack of transparency can create concerns about the reliability and safety of DRL systems.

To address this challenge, our work aims to automatically generate explanations for DRL systems, without requiring human supervision. By generating explanations automatically, we reduce the cost and effort required to provide explanations, making them more accessible and scalable for various applications. Moreover, automatic explanation avoids potential biases that may arise from human-generated explanations, as these may be influenced by subjective factors or an incomplete

understanding of the system.

Our approach involves developing models that can analyze the behavior of DRL systems and generate explanations based on that analysis. These explanations provide insights into how the DRL system works, what factors influence its decisions, and how it can be improved. By providing this information, our work contributes to enhancing the transparency and interpretability of DRL systems, which can help users to evaluate the reliability of these systems.

When it comes to potential negative societal impacts, the use of STATEMASK for enhancing RL agents may lead to the magnification of biased and unfair RL agents. Nevertheless, STATEMASK also helps users understand the decision-making process of biased and unfair RL agents, which enables the implementation of strategies to diminish their harmful effects. Therefore, STATEMASK is useful for understanding and improving RL agents.

Table B.4: Performance drop under varying attack ratios in all games. Each experiment contains 500 runs. We repeat all experiments three times and report the average rewards with the corresponding standard deviations in the table below.

Games	Attack ratio	EDGE	Value-max	LazyMDP	Ours
Pong	10%	-63.04 (3.44)	-0.40 (0.40)	-36.17 (1.15)	**-84.90 (1.40)**
	20%	-65.73 (0.81)	-7.06 (8.81)	-57.00 (0.92)	**-85.53 (0.75)**
	30%	-71.07 (1.36)	-70.13 (0.64)	-62.63 (1.63)	**-86.90 (0.76)**
You-Shall-Not-Pass	10%	-2.64 (0.60)	-14.80 (3.20)	-28.87 (1.21)	**-35.93 (1.81)**
	20%	-6.87 (0.61)	-55.13 (4.27)	-50.63 (0.80)	**-61.67 (0.12)**
	30%	-11.68 (1.70)	-61.93 (0.12)	-61.87 (0.23)	**-61.93 (0.12)**
Kick-And-Defend	10%	-10.46 (0.97)	-24.57 (1.86)	-28.33 (1.76)	**-28.37 (1.10)**
	20%	-28.51 (0.72)	-64.23 (1.63)	-60.43 (1.33)	**-68.63 (1.17)**
	30%	-37.25 (1.03)	-74.30 (0.40)	-73.57 (1.53)	**-74.10 (0.80)**
CartPole	10%	-78.98 (4.61)	-4.85 (6.64)	-0.50 (0.51)	**-127.81 (13.82)**
	20%	-153.21 (11.87)	-20.96 (6.21)	-13.92 (6.42)	**-198.06 (24.21)**
	30%	-237.03 (5.30)	-72.99 (7.98)	-56.71 (2.79)	**-332.05 (16.33)**
Pendulum	10%	-58.84 (2.79)	-255.80 (15.92)	-135.23 (1.51)	**-570.77 (1.13)**
	20%	-129.37 (1.86)	-432.15 (43.03)	-241.20 (0.95)	**-707.69 (0.74)**
	30%	-201.39 (3.31)	-563.63 (15.16)	-372.88 (4.21)	**-796.07 (12.07)**
StarCraft II	10%	-70.00 (14.18)	-77.33 (5.69)	-72.00 (13.53)	**-83.00 (7.21)**
	20%	-93.33 (4.73)	-93.00 (3.61)	-93.00 (4.58)	**-94.00 (3.00)**
	30%	-97.00 (0.00)	-97.00 (0.00)	-97.00 (0.00)	**-97.00 (0.00)**
Connect 4	10%	-33.33 (1.56)	-26.30 (1.44)	-22.57 (1.40)	**-35.37 (3.00)**
	20%	-56.73 (2.30)	-67.57 (1.57)	-53.83 (0.81)	**-71.90 (1.44)**
	30%	-76.23 (1.94)	-82.70 (0.80)	-69.83 (1.17)	**-85.50 (0.72)**
Tic-Tac-Toe	10%	-0.30 (0.23)	-8.07 (0.35)	-4.40 (0.25)	**-17.24 (0.40)**
	20%	-6.72 (0.33)	-18.20 (0.17)	-7.60 (0.12)	**-29.90 (0.45)**
	30%	-19.50 (0.36)	-25.12 (0.13)	-13.90 (0.41)	**-42.85 (0.17)**
Breakthrough	10%	-6.53 (0.50)	-18.00 (1.00)	-18.90 (0.36)	**-88.73 (1.62)**
	20%	-13.67 (0.83)	-30.93 (1.01)	-26.87 (1.03)	**-95.87 (1.21)**
	30%	-26.33 (1.53)	-42.67 (2.89)	-36.00 (2.65)	**-98.33 (1.15)**
DouDizhu	10%	-0.25 (1.71)	-18.60 (0.31)	-25.20 (0.76)	**-29.94 (0.64)**
	20%	-10.45 (1.68)	-24.40 (0.83)	-32.27 (0.69)	**-32.87 (0.20)**
	30%	-11.29 (1.40)	-28.07 (1.22)	-32.67 (0.20)	**-34.54 (0.31)**

Table B.5: Results of EDGE's policy patching method with different explanation methods. Each experiment contains 500 runs. We repeat all experiments three times and report the average reward increase with the corresponding standard deviation in the table below.

Application	Games	EDGE	Value-max	LazyMDP	Ours
Target agent's winning rate improvement after patching	Kick-And-Defend	+0.00 (0.00)	+0.00 (0.00)	+0.00 (0.00)	+0.00 (0.00)
	You-Shall-Not-Pass	+0.00 (0.00)	+0.75 (0.04)	+0.25 (0.01)	+0.25 (0.02)

Table B.6: Additional results of policy patching. Each experiment contains 500 runs. We repeat all experiments three times and report the average reward increase with the corresponding standard deviation in the table below.

Application	Games	EDGE	Value-max	LazyMDP	Ours
Target agent's winning rate improvement after patching	Kick-And-Defend	+1.03 (0.12)	+0.70 (0.20)	+0.97 (0.06)	**+1.37 (0.12)**
	Pendulum	+0.00 (0.00)	+0.29 (0.03)	+2.65 (0.05)	**+4.30 (0.40)**
	Tic-Tac-Toe	+0.03 (0.06)	+0.23 (0.06)	+0.20 (0.10)	**+ 2.47 (0.12)**

Table B.7: Patching results of sub-optimal agents in four games. Each experiment contains 500 runs. We repeat all experiments three times and report the average reward increase with the corresponding standard deviation in the table below.

Application	Games	EDGE	Value-max	LazyMDP	Ours
Target agent's winning rate improvement after patching	Pong	+5.80 (0.12)	+5.40 (0.03)	+12.00 (1.24)	**+13.60 (1.22)**
	Kick-And-Defend	+7.42(0.40)	+8.43(0.24)	+4.78(0.12)	**+10.26(0.62)**
	Connect 4	+24.80 (1.68)	+35.80 (1.84)	+36.20 (2.20)	**+37.60(1.66)**
	Breakthrough	+68.40 (3.68)	+65.80 (2.77)	+ 68.80 (3.26)	**+69.00 (3.07)**

Table B.8: Results of relative errors in four selected games. Each experiment contains 500 runs. We repeat all experiments three times and report the mean and standard deviation of the discounted total rewards in the table below.

Games		Pong	Kick-And-Defend	Connect 4	Breakthrough
Sub-optimal agent performance		0.18 (0.00)	0.09 (0.00)	0.52 (0.01)	-0.39 (0.01)
Relative error	STATEMASK	2.57 (0.03)	5.34 (0.54)	3.05 (0.34)	12.89 (1.86)
	PPOMask	3.86 (0.06)	7.53 (0.65)	5.77 (0.42)	28.57 (2.44)

Appendix C

Supplementary Material for GPO

C.1 Additional Preference Optimization Objectives

Beyond the DPO objective described in the main text, several other preference optimization methods have been developed. These methods also typically operate on an offline dataset of preference pairs $\mathcal{D} = \{(x, y^+, y^-)\}$, where y^+ is the preferred response and y^- is the dispreferred response given an input x. The policy being optimized is denoted by π_θ, with θ being its parameters, and π_{ref} is a fixed reference policy.

ORPO [60] introduces an objective that penalizes the model for assigning low likelihood to preferred responses, while simultaneously ensuring that preferred responses have higher odds than dispreferred ones. For the single preference pair (x, y^+, y^-), the ORPO loss is:

$$\mathcal{L}_{\text{ORPO}}(\pi_\theta; x, y^+, y^-) = -\log p_\theta(y^+|x) - \lambda \log \sigma \left(\log \frac{p_\theta(y^+|x)}{1 - p_\theta(y^+|x)} - \log \frac{p_\theta(y^-|x)}{1 - p_\theta(y^-|x)} \right),$$

where $p_\theta(y|x) = \exp\left(\frac{1}{|y|} \log \pi_\theta(y|x)\right)$ is a length-normalized likelihood for sequence y, $\log \pi_\theta(y|x)$ is the sum of log-probabilities of tokens in y, $\sigma(\cdot)$ is the sigmoid function, and λ is a weighting coefficient. This formulation directly encourages the policy to generate y^+ and ensures its odds are

favorable compared to y^-.

SimPO [61] offers a modification of the DPO-style loss by incorporating sequence length normalization directly into the log-probability difference and adding a margin term γ. The SimPO loss for a preference pair is:

$$\mathcal{L}_{\text{SimPO}}(\pi_\theta; x, y^+, y^-) = -\log \sigma \left(\beta \frac{\log \pi_\theta(y^+|x)}{|y^+|} - \beta \frac{\log \pi_\theta(y^-|x)}{|y^-|} - \gamma \right),$$

where β is a constant that controls the scaling of the reward difference. This objective aims to maximize the margin between the length-normalized log-likelihood of the preferred response and that of the dispreferred response.

KTO [59] introduces an alignment approach rooted in prospect theory. Instead of optimizing preference likelihoods, KTO focuses on directly maximizing the utility of each individual generation y given an input x. A distinctive characteristic is its reliance on a binary signal for every input-output pair $(x, y) \in \mathcal{D}$, classifying y as either desirable or undesirable for x. Consequently, KTO does not inherently require paired preference data (*i.e.,* y^+ vs y^-). The loss for a single such sample (x, y) is formulated to be minimized and is given by:

$$\mathcal{L}_{\text{KTO}}(\pi_\theta, \pi_{\text{ref}}; x, y) = \begin{cases} \lambda_D \left(1 - \sigma \left(\beta \left(r_\theta(x, y) - z_0\right)\right)\right) & \text{if } y \text{ is desirable for } x \\ \lambda_U \left(1 - \sigma \left(\beta \left(z_0 - r_\theta(x, y)\right)\right)\right) & \text{if } y \text{ is undesirable for } x \end{cases}$$

In this formulation, $r_\theta(x, y) = \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ represents the log-probability ratio of the current policy π_θ against a fixed reference policy π_{ref}. The term z_0 serves as a reference point related to the KL estimate. β is a hyperparameter modulating risk aversion, and λ_D, λ_U are positive hyperparameters that weight the contributions from desirable and undesirable outputs, respectively. This

structure allows KTO to process feedback on individual generations, and if paired data is available, each part of the pair (y^+, y^-) would contribute its own loss term based on its desirable/undesirable status.

C.2 Additional Experiment Details

In this section, we provide additional details on the experiments.

C.2.1 Dataset Details

As mentioned in §5.5, we utilize established train/test splits for several benchmarks. For GSM8K, MATH, MMLU, and MMLUPro, we adopt their standard train/test distributions. Specifically for BBH, the dataset is randomly partitioned into training and test sets at the sub-task level. For the AIME dataset, problems from the years 1983-2023 constitute the training set, while problems from 2024-2025 form the test set. Detailed statistics for each dataset, including the number of samples in the training and test sets, and the source of the dataset, are presented in Table C.1.

Table C.1: **Dataset Statistics**: The table presents the number of training and test samples for each dataset, along with the source of the dataset.

Dataset	# Train Samples	# Test Samples	Source
GSM8K	7470	1320	https://huggingface.co/datasets/openai/gsm8k
MATH	12000	500	https://github.com/openai/prm800k
MMLU	99842	14042	https://huggingface.co/datasets/cais/mmlu
MMLUPro	9625	2407	https://huggingface.co/datasets/TIGER-Lab/MMLU-Pro
BBH	3261	3250	https://github.com/suzgunmirac/BIG-Bench-Hard
AIME-2024	903	30	https://huggingface.co/datasets/gneubig/aime-1983-2024
AIME-2025	903	30	https://huggingface.co/datasets/yentinglin/aime_2025

C.2.2 Implementation Details

Here we provide further details on the data processing and algorithm implementation for the experiments.

Question Filtering To construct a training set of appropriate difficulty, we apply a filtering process to each dataset. For every question in the initial training pool, we generate 8 responses using the base model with a sampling temperature of 0.7. Questions that are solved correctly across all eight attempts, or conversely, incorrectly across all eight attempts, are subsequently excluded from the training set. This procedure aims to retain questions that are neither trivially easy nor prohibitively difficult for the base model, thereby focusing the fine-tuning process on a more informative problem distribution. This filtering protocol is applied uniformly to create the training data for both the baseline methods and their GPO-enhanced counterparts.

Step Grouping for Trajectory Segmentation Reasoning trajectories generated by the models are segmented into multiple steps using the following procedure. First, each trajectory is split into steps based on newlines. Multiple consecutive newline characters are collapsed into a single newline. To avoid overly short steps, any step consisting of fewer than 30 words is merged with the previous step. Furthermore, to maintain computational feasibility, we impose a maximum number of steps: 15 for offline preference optimization methods and 10 for online PPO. If a trajectory exceeds this maximum step count, all subsequent steps beyond the limit are concatenated to form the last step.

Preference Data Preparation for Offline Methods For the offline preference optimization algorithms DPO, SimPO, and ORPO, positive and negative preference data are sourced during the

question filtering process described above. Since the filtering retains questions for which the base model produces a mix of correct and incorrect responses across the 8 generated samples, these naturally provide pairs of successful (positive) and unsuccessful (negative) trajectories for the same input question.

For KTO, which requires demonstration data rather than explicit preference pairs, we adapt the approach from the original KTO paper [59]. For the baseline KTO, each preference pair (y^+, y^-) from the DPO dataset is decomposed into two separate demonstrations: y^+ with the positive tag and y^- with the negative tag. For the GPO-enhanced KTO, we first apply the GPO strategy to form paired preference data. These n preference pairs are then similarly decomposed into $2 * n$ individual positive and negative demonstrations to train the GPO-enhanced KTO model.

C.2.3 Hyper-parameters for Experiments

We list the main hyper-parameters for the experiments in Table C.2. The GPO-enhanced methods are trained with the same hyper-parameters as the baseline methods. For those unmentioned hyper-parameters, we use the default values provided by the LLaMA-Factory framework [146].

C.2.4 Running Environment

Our training process primarily utilizes the LLaMA-Factory framework [146]. The experiments are conducted on a server equipped with four AMD EPYC 7702 64-Core CPU Processors and eight NVIDIA H100 80GB GPUs. The total computational resources consumed include approximately 1800 GPU hours and 2 TB of storage for model checkpoints.

Table C.2: **Hyper-parameters for Different Datasets and Methods.**

Method		Dataset						
	Hyper-parameter	GSM8K	MATH	MMLU	MMLUPro	BBH	AIME-2024	AIME-2025
Common	LoRA Alpha (α)	2	2	2	2	2	2	2
	LoRA Rank	8	8	8	8	8	8	8
	LoRA Target	all	all	all	all	all	all	all
	Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
	Sequence Length	4096	4096	2048	2048	2048	4096	4096
PPO	Clip Value	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	KL Divergence Coeff.	0.05	0.05	0.05	0.05	0.05	0.05	0.05
	Learning Rate	1e-5	1e-5	2e-5	2e-5	1e-5	1e-5	1e-5
	Batch Size	32	64	64	64	128	32	32
	Epochs	5	5	2	3	5	5	5
DPO	Beta (β)	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Learning Rate	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
	Batch Size	4	16	16	16	8	4	4
	Epochs	10	10	5	10	10	10	10
KTO	Desirable Reward Scalar	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Undesirable Reward Scalar	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Learning Rate	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
	Batch Size	8	16	16	16	8	8	8
	Epochs	10	10	2	10	10	10	10
SimPO	Reward Margin	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	Learning Rate	1e-5	5e-6	5e-6	1e-5	1e-5	1e-5	1e-5
	Batch Size	4	16	16	16	8	4	4
	Epochs	10	10	5	10	10	10	10
ORPO	Learning Rate	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5	1e-5
	Batch Size	2	16	16	16	8	2	2
	Epochs	10	10	5	10	10	10	10

C.2.5 Experimental Details for Long-Trajectory Reasoning

We provide supplementary details for the experiment designed to validate the scalability of GPO to tasks involving very long trajectories.

Experimental Setup. To create a challenging long-context reasoning benchmark, we selected a sub-dataset from BIG-Bench Extra Hard (BBEH). This dataset was chosen for its particularly long problem descriptions (averaging approximately 1,700 tokens) and lengthy, complex reasoning chains (averaging approximately 190 lines). The experiment was conducted using a training set

Table C.3: Accuracy on a long-context reasoning subset of BBEH.

Method	Accuracy (%)
Base Model (DeepSeek-R1-Distill-Qwen-7B)	24.5
DPO Baseline	29.0
GPO-DPO (with step grouping)	**36.0**

of 1,000 questions and a test set of 200 questions. The base model used for all experiments was DeepSeek-R1-Distill-Qwen-7B.

Step Grouping Heuristic. To adapt GPO for these long trajectories without incurring prohibitive computational costs, we implemented a simple step grouping heuristic. For the GPO-DPO method, we grouped every 15 lines of the generated reasoning chain into a single logical "step". This approach maintains the core principle of identifying pivotal moments for targeted feedback while significantly reducing the number of points at which future returns need to be estimated.

Results. The results, summarized in Table C.3, demonstrate that even with this straightforward heuristic, GPO delivers substantial performance gains over both the base model and the DPO baseline. This confirms that GPO can be effectively adapted for long-trajectory reasoning tasks.

C.3 PatchAgent Vulnerability Patching Results

To evaluate GPO in a security setting, we apply GPO-tuned models to a PatchAgent vulnerability patching benchmark. The benchmark groups vulnerabilities by bug class and measures the percentage of cases for which the agent produces a successful patch.

Table C.4: **PatchAgent vulnerability patching benchmark.** Results are reported as successful patch rates by vulnerability class and in aggregate.

Model	Temporal Error	Spatial Error	Null Dereference	Numeric Error	Total
GPT-4o	56.52%	76.80%	100%	100%	78.09%
GPT-4 Turbo	47.83%	69.60%	91.30%	100%	70.79%
Claude-3 Opus	60.87%	86.40%	95.65%	100%	84.84%
Claude-3 Sonnet	34.78%	61.60%	73.91%	85.71%	60.67%
Claude-3 Haiku	39.13%	74.40%	82.61%	100%	71.91%
Qwen-2.5-32B-GPO	56.52%	80.00%	95.65%	100%	79.77%
Qwen-3-30B-GPO	60.87%	80.80%	100%	100%	84.26%
GLM-4.5air-106B-GPO	**65.21%**	**88.00%**	100%	100%	**87.07%**

C.4 User Study

To evaluate if the human annotators agree with the critical steps identified by our method, we conducted a user study. This section details the setup of the study and the format used for presenting questions to the participants.

C.4.1 User Study Setup

Before participating in the study, participants are presented with an informed consent form, outlining the purpose of the research, the nature of their participation, and confirming that no private information would be collected. Participation is voluntary.

The study consisted of five questions and was designed to take approximately five minutes to complete. For each question, participants were presented with the following:

- **Task description:** A brief overview of the problem type.
- **Question:** The specific question posed to the LLM.
- **Ground truth answer:** The correct answer to the question.
- **LLM's reasoning trajectory:** The step-by-step reasoning process generated by an LLM,

with each step clearly numbered. This reasoning leads to an incorrect answer.

- **Selection task:** Participants are then shown four potential critical steps. Their task is to select the single step they believe is the most critical in the LLM's reasoning process leading to the result. Within the four steps, there is one step that is identified by our method, and the other three steps are randomly selected from the rest of the steps in the trajectory.

C.4.2 User Study Questions and Results Format

We list the questions and the results format below. We highlight the critical step identified by our method in red for presentation in this appendix, and note that the participants did not see the red-highlighted text.

Question 1: Infer the date from context

Task description: Infer the date from context.

Q: It was Sept. 1st, 2021 a week ago. What is the date 24 hours later in MM/DD/YYYY?

Options:

(A) 03/09/2022

(B) 09/10/2021

(C) 08/31/2021

(D) 09/09/2021 (Ground Truth)

(E) 09/02/2021

(F) 09/17/2021

The large language model's reasoning trajectory with labeled step numbers is:

1. Okay, so I need to figure out the date 24 hours after September 1st, 2021. Let me break

this down step by step.

2. First, the question says it was September 1st, 2021, a week ago. Wait, does that mean today is September 8th, 2021? Because a week is seven days, so adding seven days to September 1st would land on September 8th. But the question is asking about 24 hours after September 1st, not today's date. Hmm, maybe I'm overcomplicating it.
3. Wait, no, the question is straightforward. It says, "It was Sept. 1st, 2021 a week ago. What is the date 24 hours later in MM/DD/YYYY?" So, regardless of today's date, I just need to find the date that is exactly 24 hours after September 1st, 2021.
4. So, 24 hours later would be the next day, September 1st, plus one day is September 2nd, 2021. Let me check the options provided.
5. (A) 03/09/2022-That's March 9th, 2022. That's way too far in the future.
6. (B) 09/10/2021-September 10th, 2021. That's nine days later, not just one.
7. (C) 08/31/2021-August 31st, 2021. That's the day before September 1st, so that's a day earlier, not later.
8. (D) 09/09/2021-September 9th, 2021. Again, that's eight days later.
9. (E) 09/02/2021-September 2nd, 2021. That's exactly one day after September 1st.
10. (F) 09/17/2021-September 17th, 2021. That's 16 days later.
11. So, the correct answer should be option (E), which is 09/02/2021. I think I got it right. Just adding one day to September 1st gives September 2nd.
12. The date 24 hours after September 1st, 2021 is September 2nd, 2021, which corresponds to option (E).

Participant task: There are several steps listed; please select the step that you believe is most critical to the LLM's failure.

- Step 2
- Step 3
- Step 5
- Step 11

Results for Question 1:

- Step 2 (Our method): 44%
- Step 3: 32%
- Step 5: 6%
- Step 11: 18%

Question 2: Answer questions about causal attribution

Task description: Answer questions about causal attribution.

Q: How would a typical person answer each of the following questions about causation? Drew, Kylie, Oliver, and Jen are regular customers at a small, local coffee shop. Given the selling price of the coffee and the cost of daily operation, the coffee shop will turn a profit if anyone orders coffee on a given day. Only one person ordering coffee is needed for the coffee shop to turn a profit that day. Drew, Kylie, Oliver, and Jen all usually order coffee on Tuesdays. This Tuesday, as usual, Drew ordered coffee. The same day, Kylie ordered coffee, Oliver ordered coffee, and Jen also ordered coffee. Since at least one person ordered coffee on Tuesday, the coffee shop made a profit that day. Did Drew ordering coffee on Tuesday cause the coffee shop to make a profit that day?

Options:

- Yes (Ground Truth)
- No

The large language model's reasoning trajectory with labeled step numbers is:

1. Okay, so I'm trying to figure out whether Drew ordering coffee on Tuesday caused the coffee shop to make a profit that day. Let me break this down step by step.
2. First, the coffee shop makes a profit if at least one person orders coffee. So, the key here is that only one order is needed for a profit. Now, on this particular Tuesday, Drew, Kylie, Oliver, and Jen all ordered coffee. That means there were four orders that day.
3. The question is specifically about Drew's order. Did Drew's action cause the profit? Well, since the shop only needs one order to make a profit, Drew's order alone would have been sufficient. But in this case, there were three other people who also ordered coffee. So, even if Drew hadn't ordered, the shop would still have made a profit because of the others.
4. This makes me think about the concept of causation. For Drew's order to be a cause, it should be necessary for the outcome. But since the outcome (profit) would have happened regardless of Drew's order, his action isn't the necessary cause. It's more like a contributing factor, but not the sole or necessary one.
5. In this case, Drew's order is sufficient on its own, but it's not necessary because others also ordered. So, while his order contributed, it wasn't the cause in the sense that it was required for the profit.
6. Another angle is to think about what would happen if Drew didn't order. If the shop still made a profit because of the others, then Drew's order wasn't the cause. It's like saying

if you have multiple people contributing to an event, each one isn't the sole cause unless their absence would change the outcome.

7. So, putting it all together, Drew's order wasn't the cause because the profit would have occurred without him. Therefore, the answer should be no.
8. The coffee shop made a profit because at least one person ordered coffee, but since others also ordered, Drew's order wasn't necessary for the profit. Therefore, Drew's action wasn't the cause.

Participant task: There are several steps listed; please select the step that you believe is most critical to the LLM's failure.

- Step 1
- Step 4
- Step 5
- Step 6

Results for Question 2:

- Step 1: 2%
- Step 4 (Our method): 68%
- Step 5: 22%
- Step 6: 8%

Question 3: Questions that involve enumerating objects and asking the model to count them

Task description: Questions that involve enumerating objects and asking the model to count them.

Q: I have a microwave, a lamp, four cars, a stove, a toaster, and a bed. How many objects do I have?

Ground Truth Answer: 9

The large language model's reasoning trajectory with labeled step numbers is:

1. Okay, so I need to figure out how many objects the person has. Let me list them out one by one to make sure I don't miss anything.
2. First, they mention a microwave. That's one object. Then there's a lamp, so that's two.
3. Next, they have four cars. Wait, four cars would count as four separate objects, right? So that adds four to the total.
4. After that, there's a stove, which is another object, making it five so far. Then a toaster, that's six. Finally, a bed, which brings the total to seven.
5. Let me double-check to make sure I didn't skip any. Microwave (1), lamp (2), four cars (4), stove (5), toaster (6), bed (7). Yep, that adds up to seven objects in total.
6. The person has a microwave, a lamp, four cars, a stove, a toaster, and a bed. Counting each item individually, the total number of objects is 7.

Participant task: There are several steps listed; please select the step that you believe is most critical to the LLM's failure.

- Step 2
- Step 4
- Step 5
- Step 6

Results for Question 3:

- Step 2: 4%
- Step 4 (Our method): 88%
- Step 5: 2%
- Step 6: 6%

Question 4: Multi-step arithmetic word problem

Task description: Multi-step arithmetic word problem.

Q: Jeremy is driving 600 miles to visit his parents. His average speed for this trip is 50 miles per hour. Every two hours of driving, he makes a rest stop for 15 minutes. His car gets 18 miles per gallon of gas, and his gas indicator comes on when he's used 15 gallons, at which point he stops to refill, which takes 10 minutes. How long, in minutes, does he take to get to his parents' house if he starts out with a full tank of gas?

Ground Truth Answer: 815

The large language model's reasoning trajectory with labeled step numbers is:

1. First, I need to determine how long it will take Jeremy to drive 600 miles at an average speed of 50 miles per hour. Next, I'll calculate the total driving time by dividing the total distance by his average speed.
2. Then, I'll figure out how many rest stops he'll need. Since he takes a 15-minute break every two hours of driving, I'll divide the total driving time by 2 to find the number of rest stops. After that, I'll calculate the total rest stop time by multiplying the number of rest stops by 15 minutes.
3. Next, I'll determine how much gas he uses during the trip by dividing the total distance by his car's fuel efficiency. Then, I'll find out how many times he needs to refill his gas

tank by dividing the total gallons used by the capacity of his gas tank.

4. Each refill takes 10 minutes, so I'll multiply the number of refills by 10 to get the total gas stop time. Finally, I'll add up the driving time, rest stop time, and gas stop time to get the total time for the trip in minutes.

5. Let's break down Jeremy's trip step by step to determine the total time he takes to reach his parents' house. ###1. Calculate the Total Driving Time Distance to Travel: 600 miles Average Speed: 50 miles per hour Driving Time = Distance/Speed = 600 miles/50 mph = 12 hours

6. ###2. Determine the Number of Rest Stops Jeremy makes a rest stop every 2 hours of driving. Number of Rest Stops = 12 hours/2 hours per stop = 6 stops. Each rest stop lasts 15 minutes. Total Rest Stop Time = 6 stops * 15 minutes = 90 minutes = 1.5 hours

7. ###3. Calculate the Gas Consumption and Stops Car Fuel Efficiency: 18 miles per gallon Total Gas Used: Total Gas = 600 miles/18 mpg = 33.33 gallons

8. Number of Gas Stops: Since the gas indicator comes on after using 15 gallons, we divide the total gas by the gas used per stop. Number of Gas Stops = 33.33 gallons/15 gallons = 2.22 stops

9. However, since he starts with a full tank, he will need to refill before the gas runs out. Therefore, he makes 2 gas stops. Each gas stop takes 10 minutes. Total Gas Stop Time = 2 stops * 10 minutes = 20 minutes = 1/3 hour

10. ###4. Calculate the Total Time Total Time = Driving Time + Rest Stop Time + Gas Stop Time Total Time = 12 hours + 1.5 hours + 1/3 hours = 13.833 hours Convert the total time to minutes: 13.833 hours * 60 minutes/hour = 830 minutes

Participant task: There are several steps listed; please select the step that you believe is most critical to the LLM's failure.

- Step 3
- Step 4
- Step 6
- Step 9

Results for Question 4:

- Step 3: 14%
- Step 4: 10%
- Step 6 (Our method): 76%
- Step 9: 0%

Question 5: Financial calculation word problem

Task description: Financial calculation word problem.

Q: Tom's cat needs an expensive surgery. He has had pet insurance for 24 months that costs $20 per month. The procedure cost $5000, but the insurance covers all but 20% of this. How much money did he save by having insurance?

Answer: 3520 (Ground Truth)

The large language model's reasoning trajectory with labeled step numbers is:

1. First, I need to calculate the total cost of Tom's pet insurance over 24 months. The insurance costs $20 per month, so multiplying that by 24 gives the total insurance expenditure.
2. Next, I'll determine the cost of the surgery. The surgery itself costs $5000, but Tom's pet insurance covers all but 20% of this amount. To find the amount covered by insurance,

I'll calculate 20% of $5000 and subtract that from the total surgery cost to get the out-of-pocket expense.

3. After that, I'll compare the total insurance cost with the out-of-pocket surgery cost to find the savings. By subtracting the out-of-pocket expense from the insurance cost, I can determine how much money Tom saved by having insurance. Finally, I'll present the calculated savings as the answer.

4. Let's break down the problem step by step to determine how much money Tom saved by having pet insurance. ###Step 1: Calculate the Total Pet Insurance Cost Tom has pet insurance for 24 months at a cost of $20 per month.

5. Total Insurance Cost = 24 months * $20/month = $480 ###Step 2: Determine the Surgery Cost Covered by Insurance The surgery costs $5000, and the insurance covers all but 20% of this amount.

6. Insurance Coverage = 100% - 20% = 80% Amount Covered by Insurance = 80% * $5000 = 0.8 * $5000 = $4000

7. ###Step 3: Calculate the Out-of-Pocket Expense Tom will pay the remaining 20% of the surgery cost out of pocket. Out-of-Pocket Expense = 20% * $5000 = 0.2 * $5000 = $1000

8. ###Step 4: Determine the Savings To find out how much Tom saved by having insurance, subtract the out-of-pocket expense from the total insurance cost. Savings = Total Insurance Cost - Out-of-Pocket Expense = $480 - $1000 = -$520

9. Wait, this result indicates that Tom actually spent $520 more with insurance compared to not having insurance. However, this doesn't make sense because insurance should reduce

his expenses. Let's re-examine the calculations. Revised Approach The correct way to calculate savings is:

10. The correct way to calculate savings is: Savings = Surgery Cost - (Surgery Cost - Insurance Coverage) - Total Insurance Cost Savings = $5000 - ($5000 * 0.8) - $480 = $5000 - $4000 - $480 = $520

Participant task: There are several steps listed; please select the step that you believe is most critical to the LLM's failure.

- Step 4
- Step 6
- Step 8
- Step 10

Results for Question 5:

- Step 4: 6%
- Step 6: 8%
- Step 8 (Our method): 56%
- Step 10: 30%

C.4.3 Overall Findings

Overall, the results indicate a strong alignment between the critical steps identified by GPO and human judgment. Across the five questions evaluated, the percentage of participants who selected the GPO-identified step as the most critical was 44%, 68%, 88%, 76%, and 56%, respectively. The strong alignment between the steps identified by our method and those recognized by humans as critical points indicates that our process effectively highlights key reasoning steps. This alignment

serves as qualitative validation for the core mechanism of GPO, reinforcing the hypothesis that its empirical improvements are well-founded.

C.5 Broader Impact

Beyond enhancing the reasoning capabilities of LLMs, GPO holds significant potential for advancing the trustworthiness and transparency of these models. By employing our method, we can identify and highlight the critical steps within the reasoning processes of LLMs. This capability not only aids in demystifying the decision-making pathways of these models but also empowers users to gain a deeper understanding of how conclusions are reached. Consequently, this increased clarity can foster greater trust in the outputs generated by LLMs. Furthermore, by making the reasoning process more transparent, stakeholders can more easily verify and validate the model's decisions, thereby enhancing the overall reliability and acceptance of LLMs in various applications. There could be potential negative societal impacts of our work. For example, if the critical steps are not properly highlighted, it could lead to misinformation and undermine trust in LLMs. However, we believe that the potential benefits of our work outweigh the potential risks, as we open-source our method and inspire more research on critical-step identification in LLMs.

C.6 Ethics Considerations

Our work involves the use of a user study to evaluate whether the identified critical steps align with human preferences. We have detailed the instructions we gave to participants in §C.4 to make it transparent and reproducible. The questions we designed only evaluate the critical steps; thus, they pose no harm to the participants. We **consulted the IRB office at our institution and received an exemption for this study**. Moreover, we do not collect any sensitive information from the users.

C.7 Theory

C.7.1 Online policy gradient algorithm

First, we introduce the performance difference lemma from [147].

Lemma 1 (performance difference lemma [147])**.** *For any policy π, π' and reward function r, we have*

$$V_0^{\pi}(s_0) - V_0^{\pi'}(s_0) = \sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi}}[\langle Q_h^{\pi'}(s_h), \pi_h(s_h) - \pi'_h(s_h)\rangle] \tag{C.1}$$

Based on the performance difference lemma, we can rewrite the regret as

$$\textit{regret} = \sum_{t=1}^{T}(V_0^{\pi^*}(s_0) - V_0^{\pi^t}(s_0)) = \sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}}[\langle Q_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h)\rangle] \tag{C.2}$$

where $\hat{Q}_h^{\pi_t}(.)$ is the estimated Q function for the policy π^t and $d_h^{\pi^*}$ denotes the state-action visitation measure given the optimal policy π^*. So we decompose the above equation to be

$$\begin{aligned}
\textit{regret} = &\sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}}[\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h)\rangle] && \text{(term(1))} \\
&+ \sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}}[\langle Q_h^{\pi^t}(s_h) - \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h)\rangle] && \text{(term(2))}
\end{aligned}$$

We present the following theorem to bound the term (1).

Theorem 3. *Suppose Assumption 1 holds, we have*

$$\sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}} \left[\langle \hat{Q}_h^{\pi_t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h)\rangle\right] \leq r_{\max} H \sqrt{\frac{T \log |\mathcal{A}|}{2}} \tag{C.3}$$

Proof of Theorem 3. First, we decompose term (1) into two parts.

$$\begin{aligned}
&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h) \rangle \\
=&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle + \langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^{t+1}(s_h) - \pi_h^t(s_h) \rangle \\
\leq&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle + \|\hat{Q}_h^{\pi^t}(s_h)\|_\infty \|\pi_h^{t+1}(s_h) - \pi_h^t(s_h)\|_1
\end{aligned}$$

By Assumption 1, we have $\|\hat{Q}_h^{\pi^t}(s_h)\|_\infty \leq r_{\max}$, which further implies

$$\begin{aligned}
&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h) \rangle \\
\leq&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle + r_{\max} \|\pi_h^{t+1}(s_h) - \pi_h^t(s_h)\|_1
\end{aligned}$$

Note that the policy update formula for the online policy gradient algorithm has a closed-form expression [116]

$$\pi_h^{t+1}(s_h) = \frac{1}{Z_h^t(s_h)} \pi_h^t(s_h) e^{\eta \hat{Q}_h^{\pi^t}(s_h)} \tag{C.4}$$

where $Z_h^t(s_h)$ is a normalization factor. Take the logarithm over both sides, and we have

$$\eta \hat{Q}_h^{\pi^t}(s_h) = \log Z_h^t(s_h) + \log \pi_h^{t+1}(s_h) - \log \pi_h^t(s_h) \tag{C.5}$$

Thus,

$$\begin{aligned}
&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle \\
=&\langle \frac{1}{\eta} (\log Z_h^t(s_h) + \log \pi_h^{t+1}(s_h) - \log \pi_h^t(s_h)), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle
\end{aligned}$$

Note that $\log Z_h^t(s_h) \sum_a [\pi^*(a|s_h) - \pi^{t+1}(a|s_h)] = 0$. We can reorganize the above equation as

$$\begin{aligned}
&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle \\
=& -\frac{1}{\eta} KL(\pi_h^*(s_h)||\pi_h^{t+1}(s_h)) + \frac{1}{\eta} KL(\pi_h^*(s_h)||\pi_h^t(s_h)) \\
&- \frac{1}{\eta} KL(\pi_h^{t+1}(s_h)||\pi_h^t(s_h))
\end{aligned}$$

Note that by Pinsker's inequality, we have

$$\|\pi_h^{t+1}(s_h) - \pi_h^t(s_h)\|_1 \leq \sqrt{2KL(\pi_h^{t+1}(s_h)||\pi_h^t(s_h))} \tag{C.6}$$

We further obtain the following bound for $\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle$:

$$\begin{aligned}
&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle \\
\leq& -\frac{1}{\eta} KL(\pi_h^*(s_h)||\pi_h^{t+1}(s_h)) + \frac{1}{\eta} KL(\pi_h^*(s_h)||\pi_h^t(s_h)) \\
&- \frac{1}{2\eta} \|\pi_h^{t+1}(s_h) - \pi_h^t(s_h)\|_1^2
\end{aligned}$$

Therefore, we have the following upper bound for $\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h) \rangle$:

$$
\begin{aligned}
&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h) \rangle \\
=&\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^{t+1}(s_h) \rangle + \langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^{t+1}(s_h) - \pi_h^t(s_h) \rangle \\
\leq&\frac{1}{\eta}[KL(\pi_h^*(s_h)||\pi_h^t(s_h)) - KL(\pi_h^*(s_h)||\pi_h^{t+1}(s_h))] + r_{\max}\|\pi_h^{t+1}(s_h) - \pi_h^t(s_h)\|_1 \\
&- \frac{1}{2\eta}\|\pi_h^{t+1}(s_h) - \pi_h^t(s_h)\|_1^2 \\
\leq&\frac{1}{\eta}[KL(\pi_h^*(s_h)||\pi_h^t(s_h)) - KL(\pi_h^*(s_h)||\pi_h^{t+1}(s_h))] + \frac{1}{2}\eta r_{\max}^2
\end{aligned}
$$

Summarizing over all horizons and over all iterations, we have

$$
\begin{aligned}
&\sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}} \langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h) \rangle \\
\leq&\sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}} \left\{ \frac{1}{\eta}[KL(\pi_h^*(s_h)||\pi_h^t(s_h)) - KL(\pi_h^*(s_h)||\pi_h^{t+1}(s_h))] + \frac{1}{2}\eta r_{\max}^2 \right\} \\
=&\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}} \left\{ \frac{1}{\eta}[KL(\pi_h^*(s_h)||\pi^1(s_h)) - KL(\pi_h^*(s_h)||\pi_h^{t+1}(s_h))] + \frac{1}{2}\eta r_{\max}^2 T \right\}
\end{aligned}
$$

Note that initially $\pi^1(a_h|s_h)$ is a uniform distribution over the action space $\mathcal{A}$. We further have

$$
\begin{aligned}
&\sum_{t=1}^{T}\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}} \langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h) \rangle \\
\leq&\sum_{h=0}^{H-1} \mathbb{E}_{s_h \sim d_h^{\pi^*}} \left(\frac{\log|\mathcal{A}|}{\eta} + \frac{1}{2}\eta r_{\max}^2 T \right)
\end{aligned}
$$

Let $\eta = \sqrt{\frac{2\log|\mathcal{A}|}{r_{\max}^2 T}}$ and we get

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{h=0}^{H-1}\mathbb{E}_{s_h\sim d_h^{\pi^*}}\langle \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h)\rangle \\
\leq &\sum_{h=0}^{H-1}\mathbb{E}_{s_h\sim d_h^{\pi^*}}\left(\frac{\log|\mathcal{A}|}{\eta} + \frac{1}{2}\eta r_{\max}^2 T\right) \\
= &r_{\max} H\sqrt{\frac{T\log|\mathcal{A}|}{2}}
\end{aligned}$$

We present Algorithm C.7.1 to obtain the estimate $\hat{Q}^{\pi^t}$ function.

Algorithm 6 Q function estimation

Input: Number of rollouts K, current policy π^t, reward r, hyperparameter η
Initialize: $\mathcal{D}_t = \emptyset$
for $k = 1, 2, \ldots, K$ **do**
  Collect one trajectory $\{(s_0^k, a_0^k, s_1^k, a_1^k, \ldots, s_{H-1}^k, a_{H-1}^k)\}$.
  Compute the advantage $A^{\pi^t}(s,a) = Q^{\pi^t}(s,a) - V^{\pi^t}(s)$ for each (s,a).
  Sample (s,a) with probability proportional to $\exp(\gamma A(s,a))$. Denote the sampled state-action pair as (s_m^k, a_m^k).
  Reset π^t to s_m^k and follow π^t to generate a trajectory $\{(s_m^k, a_m^k, \ldots, s_H^{k'}, a_H^{k'})\}$.
  Compute $q_m^k = \sum_{j=m}^{H-1} r_j$ and add $(s_m^k, a_m^k, y_m^k, q_m^k)$ into $\mathcal{D}_t$.
end for
Compute $\hat{Q}^{\pi^t} = \operatorname{argmin}_f \mathbb{E}_{\mathcal{D}_t}[(f(s,a) - q)^2]$.

Note that under Algorithm C.7.1, we actually reweight the state-action occupancy, i.e., reweighted state-action occupancy

$$d_h^{\rho}(s,a) = d_h^{\pi^t}(s,a)e^{\gamma A^{\pi^t}(s,a)}/Z_h(s)$$

where $Z_h(s)$ is a normalization factor

$$Z_h(s) = \sum_{a\in\mathcal{A}} d_h^{\pi^t}(s,a)e^{\gamma A^{\pi^t}(s,a)}$$

Then, we have the following lemma:

Lemma 2 ([148]). *With probability* $1-\delta$, *Algorithm C.7.1 guarantees that, for every* $t = 1, 2, \ldots, T$ *and* $h = 1, 2, \ldots, H$

$$\mathbb{E}_{(s_h,a_h)\sim d_h^\rho}[\hat{Q}_h^{\pi^t}(s_h,a_h) - Q_h^{\pi^t}(s_h,a_h)] \leq \frac{C' r_{\max}^2 \log(TH|\mathcal{F}|/\delta)}{K}$$

where C' *is an absolute constant.*

Now, we bound term (2). Note that

$$\begin{aligned}
&\mathbb{E}_{s_h\sim d_h^{\pi^*}}[\langle Q_h^{\pi^t}(s_h) - \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h) - \pi_h^t(s_h)\rangle] \\
\leq &|\mathbb{E}_{s_h\sim d_h^{\pi^*}}[Q_h^{\pi^t}(s_h) - \hat{Q}_h^{\pi^t}(s_h), \pi_h^*(s_h)]| + |\mathbb{E}_{s_h\sim d_h^{\pi^*}}[Q_h^{\pi^t}(s_h) - \hat{Q}_h^{\pi^t}(s_h), \pi_h^t(s_h)]| \\
= &|\mathbb{E}_{(s_h,a_h)\sim d_h^{\pi^*}}[Q_h^{\pi^t}(s_h,a_h) - \hat{Q}_h^{\pi^t}(s_h,a_h)]| \\
&+ |\mathbb{E}_{s_h\sim d_h^{\pi^*}, a_h\sim\pi_h^t(a_h|s_h)}[Q_h^{\pi^t}(s_h,a_h) - \hat{Q}_h^{\pi^t}(s_h,a_h)]| \\
\leq &\sqrt{\mathbb{E}_{(s_h,a_h)\sim d_h^{\pi^*}}\left[\left(Q_h^{\pi^t}(s_h,a_h) - \hat{Q}_h^{\pi^t}(s_h,a_h)\right)^2\right]} \\
&+ \sqrt{\mathbb{E}_{s_h\sim d_h^{\pi^*}, a_h\sim\pi_h^t(a_h|s_h)}\left[\left(Q_h^{\pi^t}(s_h,a_h) - \hat{Q}_h^{\pi^t}(s_h,a_h)\right)^2\right]} \\
\leq &2\sqrt{\left(\max_{h\in[H]}\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} w(s,a,h)\right)\mathbb{E}_{(s_h,a_h)\sim d_h^\rho}\left[\left(Q_h^{\pi^t}(s_h,a_h) - \hat{Q}_h^{\pi^t}(s_h,a_h)\right)^2\right]} \\
\leq &\frac{C r_{\max}^2 \log(TH|\mathcal{F}|/\delta)}{K}\sqrt{\max_{h\in[H]}\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} w(s,a,h,\gamma)}
\end{aligned}$$

where $w(s,a,h,\gamma) = \frac{d_h^{\pi^*}(s,a)}{d_h^\rho(s,a)}$ is the density ratio between $d_h^{\pi^*}(s,a)$ and $d_h^\rho(s,a)$ and C is a constant.

Therefore, with probability $1-\delta$, we have

$$\sum_{t=1}^{T}\sum_{h=0}^{H-1}\mathbb{E}_{s_h\sim d_h^{\pi^*}}[\langle Q_h^{\pi^t}(s_h)-\hat{Q}_h^{\pi^t}(s_h),\pi_h^*(s_h)-\pi_h^t(s_h)\rangle] \tag{C.7}$$

$$\leq TH\frac{Cr_{\max}^2\log(TH|\mathcal{F}|/\delta)}{K}\sqrt{\max_{h\in[H]}\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}}w(s,a,h,\gamma)} \tag{C.8}$$

$$\tag{C.9}$$

Now, we show that the density ratio for high-advantage actions decreases as γ increases.

Based on Theorem 1 in [149], we can rewrite $d_h^\rho(s,a)$ as $d_h^\rho(s,a)=d_h^{\pi^t}(s,a)\pi^{t+1}(a|s)^\gamma$. The density ratio $w(s,a)$ is

$$\frac{d_h^{\pi^*}(s,a)\sum_{\mathcal{A}}d_h^{\pi^t}(s,a)\pi^{t+1}(a|s)^\gamma}{d_h^{\pi^t}(s,a)\pi^{t+1}(a|s)^\gamma}$$

Suppose $d_h^{\pi^t}(s,a)\propto\exp(\beta_1 A^{\pi^*}(s,a))$ and $\pi^{t+1}(a|s)\propto\exp(\beta_2 A^{\pi^*}(s,a))$ for some parameter $\beta_1<\beta_2$. This is reasonable since the updated policy π^{t+1} is better than the current policy π^t. Take the logarithm and we have

$$\log w(s,a,h,\gamma)=\log d_h^{\pi^*}(s,a)+\log\left(\sum_{\mathcal{A}}e^{(\beta_1+\beta_2\gamma)A^{\pi^*}(s,a)}\right)-(\beta_1+\beta_2\gamma)A^{\pi^*}(s,a) \tag{C.10}$$

Partial derivative of $\log w(s,a,h,\gamma)$ with respect to γ is:

$$\frac{\partial}{\partial\gamma}(\log w(s,a,h,\gamma))=\beta_2\left[\frac{\sum_{\mathcal{A}}A^{\pi^*}(s,a)e^{(\beta_1+\beta_2\gamma)A^{\pi^*}(s,a)}}{\sum_{\mathcal{A}}e^{(\beta_1+\beta_2\gamma)A^{\pi^*}(s,a)}}-A^{\pi^*}(s,a)\right] \tag{C.11}$$

For a high-advantage action such as $a^*=\arg\max_a A^{\pi^*}(s,a)$, the softmax-weighted average in the bracket is no larger than $A^{\pi^*}(s,a^*)$. Therefore, the derivative is non-positive for a^*, meaning that increasing γ reduces the density ratio on high-advantage actions and motivates the advantage

reweighting strategy.

C.7.2 Preference optimization

Proof. We follow the proof strategy outlined in Theorem 6.1 of [66]. To derive the desired result, we begin with the key observation that DPO [58] is equivalent to optimizing a KL-regularized expected reward objective, where the reward function is used to define preferences via the Bradley-Terry model. Specifically, the optimal policy $\pi^*(\cdot \mid \cdot)$ that maximizes the following regularized objective:

$$\max_{\pi} \mathbb{E}_{x\sim\mu,\, y\sim\pi(\cdot|x)}\left[r(x,y)\right] - \beta D_{\mathrm{KL}}\left(\pi(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\right)$$

is given in closed form as:

$$\pi^*(y \mid x) \propto \pi_{\text{ref}}(y \mid x) \cdot \exp\left(\frac{r(x,y)}{\beta}\right). \tag{C.12}$$

This optimal policy can be recovered by applying DPO to preference-labeled pairs (x, y_1, y_2), where preferences are sampled from the Bradley-Terry model [114] defined by the reward function r:

$$p(y_1 \succeq y_2 \mid x) = \frac{\exp(r(x,y_1))}{\exp(r(x,y_1)) + \exp(r(x,y_2))}. \tag{C.13}$$

Given this background, we consider preference pairs of the form $(x, [y_{0:i-1}, y_i^+], [y_{0:i-1}, y_i^-])$, where both continuations are sampled from π_{ref}: $y_i^+ \sim \pi_{\text{ref}}(\cdot \mid x, y_{0:i-1}), \quad y_i^- \sim \pi_{\text{ref}}(\cdot \mid x, y_{0:i-1})$, and the preference is determined based on the advantage estimates $A^{\pi_{ref}}(x, y_{0:i-1}; \cdot)$. Combined with Eqn. C.12, this yields the following equation:

$$\pi_i(y_i \mid x, y_{0:i-1}) \propto \pi_{\text{ref}}(y_i \mid x, y_{0:i-1}) \cdot \exp\left(\frac{A^{\pi_{ref}}(x, y_{0:i-1}; y_i)}{\beta}\right). \tag{C.14}$$

Moreover, since the optimal advantage-weighted RL policy that maximizes Eqn. C.14 coincides with the solution in Eqn. 5.5, the proof is complete.