Enhancing Policy Transfer in Action Advising for Reinforcement Learning

Yue (Sophie) Guo

CMU-CS-24-118

May 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Katia Sycara, Chair
Fei Fang
Zico Kolter
Matthew E. Taylor (University of Alberta)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Copyright © 2024 Yue (Sophie) Guo

This research was sponsored by the Air Force Office of Scientific Research under award number FA95501810251, the Air Force Research Laboratory under award numbers FA95501510442 and FA95501810097, the Defense Advanced Research Projects Agency under award numbers HR00111920012, HR00111920028, and HR001120C0036, the Department of Agriculture under award number 20176700726152, Honda Research Institute under award number RPS11POHRI0054202, the National Science Foundation under award number 1724222, the University of Pittsburgh (Army) under award numbers AWD000053154186671 and AWD000053154186651, and the U.S. Army Contracting Agency under award number W911NF1610514. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Reinforcement Learning, Multi-agent System, Transfer Learning, Knowledge Transfer, Robotics, Deep Learning

Abstract

As human students benefit from teachers' advice to accelerate their learning, so could agents benefit from advice. Agents can not only learn from humans, e.g. via human expert demonstrations, but also from other agents. This thesis focuses on action advising, a knowledge transfer technique built upon the teacher-student paradigm of reinforcement learning. In this approach, the teacher agent provides action advice calculated from its policy given the student's observations.

Although action advising has been studied over the past decade, the focus of the related work has primarily been on when to advise. We extend the current state-of-the-art by studying the following additional challenges: (1) In existing work, advice is given without explaining the rationale behind it. The student can therefore hardly understand teacher's decisions, or internalize the knowledge to generalize teacher's advice; (2) In many situations, the teacher might be suboptimal in a new environment, but there are no current approaches that enable the student to discern when some particular pieces of advice might not be applicable; (3) No present techniques enable the teacher to evaluate the quality of its advice before giving it to the student; (4) If the student interacts in a new environment, the teacher has limited knowledge as it does not collect the student's data; (5) The teacher with a fixed pre-trained policy might not be able to provide flexible advice; (6) The advice has rarely been applied to human students.

In this thesis, we present our solutions to tackle the aforementioned challenges and exhibit the empirical effectiveness of our proposed methods. We also propose potential pathways for subsequent research. Our ultimate goal is to delve into the intricacies and potentials of action advising in reinforcement learning, thereby directing future advancements in the field.

Additionally, this thesis broadens its scope by examining further applications of transfer learning, showcasing its utility and adaptability in varied contexts beyond action advising. These explorations contribute to a deeper understanding of transfer learning's potential within the broader field of reinforcement learning.

Acknowledgments

I am thankful to my advisor, Professor Katia Sycara. She has given me ample encouragement and support as I explored the areas of model transferability, reinforcement learning, and human-AI teaming. I have not only learned methodologies for conducting research but also acquired valuable skills in communication, task management, and stress handling from her. I deeply appreciate the time and effort she has invested in me.

I am grateful to the PhD thesis committee members for their valuable advice on my thesis. First, I would like to thank Professor Fei Fang, whose class on multi-agent learning greatly inspired my thesis idea and whose collaboration contributed to a publication and to this thesis work. I am also thankful to Professor Zico Kolter for his time discussing the directions of deep reinforcement learning during my early years in the PhD program and for his guidance both in his general AI class and externally. Furthermore, I appreciate Professor Matthew E. Taylor, a pioneer in the field of action-advising policy transfer, for guiding me towards valuable research topics in this area.

My gratitude extends to the postdoctoral fellows and research scientists in our lab. I thank Dr. Joseph Campbell, Dr. Simon Stepputtis, and Dr. Dana Hughes for their guidance on my projects and thesis and for contributing to my publications. I have learned crucial methodologies for conducting research experiments and have improved my engineering skills. I am also grateful to my lab-mates and collaborators, including but not limited to Professor Michael Lewis, Huao Li, Xijia (Pony/Polina) Zhang, Fiona Xie, Ruiyu Li, Rohit Jena, and Boshi Wang. I also appreciate discussions from other lab-mates such as Muhan Lin, Shuyang Shi, Weihao (Zack) Zeng, Seth Karten, Woojun Kim, Wenhao Luo, Sha Yi, Yaqi Xie, Shreya Sharma, Sarthak Bhagat, Zongyue Zhao, Stephanie Milani, and Vidhi Jain and fellow PhD students Shuqi Dai and Chun Kai Ling. Additionally, I thank DARPA, AFRL, and Honda Research Institute (HRI) etc. for funding our projects and providing valuable feedback during events and meetings. Especially discussions from HRI with Vaishnav Tadiparthi, Behdad Chalaki, Hossein Nourkhiz Mahjoub, Jovin Dsa, Ehsan Moradi Pari, and Kwonjoon Lee greatly enriched the thesis.

Lastly, I am thankful to my husband, Jinnan (Abner) Zhang, for his warm support and care for me and our lovely dog, Rara. I am also grateful to my friends Weijia He and Jingyi Lyu, and my mother Hongdi Zhang and father Yilin Guo (and their corresponding family members), as well as my parents-in-law Fengqin Li and Dongming Zhang, who have given me love and courage to overcome challenges.

Contents

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Related Work . . . 2
- 1.3 Preliminaries . . . 4
- 1.4 Overview of Chapters . . . 5

2 Providing Explanations and Building Student's Internal Model 7
- 2.1 Background . . . 8
- 2.2 Methodology . . . 9
  - 2.2.1 Explainable Action Advising . . . 9
  - 2.2.2 Decision Tree Reconstruction . . . 12
- 2.3 Experiment . . . 13
  - 2.3.1 Experimental Setup . . . 14
  - 2.3.2 Benchmarks . . . 15
  - 2.3.3 Results . . . 16
- 2.4 Conclusion . . . 18

3 Student Rejecting Bad Advice Based on Explanations and its Internal Model 19
- 3.1 Background . . . 20
- 3.2 Methodology . . . 21
- 3.3 Experiment . . . 23
  - 3.3.1 Experiment Setup . . . 23
  - 3.3.2 Comparison to Fine-tuning . . . 24
  - 3.3.3 Self-Reflection . . . 25
  - 3.3.4 Comparison to Ablated Version . . . 26
- 3.4 Conclusion . . . 29

4 Teacher's Introspection for Preventing Bad Advice 30
- 4.1 Methodology . . . 31
  - 4.1.1 Introspection: . . . 32
  - 4.1.2 Off-Policy Correction: . . . 32
  - 4.1.3 Algorithm: . . . 32
- 4.2 Experiments . . . 34
  - 4.2.1 Experimental Setup . . . 36

4.2.2 Results - Gridworld . . . 37
4.2.3 Results - Atari . . . 40
4.3 Conclusion . . . 47

5 Teacher's Adaptation to Improve Advice based on Estimating Student's Rewards 48
5.1 Background . . . 49
5.2 Methods . . . 50
5.3 Experiments . . . 54
5.3.1 Action Advising Comparisons . . . 56
5.3.2 Transfer Learning Comparisons . . . 59
5.3.3 Adaptability with respect to Policy Variance . . . 60
5.4 Supplementary Material . . . 61
5.4.1 Hyper-parameters . . . 61
5.4.2 Reward Similarity . . . 64
5.4.3 Bidirectional Navigation . . . 65
5.4.4 Joint Data Training . . . 67
5.5 Extending with Predicting Transition Function . . . 68
5.5.1 Algorithm . . . 68
5.5.2 Experiments and Results . . . 70
5.5.3 Environments . . . 70
5.5.4 Results and Analysis . . . 71
5.6 Conclusion . . . 75

6 Human Acceptance to Advice and Explanations from Artificial Teacher 76
6.1 Train the Teacher . . . 78
6.1.1 RL Expert . . . 78
6.1.2 Representable DT Teacher . . . 79
6.1.3 Keyboard Interaction Interface . . . 79
6.2 Amazon Mechanical Turk (MTurk) Experiment . . . 82
6.2.1 Design of Experiment . . . 82
6.2.2 MTurk Samples Generation . . . 82
6.2.3 Experiment Results . . . 87
6.3 Conclusion . . . 89

7 Utilizing a Language Model to Improve Explanations to Humans 90
7.1 Background . . . 91
7.2 Methods . . . 93
7.2.1 Distilling a Decision Tree . . . 93
7.2.2 Behavior Representation Generation . . . 93
7.2.3 In-Context Learning with Behavior Representations . . . 94
7.2.4 Interactive Explanations . . . 94
7.3 Experiments . . . 96
7.3.1 Does Our Approach Produce Useful Explanations? . . . 97
7.3.2 Is User Interaction Helpful? . . . 99

7.3.3 Does Our Approach Decrease Hallucination? 100
7.4 Conclusion 101

8 A Naive Finetuning Approach to Transfer Prediction 102
8.1 Background 103
8.2 METHODS 105
8.2.1 USAR Domains 105
8.2.2 Agent Observations 107
8.2.3 Navigation Prediction 108
8.2.4 Triage Strategy Prediction 111
8.3 EXPERIMENTS 112
8.3.1 Evaluation Metrics 112
8.3.2 Navigation Prediction with Transfer Learning 114
8.3.3 Triage Strategy Prediction with Transfer Learning 119
8.4 Conclusion 120

9 An Automatic Parameter Selection Method for Network-based Transfer 121
9.1 Background 122
9.2 Methods 126
9.3 Experiments 129
9.3.1 Traditional Network Transfer By Hand 130
9.3.2 RL Variations Trained Over Time 131
9.3.3 Best Models of RL Variations 133
9.3.4 Layer-level Transfer 134
9.3.5 Runtime Comparison 135
9.4 Conclusion 136

10 Conclusion 137

Bibliography 138

List of Figures

2.1 Overview of EAA. As the student makes observations during training, one of three cases may occur. **1)** The student has previously received advice for a similar state and is able to identify it through an explanation in its internal model, then follow the associated action. **2)** The student has not previously received advice, but is now given advice by the teacher and stores the explanation in its internal model, then follows the advised action. **3)** The student has not previously received advice and the teacher is not currently providing advice; the student follows its own policy. In cases where the teacher is trained in a different environment than the student, the student may choose to reject unhelpful advice. 10
2.2 The decision tree reconstruction process that is performed when updating a student's internal model $\hat{\pi}$. The student receives an explanation decision path $\mathcal{F}_1 \wedge ! \mathcal{F}_2$ from the teacher's distilled policy $\hat{\pi}^*$, finds the last common node $\mathcal{F}_1$ currently in its model, and adds all subsequent missing nodes – in this case only $!\mathcal{F}_2$. 12
2.3 Environment Screenshots. The 4 and 14 room environment have a medic (red circle), engineer (blue circle) and victims (red plus). 13
2.4 Comparing Explainable Action Advising (Early: Red, Alternative: Purple, Importance: Yellow, Mistake Correcting: Green) with Action Advising (Blue) on the PacMan, HFO, and USAR (four and fourteen room) environments. Dash-Dotted lines indicate when advising budget is exhausted. Results are averaged over 5 independent trials except for 14-room over 3. 16
2.5 Baseline of Pacman (left) and 14-room USAR comparing EAA to SOTA algorithms. Results are averaged over 3 random seeds except for Learn to Teach which uses 1 due to run-time. 17

3.1 Transfer learning performance for students training in the USAR environment with rubble by a teacher who was originally trained without rubble. The teacher only has access to the extracted decision tree where the teacher NN is not present in (b), compared to (a) where the teacher NN is present. Results are averaged over 5 independent trials. 24
3.2 A comparison of EAA with reflection, called EAA (Explore), to standard EAA. The student is pretrained in a source environment (4-room no rubble) and transferred to a target environment (4-room with rubble). Results are averaged over 5 independent trials. 25

3.3 A comparison of an ablated version of EAA which always takes the teacher's advice, EAA (Always Accept), to standard EAA when the teacher has access to both its original neural network policy and the extracted decision tree (DT + NN). 26
3.4 A comparison of an ablated version of EAA which always takes the teacher's advice, EAA (Always Accept), to standard EAA when the teacher only has access to its extracted decision tree (DT). 27

4.1 The source and target Gridworld tasks. Agent start (red triangle), goal (green square), and key positions are randomized. The agent must learn to navigate to the goal. The source task contains no locked doors while the target task contains exactly one. 35
4.2 The source task (Air Raid) and a subset of target tasks in Atari (Phoenix, Space Invaders, Carnival, and Demon Attack shown). The nature of the task is dependent on the game. 35
4.3 The training curves between our proposed method and fine-tuning methods (left) and transfer methods methods (right) given a Gridworld environment where the source and target tasks are similar enough such that the teacher issues helpful advice. 37
4.4 The training curves for a Gridworld environment where the source and target tasks differ such that the teacher largely gives unhelpful advice. 38
4.5 A sequence from a single training episode showing the student's sampled action (red) and the teacher's advised action (blue) in Gridworld. Bold text indicates the executed action as determined by IAA given $\epsilon = 0.1$. 39
4.6 The relative performance of our proposed approach and the transfer learning baseline models to the Baseline policy learned in the target task without any transfer. Each value represents the percentage of improvement in the mean reward after 10M environment steps over four independent training runs with randomly initialized weights. The target task is shown along the x-axis with Air Raid used as the source task in all scenarios. 40
4.7 The training curves between our proposed method and transfer learning methods (left) and advising methods (right) for Carnival. Each line is the average reward over four independent training runs for 10M environment steps, with the shaded region indicating standard deviation. 41
4.8 The training curves between our proposed method and transfer learning methods (left) and advising methods (right) for Demon Attack. Each line is the average reward over four independent training runs for 10M environment steps, with the shaded region indicating standard deviation. 42
4.9 Training curves for multiple introspection thresholds of IAA for different Atari tasks. 43
4.10 The relative performance of IAA compared to the distribution shift of each Atari task. 44
4.11 The amount of advice issued by each advising algorithm for the game Carnival. . 45
4.12 The amount of advice issued across Atari games by IAA given $\epsilon = 0.15$. 45

4.13 A sequence of states from a single training episode showing the student's sampled action (red) and the teacher's advised action (blue) in Carnival. Bold text indicates the executed action as determined by IAA given $\epsilon = 0.45$. 46

5.1 An overview of Adaptive Reward Teaching. The teacher initially issues action advice to the student from a policy trained in the source task – leading to low rewards in the target task due to differences in the reward functions. For example, the teacher could be a ghost chaser while the student is a conservative player that always avoids ghosts and eats food. The teacher has a much higher reward of eating ghosts than the student, however, it would not find this out until examining the data collected by the student. After observing the student's trajectory returns, the teacher adapts its policy using its so as to maximize the student's reward in the target task. 50
5.2 Search and Rescue: The agent (blue circle) navigates to the room where a rubble is removable, and a victim represented by red cross needs to be rescued. 54
5.3 Skiing: The agent (red) needs to avoid the trees (green) and gates (blue). It can pass the gates if it goes through the empty space between gate pairs. 55
5.4 Pacman: The pacman (yellow) eats up food (green) and power pellets (blue). In the default mode, the ghosts (pink) chase the pacman and can eat it up; After the pacman eats up a power pellet, it can eat ghosts. 55
5.5 Training curves for a student which receives action advice from our proposed method as well as the action advising variants described in Sec. 5.3. Left-to-right: USAR, Skiing, and Pacman. 58
5.6 Training curves for a student which receives action advice from our proposed method as well as the methods defined in Sec. 5.3.2. Left-to-right: USAR, Skiing, and Pacman. 59
5.7 Training curves for our proposed method as well as a student advised by a non-adaptive teacher and a student with no advising. The reward for removing each piece of rubble varies from 0.5 (left), 0.25 (middle), and 0.125 (right). 60
5.8 USAR 10 Rubble: Percentage of removed rubble (out of 10) and triaged victims (out of 1) over 1000 rollouts across five student seeds each. 61
5.9 Hyper-parameters 62
5.10 The advising probability of each environment. 63
5.11 Improvement over the reward difference. 65
5.12 Environments of the teacher and student in the bidirectional navigation task. . . . 66
5.13 Baseline and benchmark results of the bidirectional navigation task 66
5.14 Compare with teacher using both its data and the collected student data to train its reward network. 67
5.15 An overview of the Reward-Adaptive Teacher with the predicted transition function. The teacher initially issues action advice to the student from a policy trained in the source task – leading to low rewards in the target task due to differences in the reward functions. After observing the student's trajectory returns, the teacher adapts its policy using its internal world model so as to maximize the student's reward in the target task without requiring direct access to the environment. . . . 68

5.16 RAA (in orange) compared to ablated methods. The shaded region denotes the adaptation period, which happens every 2 iterations in USAR-4Room, 5 in USAR 14-Room, and only once in Navigation. No advising occurs prior to the first β episodes. 71
5.17 RAA (in orange) compared to state-of-the-art transfer learning approaches. . . . 72
5.18 Rollout of an RAA-adapted policy given real-world student trajectories. 73
5.19 Policy adaptation results in simulation and real-world across 50 trajectories respectively. A consistent initialization of Pre-Adaptation and Post-Adaptation is kept. 73
5.20 Throughout the teacher's fine-tuning process, its success rate in its source environment (gray) gradually decreases while the one in the target environment (pink), which reflects student's rewards, grows. This shows the teacher gradually learn to adapt to student's reward and is able to make alternative advice based on student's need. 74

6.1 The start of pacman game. 77
6.2 The advice and explanation received. 80
6.3 MTurk Samples (1) 82
6.4 MTurk Samples (2) 83
6.5 MTurk Samples (3) 84
6.6 MTurk Samples (4) 85
6.7 MTurk Samples (5) 86
6.8 Acceptance Group 87
6.9 Neutral Group 88

7.1 Overview of our three-step pipeline to explain policy actions: left: A black-box policy is distilled into a decision tree; middle: a decision path is extracted from the tree for a given state which contains a set of decision rules used to derive the associated action; right: we utilize an LLM to generate an easily understandable natural language explanation given the decision path. Lastly, a user can ask further clarification questions in an interactive manner. 92
7.2 Interactive Conversation: The user is asking more clarification questions about the initial explanation generated by our method. The users are fluent English speakers and reside in the United States. From the right side, the first prompt of providing information to the agent is from the human researchers, and the second prompt of asking questions is an example question asked by human participants. The teacher agent's responses are from the left side. 95
7.3 Participant preference when presented with two explanations and asked to choose which is most helpful to understand agent behavior: Ours vs. Template (top) and Ours vs. Human (bottom) over 320 responses. 97
7.4 Frequency of selected features when explaining actions. Features highlighted in red were part of the examples provided in the LLM's prompt. 97
7.5 Helpfulness of interaction after presented with an explanation with respect to optimal and sub-optimal policies across 20 responses. 99

7.6 Number of hallucinations that are factually wrong (left) or have incorrect assumptions (right) across our samples. 100

8.1 Sparky 106
8.2 Falcon 106
8.3 Map of the three versions of Falcon: easy, medium, and hard collected in [67]. Grey indicates walls, Magenta indicates blockages, and Cyan is for openings. . . 107
8.4 The Architecture of the Navigation Prediction Process 108
8.5 Transfer from Map Sparky to Map Falcon 115
8.6 Transfer from Map Sparky to Map Falcon 115
8.7 Transfer from Map Falcon Med to Map Falcon Hard and Easy 116
8.8 Transfer from Map Falcon Med to Map Falcon Hard and Easy 116
8.9 Transfer from Map Falcon to Map Sparky 117
8.10 Transfer from Map Falcon to Map Sparky 117
8.11 Compare with Lower Level abstraction 118
8.12 Compare with Lower Level abstraction 118
8.13 Transfer learning on triage strategy prediction 119

9.1 A visualization of the procedure. 126
9.2 Compare freezing and pretraining residual blocks by hand. 130
9.3 Compare RL variations with 60 folds and 150 folds of data to train. 132
9.4 Compare best RL variations. 133
9.5 Layer level Transfer 134
9.6 Runtime of the Experiments 135

List of Tables

4.1 Summary of transfer performance shown in Fig. 4.6. Positive Transfer Ratio is the number of tasks for which performance improved, Minimum Transfer is the smallest relative performance achieved over all tasks, and Maximum Transfer is the greatest relative performance achieved. Bold represents the best value for each metric. PTR, MinT, MedianT, and MaxT refer to Positive Transfer Ratio, Minimum Transfer, Median Transfer, Maximum Transfer. 40

5.1 Important Hyper-parameters 63
5.2 RL Training Hyper-parameters 64

6.1 Performance evaluation of the best RL policy. 78
6.2 Performance evaluation of DT policy. 79

Chapter 1

Introduction

1.1 Motivation

We believe that agents could benefit from advice from other agents. This could prevent them spending time exploring unpromising directions and accelerate learning. One suitable area for action advising is Reinforcement learning (RL). RL's success spans numerous fields, such as learning to play various games [84, 133, 134], and training complex policies in autonomous driving [81, 113, 158]. However, even the most accomplished RL algorithms have not fully resolved the issue of sample inefficiency in high-dimensional tasks, necessitating large amounts of training data [127, 162]. High sample complexity remains a bottleneck in various algorithms [32], resulting in extended training times. In the case of real-world robots, where RL is used for complex tasks, this poses significant challenges due to the extended time required for sample collection, and the risk of damage to costly robots or other equipment during prolonged exploratory long-runs.

Recent research in action advising for RL has seen a rise over the past decade [33, 110, 144, 172] . Central to action advising in RL is the teacher-student paradigm, where a teacher agent, trained in a source task, offers advice to a student agent trained in a target task. The goal of teacher-student action advising is expediting the learning process of the student agent in previously unseen situations by transferring the knowledge of the teacher agent to the student. These methods present potential in various scenarios, including but not limited to: (1) situations where a direct transfer of all the teacher's knowledge is infeasible, such as copying a policy from a different model or from a human; (2) situations when the teacher is sub-optimal, and it would be beneficial to transfer only a specific subset of its knowledge, instead of a complete application; or (3) situations where the teacher provides a curriculum for incremental student learning. In teacher-student action advising, the teacher offers advice in the form of state-action pairs to the student, suggesting a recommended course of action. This differs from related techniques such as inverse reinforcement learning [65], intention inference [50], and offline reinforcement learning [45], which rely on an offline dataset of expert trajectories. Most of the work on teacher-student action advising to date focuses on when and how to provide advice [144].

We extend the current literature by providing methods to answer the following research questions:

1. Can the teacher provide explanations for the given advice in addition to the advice itself,

and let the student assimilate them?

2. If the teacher is non-expert and as a result may give inappropriate advice, how can the student discern if it should reject the advice?
3. From the teacher's perspective, how can it filter out bad advice, or suggest advice that adapts to the student's preferences?
4. What is the human's acceptance towards the advice?

Addressing these issues could significantly enhance the teacher-student action advising framework, leading to more effective and efficient knowledge transfer.

Beyond the primary focus on action advising in RL, this thesis expands its exploration to assess the utility of transfer learning in various contexts, enriching our comprehension of its versatility and strategic applications. While we delve deeply into action advising to address sample inefficiency and knowledge transfer challenges in RL, the inclusion of further chapters broadens our discourse, demonstrating how transfer learning principles can be adeptly applied to optimize learning and decision-making in distinct domains. This broader examination is integral to contextualizing our methodology's strengths and delineating potential applications. By contrasting these diverse applications with our focused methodologies in action advising, we gain valuable insights into our approach's advantages and the prospective directions for its application. These additional explorations underscore the adaptability and wide-reaching impact of transfer learning strategies, showcasing their critical role in propelling the field of reinforcement learning forward and highlighting their potential to foster efficient learning and robust decision-making.

1.2 Related Work

Work on action advising in the field of reinforcement learning (RL) encourages knowledge transfer from the teacher agent to the student agent. Most of this research considers the advising process as happening during the training of a student agent, where the student utilizes the teacher's knowledge to accelerate and improve its own training process. Early work at the end of the last century shed light on the direction that advice provision may come from an external observer [97]. Although the number of studies has grown significantly over the past decade, they stem from earlier works where the idea of agent advising was first introduced [83, 98, 143]. In these studies, RL agents were considered teachable and could be guided to improve their training efficiency. The instrumental assistance came from human agents who could potentially map their expertise to RL tasks effectively, while a mapping based on domain knowledge had to be pre-designed by the users to ensure the knowledge could be transferable [145, 146]. In the field of action advising for reinforcement learning over the recent decade, the teacher guides the student agent by recommending suitable actions within a given state, based on the student's observations. This enables the student agent to swiftly uncover effective actions, leading to efficient explorations, especially within vast observation spaces, rather than relying solely on self-exploration.

This advising approach typically embeds the teacher-student paradigm where both of the agents are RL agents, and encompasses a teacher's knowledge transfer to a student in the form of state-action pairs. A survey paper [30] lists and analyzes many significant approaches which deploy the proposed paradigm. Foundational work ventured into the domain with budget constraints as an important measuring metric to determine if the advising was efficient, and also offered heuristic methodologies for issuing advice in single-agent tasks [144] of simple video games. Subsequent studies delved into various teacher-student dynamics that were defined by who initiated the advice, for example, (a) student-initiated [32], (b) teacher-initiated [38], and (c) jointly-initiated [4]. Most of the work still considers the teacher initiating the advice, as the teacher is often more knowledgeable to determine the situations encountered by the student. Additionally, many of the approaches center around *when* and *which* action should be advised. The core of the work considers how advice allocation can maximize its effect given the limited budget of advising. Framing advising as a reinforcement learning task, where the teacher agent's reward is based on the student's performance [38, 110, 177], has received much attention. This approach tackles the problem of when to issue advice in a more robust way, but prolongs the training process because every time the teacher improves its advising technique in a session, the student has to converge from scratch in its training episodes. On the other hand, some work advances previously established heuristic methods for deep reinforcement learning [31, 70], especially when the focus of the work is not when to advise. Typically the work involves comparing various heuristic methods or solely implementing the most feasible one.

Within the setting of multi-agent reinforcement learning (MARL), action advising approaches consider each agent as both a student and a teacher, switching roles alternately [31, 80, 110]. This means the agents teach each other to complete a goal together for the entire team. No prerequisite expert knowledge is required for the agents, thus all agents start in inexperienced roles and gradually learn over the training process. These approaches do not assume stationarity in the MARL setting, but do assume that the policies of other teammates are temporarily fixed while each agent is being optimized. In order to determine whether to accept advice, some studies have proposed agents that can probabilistically reject advice from their teammates [31], however, this approach does not comprehensively consider the quality of the advice. A more robust approach considers agents learning to teach each other by switching roles of teachers and students [110], where the quality of the advice is further measured and the mechanism determines if it is helpful to give advice. To enhance efficiency in the framework of agents learning to teach each other, option learning is used to train a high-level agent policy [160] so as to handle more complicated situations by abstracting the atomic actions away. Additionally, distillation and value-matching techniques are utilized to aggregate knowledge from various homogeneous agents, simplifying the multi-agent task [153]. Besides the framework of agents learning to teach each other, studies have also delved into teaming with agents who are in partially observable environments in decentralized settings [69]. Here, heuristic teaching strategies are employed to offer advice to a decentralized team. This approach contrasts with the centralized settings, which use a single centralized teacher to oversee the entire space and communicate advice to the student agents [57], who do not have a complete global awareness.

Recent studies have pioneered the reuse of advice, structured as state-action pairs, in order to save advising budgets and maximize communication efficiency. One such methodology involves storing state-action pairs directly in memory, and selectively reusing the advice later [172]. In this work, multiple methods are used to determine if the storage has reached its maximum capacity. An imitation learning module has been proposed that mimics the teacher's advice, enabling the generation of advice even after the advice budget has been used up [70]. This further improves the reuse of fixed advice by providing similar pieces of advice. Additionally, researchers have considered a teacher that can provide qualitative evaluations of a given state, when advice is required for it. This provides students with an understanding of how favorable a particular state is [6], which may improve the selection of advice. Nevertheless, while these approaches successfully save the budget, they encounter difficulties when generalizing action advice to unfamiliar states and environments.

Human teachers have an innate ability to first determine if their advice is appropriate, and adjust their advice based on the observed experiences of their students. Even though the human teachers are not perfect, it is undoubtedly more difficult for artificial teachers to make adjustments. Most recent approaches in action advising, which rely on an expert policy for advising, struggle when the teacher's domain differs from the student's domain. This especially happens when agents learn to teach each other and all agents are homogeneous [31, 111]. In a scenario where the teacher is sub-optimal and the student's domain differs from the teacher's, the utility of the teacher heavily relies on the quality of its advice [165]. This research problem, inspired by DAGGER [125], was addressed [165] through a curriculum of teachers who constantly observe and adapt their behaviors. This approach, however, necessitates the presence of multiple teachers, which means it may not be applicable in scenarios where there is only one sub-optimal teacher.

Even though the field of action advising for reinforcement learning has been studied for more than a decade, rarely has work been done for representing the rationale behind advising. The aforementioned approaches overlook the importance of explainability or transparency when the teacher issues its advice, which is an important part of handling teacher's suboptimality. In our work [54], where the teacher is not limited to providing the advice itself, explanations on why the advice are given are also sent to the student. Although the teacher cannot improve its own fixed policy, which is equivalent to reusing advice in memory [172], the explanation enables the student to reflect on the given advice and transfer to a different environment.

We also introduce the approach allowing the teacher to reflect upon its own advice [23] . The teacher withholds its advice if the expected outcome does not align with the student's experience, and gradually learns to filter out bad advice and keep the good ones. Previous work exists to address sub-optimal teachers [136], however, there is little work done on mechanisms that enable a student to comprehend the rationale behind sub-optimality. We directly address the domain shift between the student's and teacher's tasks, which quickly adapts the teacher's world model composed of prediction networks based on a collection of the student's observed behavior and returned reward. Our teacher is thus not limited to withholding advice that negatively impacts the student, but also can actively improve the quality of issued advice, which is beyond previous approaches.

1.3 Preliminaries

Reinforcement Learning: We consider the problem of transferring knowledge from a teacher to a student in the context of reinforcement learning. In our model, both single-agent and multi-agent scenarios fit our work.

The Markov Decision Process (MDP) [139] of a single-agent reinforcement learning task is a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}\, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{R} : \mathcal{S} \mapsto \mathbb{R}$ is a reward function, $\mathcal{T}(s, a, s') = P(s'|s, a)$ is a transition function, and γ is a discount factor. A stochastic policy is denoted by $\pi(a|s) : \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, indicating the probability that an agent chooses action a given the state s. To calculate the state-value and action-value functions, definitions are given as follows:

$$V_\pi(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t \right],$$
$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t, a_t \right],$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value given that an agent follows its policy π. The goal is to learn a policy which maximizes the discounted cumulative reward, i.e. to learn π such that the expected discounted return V_π or Q_π is maximized.

The Multi-agent Markov Decision Process (MDP) of a multi-agent reinforcement learning task is a tuple of $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$, where $\mathcal{N} = \{0...n\}$ is a set of agents, $\mathcal{S}$ is a set of states, $\mathcal{A} = \Pi_i \mathcal{A}_i$ is a set of joint actions, $\mathcal{R} : \mathcal{S} \mapsto \mathbb{R}$ is a reward function, $\mathcal{T}(s, a_0, a_1...a_n, s') = P(s'|s, a_0, a_1...a_n)$ is a transition function, and γ is a discount factor. There is a set of policies $\pi_i(s, a_i) = P(a_i|s)$ for each agent i, where the single-agent setting is a special case. The goal is similarly also to learn a policy which maximizes the discounted cumulative reward. Same as the single-player case, value-based methods similar to Q-learning maintain a Q-table or a Q-value network, and agents choose actions to maximize Q values, such as VDN [138] and QMIX [120]. Actor-critic methods such as COMA [41] have one policy network for each agent, which takes in an agent's observation and infers an action, while another critic network estimates the quality of the policy.

Action Advising We investigate action advising in the context of reinforcement learning. Action advising hinges on the teacher-student paradigm, where a teacher agent aims to advise the student agent by offering guidance in the form of actions based on the student's observations during the training process. The teacher agent has access to a policy pre-trained in a source task, π^T, and provides recommended actions to the student agent, who attempts to learn a policy π^S in a target task. Formally, let s_t represent the student's perceived state of the environment at time step t, and a_t denote the action undertaken by the student. Thus, following the formulation in a prior work [23], we have:

$$a_t = a_t^T \sim \pi^T(a_t|s_t) \text{ if } H(s_t) = 1, \text{ or } a_t^S \sim \pi^S(a_t|s_t) \text{ if } H(s_t) = 0, \tag{1.1}$$

where $H : \mathcal{S} \mapsto \{0, 1\}$ serves as a binary indicator function, which is assigned 1 when the teacher provides advice, and 0 when the teacher does not.

1.4 Overview of Chapters

The arrangement of the rest of the chapters is as follows:

Chapter 2 We present our work [54] to demonstrate how we use a decision tree policy as the distilled policy of a teacher, and how its decision tree paths are served as explanations given to the student, in addition to the action advice. Experiments show that the student performance is accelerated and improved as student assimilates knowledge from teacher into its internal model.

Chapter 3 Based on the explanation work, we let the student determine if the advice from the teacher should be taken or rejected in an environment that the teacher has never seen before [54]. The student also tests its transferability to an unseen environment without the teacher - only depending on its learnt internal knowledge.

Chapter 4 Instead of letting the student decide if the advice is good enough, we let the teacher introspect its own advice [23]. If the teacher's estimated outcome is similar to what the real situation is, then the teacher considers the advice beneficial.

Chapter 5 We take one further step where the teacher may improve its own policy to provide alternative advice. The teacher collects data from its own environment and the student's environment to train prediction networks on the student rewards, for the teacher to train in its imagination to adapt to the student's need.

Chapter 6 We examine if the artificial agent's advice is acceptable to human students, and if explanations improve the acceptance.

Chapter 7 The advising methodology extended to advising humans, utilizing the quickly-developing large language models.

Chapter 8 We explore transfer learning in urban search and rescue, showcasing how a model trained in one environment can generalize to aid decision-making in a new, complex setting. This chapter demonstrates transfer learning's impact on improving response strategies and operational efficiency in critical rescue operations.

Chapter 9 Application of transfer learning in self-driving technology, focusing on an innovative approach to optimize parameter selection across different driving contexts. The chapter highlights a reinforcement learning-based method to enhance model adaptability and efficiency in international automotive markets.

Chapter 10 We conclude our existing contributions and plan for the future.

While Chapter 2 focuses on providing explanations, and paves the way for Chapter 3 that utilizes explanations and internal model to reject bad advice, Chapters 3 - 5 considers handling the sub-optimal advice. Chapter 3 handles from the student's side, however, as teacher is a more experienced agent, Chapter 4 - 5 handle sub-optimally from the teacher side and are more effective. Chapter 6 -7 are the human experiments, where we evaluate our methods with human feedback. Chapter 8 - 9 are previous related works of transfer learning, that we would like to show how we motivate our directions and are inspired.

Chapter 2

Providing Explanations and Building Student's Internal Model

In this chapter, we introduce our work on how Explainable Action Advising (EAA) could improve the transparency of teacher-student action advising [54]. In our proposed method, we assume we have access to a teacher's "black box" policy, i.e., neural network, from which we can sample demonstration trajectories. These trajectories are used to distill the policy into a decision tree in which each leaf represents an action, and the corresponding path from root to leaf represents the explanation for why an action should be taken. When the teacher issues advice, both the decision path (explanation) as well as the leaf (action) are given to the student, who incrementally re-assembles the decision paths together in an attempt to reconstruct the teacher's decision tree in its internal model. This internal model serves two purposes: 1) it acts as a memory and allows a student to apply prior advice to same or similar future states, and 2) it allows a student to identify unhelpful advice and selectively ignore it, which is useful for novel environments. This is analogous to the self-explanations [26] employed by human students in order to improve learning effectiveness. Importantly, EAA is agnostic to the student's underlying reinforcement learning algorithm – intuitively the teacher is guiding policy exploration.

EAA makes three main contributions: a) introduce an RL algorithm-agnostic action advising method which uses explanations to improve advice generalization; b) empirically show that our approach improves policy returns and sample efficiency in single-agent and multi-agent environments, when compared to multiple state-of-the-art baselines; c) utilize explanations to make use of sub-optimal advice in transfer learning. Specifically, we discuss (a) and (b), which focus on how the explanations are generated and how this approach helps the performance, in this chapter, and (c), which considers transferring to an unseen scenario, in the next chapter.

2.1 Background

Decision Trees and Policy Distillation: A precondition of our method is the distillation of a black-box teacher agent policy into an interpretable representation, i.e., a decision tree which is commonly employed as an interpretable proxy model [46]. Given a pretrained neural network policy, we extract it into an equivalent decision tree model utilizing the distillation method from the *Verifiability via Iterative Policy ExtRaction* (VIPER) [15] framework. VIPER extracts a suitable decision tree policy $\hat{\pi}^*$ from a policy network π^* through an iterative process of sampling trajectories and training a decision tree with standard algorithms, in a DAGGER-based [125] setup. Over the course of N iterations, VIPER samples trajectories of state-action pairs $(s, \pi^*(s))$ into a running data set D_t. This data set is re-sampled according to $(s, a) \sim P((s, a)) \propto \hat{l}(s)\mathbb{1}[(s, a) \in D]$ where $\mathbb{1}()$ is the indicator function and the loss is computed by:

$$\hat{l}_t(s) = V_t^{(\pi^*)}(s) - min_{a \in A} Q_t^{(\pi^*)}(s, a)$$

After re-sampling to get the current data set D_t, a single decision tree policy is trained. After N iterations, the best tree policy $\hat{\pi}^*$ out of all candidate policies is selected.

2.2 Methodology

We introduce Explainable Action Advising (see Fig. 2.1), in which a teacher agent provides action advice as well as an explanation to a student agent. The student may query its internal model if there is an advice, or ask the advice from the teacher. If no advice is provided, the student may take its own advice over its training process.

2.2.1 Explainable Action Advising

An outline of the EAA algorithm is shown in Algorithm 1. The distilled teacher policies $\hat{\pi}_1^*, \hat{\pi}_2^*...\hat{\pi}_n^*$ are provided as input to EAA – one for each student agent – along with a heuristic function $h(\cdot)$ which determines when advice should be issued to an agent. We define two hyperparameters: an advising budget b and a decay parameter γ. The budget b is used to limit the amount of advice that is issued during the student's training, which is useful for limiting the communication between teacher and student, and motivated by preventing cognitive overload in potential human students [144]. The decay rate γ determines the likelihood with which an agent will re-use advice that has been stored in its internal model $\hat{\pi}_i$, giving the student freedom to execute its own policy after it has sufficiently learned from the teacher's advice and potentially exceed the teacher's own performance.

Our algorithm operates over multiple iterations, during which we roll out a policy and sample T timesteps from the environment. For each time step t, one of three things happens: a) the student will query its internal model for an action, b) the teacher will issue advice and an explanation which the student will incorporate into its internal model and then follow, or c) the student will sample an action from its own current policy. In the first case on Line 6, if the student has previously received and stored advice applicable to its current state, it follows the corresponding action with a probability determined by the decay rate γ. Intuitively, the agent should become less reliant on the teacher's prior advice over time and increasingly follow its own actions.

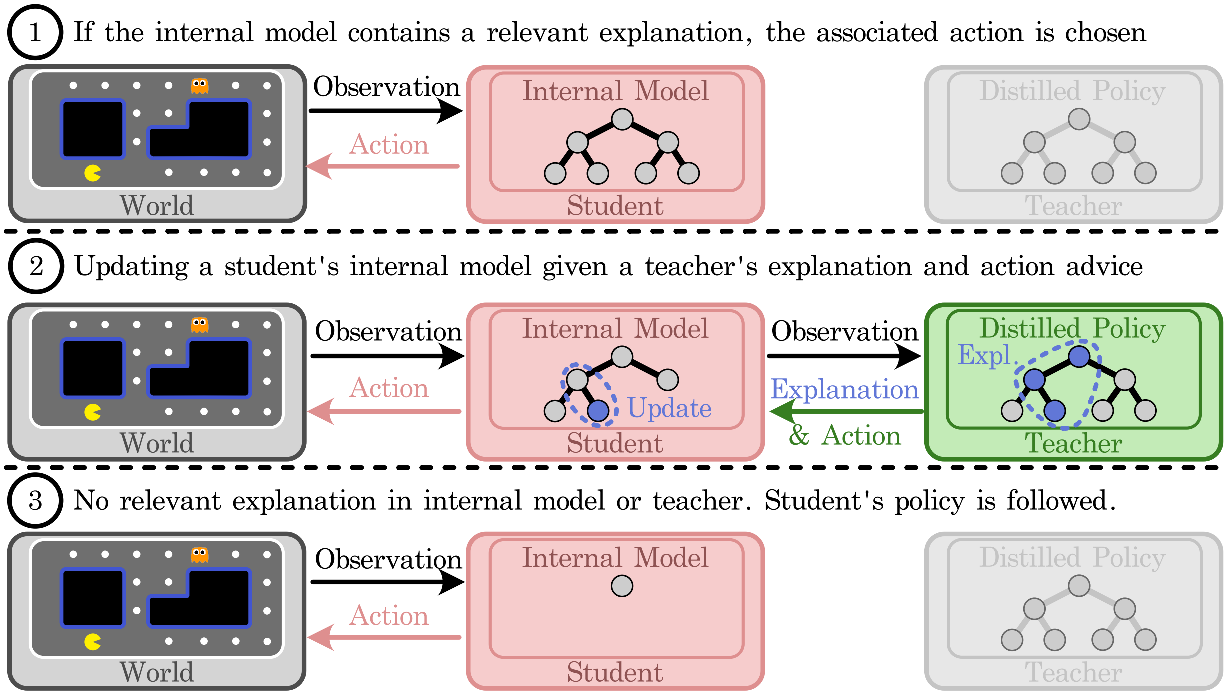

① If the internal model contains a relevant explanation, the associated action is chosen
Observation
Action
World
Internal Model
Student
Distilled Policy
Teacher
② Updating a student's internal model given a teacher's explanation and action advice
Observation
Action
World
Internal Model
Update
Student
Observation
Explanation
& Action
Distilled Policy
Expl.
Teacher
③ No relevant explanation in internal model or teacher. Student's policy is followed.
Observation
Action
World
Internal Model
Student
Distilled Policy
Teacher

Figure 2.1: Overview of EAA. As the student makes observations during training, one of three cases may occur.
1) The student has previously received advice for a similar state and is able to identify it through an explanation in its internal model, then follow the associated action.
2) The student has not previously received advice, but is now given advice by the teacher and stores the explanation in its internal model, then follows the advised action.
3) The student has not previously received advice and the teacher is not currently providing advice; the student follows its own policy. In cases where the teacher is trained in a different environment than the student, the student may choose to reject unhelpful advice.

Algorithm 1 Explainable Action Advising

Input: Distilled teacher policies $\hat{\pi}_1^*, ...\hat{\pi}_n^*$ for n-agent mission, heuristic function $h(\cdot)$
Parameter: Advice budget b, advice decay rate γ

```
 1: Internal model $\hat{\pi}_i = \emptyset$, student $\pi_i = \emptyset \; \forall i = 1, \ldots, n$
 2: for iter $j = 1, 2, \ldots$ do
 3:     for time step $t = 1, \ldots, T$ do
 4:         for agent $i = 1, \ldots, n$ do
 5:             if $\gamma^j < \alpha \sim \mathcal{U}[0, 1]$ and $s_i^t \in \hat{\pi}_i$ then
 6:                 take action $a_i^t = \hat{\pi}_i(s_i^t)$
 7:             else if $b > 0$ and $h(s_i^t)$ then
 8:                 give advice $\hat{a}_i^t$ and expl. $\hat{p}_i^t = \texttt{path}(\hat{\pi}_i^*, s_i^t)$
 9:                 update internal model $\hat{\pi}_i = \texttt{store}(\hat{\pi}_i, \hat{p}_i^t)$
10:                 take action $\hat{a}_i^t$
11:             else
12:                 take action $a_i^t = \pi_i(s_i^t)$
13:             end if
14:         end for
15:     end for
16:     Update $\pi_i$ via policy optimization for $i = 1, \ldots, n$
17: end for
```

In the second case on Line 8, if the student has not previously received relevant advice the teacher agent determines whether advice should be issued to its student, which is controlled by the heuristic function $h(\cdot)$, the remaining budget b, and the uncertainty associated with the advised action such that only sufficiently confident predictions are used as advice. If this occurs, the teacher samples a leaf node action $\hat{a}_i^t$ and its decision path explanation $\hat{p}_i^t$ from its distilled decision tree and sends them to the student. The student takes this explanation, incorporates it into its own internal model $\hat{\pi}_i$, and follows the advised action.

Lastly, in the third case on Line 12 the student simply follows its own current policy π_i, which is itself updated on each iteration with the data collected from the rollouts. Additionally, there is an optional check in which a teacher agent may only issue advice if: a) the uncertainty of executing the action is sufficiently small, and b) the advice from its internal model matches the action from its expert policy – useful if the distilled tree is overly noisy. If the action uncertainty is too high, the teacher may choose to issue only action advice and withold the explanation from the student. This allows the student to leverage the teacher's learned action probabilities without committing them to memory.

The algorithm is explainable as the decision tree policy, who represents the original agent's policy, is explainable. The features on the decision tree paths indicate what features lead to the output action as the leaf node. This ensures the decisions generated from black-box neural networks could be explained.

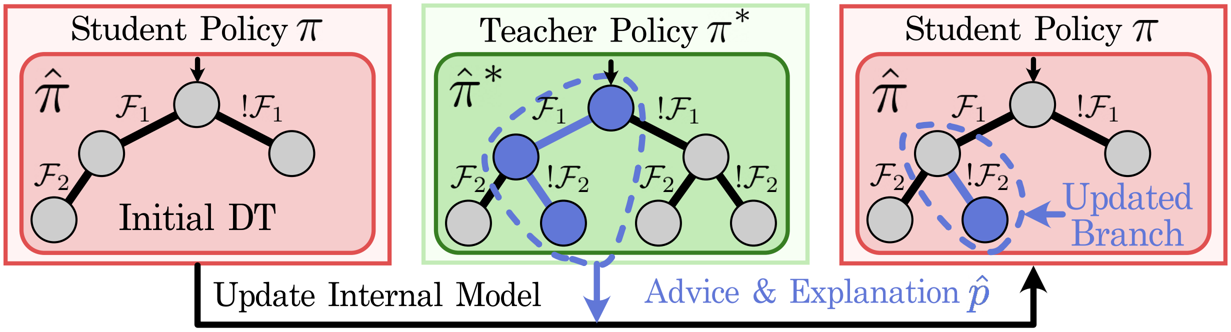
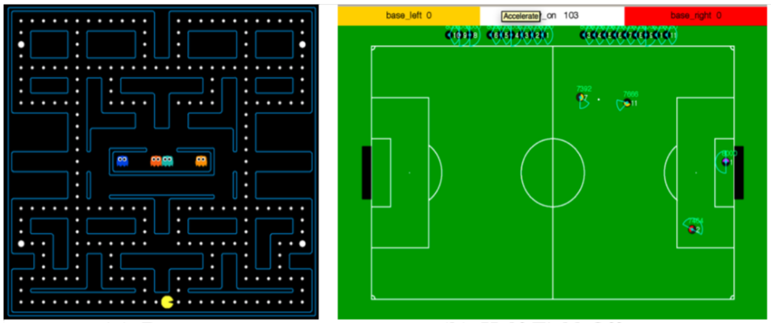


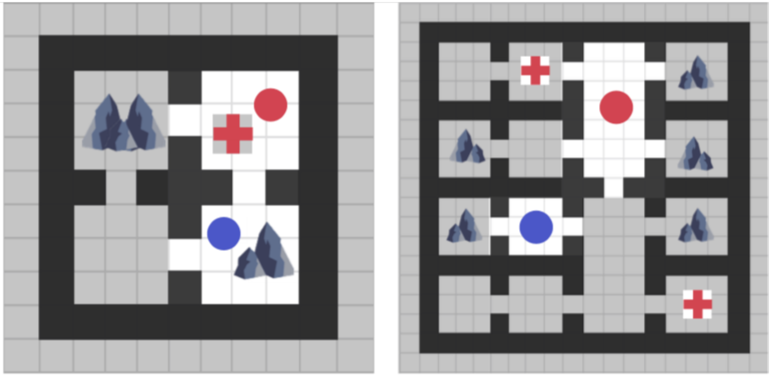
Figure 2.2: The decision tree reconstruction process that is performed when updating a student's internal model $\hat{\pi}$. The student receives an explanation decision path $\mathcal{F}_1 \wedge ! \mathcal{F}_2$ from the teacher's distilled policy $\hat{\pi}^*$, finds the last common node $\mathcal{F}_1$ currently in its model, and adds all subsequent missing nodes – in this case only $!\mathcal{F}_2$.

2.2.2 Decision Tree Reconstruction

The reconstruction process performed in the `store` function, shown in Fig. 2.2, consists of merging the current decision tree $\hat{\pi}_i$ and the decision path explanation, $\hat{p}_i^t$ which reduces to merging two trees. Following standard decision tree architecture, we assume that the tree consists of a set of K nodes $u_0, \ldots, u_K$, where each node contains information regarding its child nodes and branching condition. By transferring the teacher's tree in a piece-wise fashion, we transmit only useful advice which aids in the generalization process – uncertain actions are ignored – while minimizing the amount of information transmitted. The size of the student's tree is conditional on the budget and complexity of the policy.



(a) Pacman (b) Half Field Offense



(c) 4-Room Env. (d) 14-Room Env.

Figure 2.3: Environment Screenshots. The 4 and 14 room environment have a medic (red circle), engineer (blue circle) and victims (red plus).

2.3 Experiment

We empirically evaluate EAA in three simulated environments: Pacman, Half Field Offense (HFO), and Urban Search and Rescue (USAR). In each of the environments we first train a teacher policy to convergence and use this policy to subsequently advise a student policy using EAA as well as several baseline algorithms. To evaluate EAA's transfer learning ability, we perform an additional experiment in the USAR environment where the teacher is trained in a variant where a feature is missing, and then advise a student in the environment with that feature.

2.3.1 Experimental Setup

Figure 2.3 depicts our three environments (Pacman, HFO, and USAR), with the USAR task implemented in a simpler 4-room and more difficult 14-room setup.

Pacman: a single-agent scenario with a discrete state space [73]. We use the standard single-agent environment [73] with the *OriginalClassic* layout (Fig. 2.3 (a)) and 4 aggressive ghosts. An additional reward penalty of -1 is received when an invalid action is predicted, in place of action masking. The student and teacher policies were trained via PPO [130] for $10,000$ episodes with an advice budget of $b = 10$. The teacher and distilled teacher achieved average rewards of 450.

HFO: a multi-agent scenario with a continuous state space where two homogeneous RoboCup agents play as a team [61, 77]. We use the standard 2v2 environment [77] (Fig. 2.3 (b)) and train teacher and student policies for the offense agents with SARSA [139] as in prior work [31] for $2,000$ episodes with an advice budget of 300. The teacher and distilled teacher achieved an average reward of 0.8.

USAR: a multi-agent scenario with a discrete state space and heterogeneous agents with different action spaces and hard coordination constraints [42, 53, 86]. We have two layouts: a simple 4-room layout (Fig. 2.3 (c)) and a more complex 14-room layout (Fig. 2.3 (d)). The environment consists of rubble which can be cleared and victims which must be healed, and a team of two agents: a medic who heals unblocked victims and an engineer who can clear rubble from blocked victims. The reward is team-level and gives $+10$ for each victim healed. Both layouts contain rubble in every room, while the 4-room layout contains 1 victim randomly distributed and the 14-room layout contains 2 victims. The teacher and student policies are trained using COMA [41] for $16,000$ episodes with an advice budget of $100,000$. The teacher and distilled teacher achieved an average reward of 10 in both layouts, indicating a sub-optimal teacher for the 14-room layout. This task necessitates cooperation between a medic and an engineer robot to save trapped victims – the engineer must remove rubble before the medic comes across and heals the victim, and is the most difficult environment.

2.3.2 Benchmarks

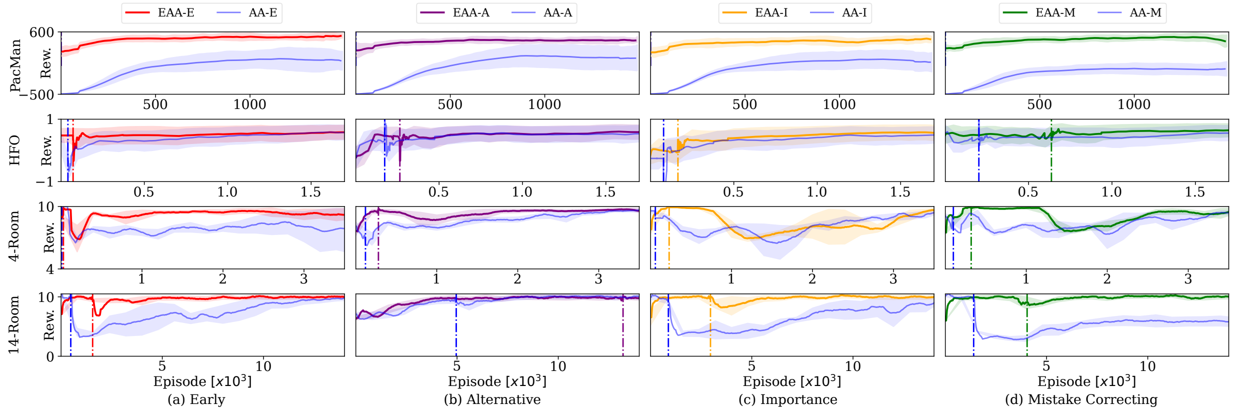
In all experiments, we define four variants of EAA utilizing the heuristic functions established in prior work [144] as $h(\cdot)$: EAA with early advising (EAA-E), alternative advising (EAA-A), importance advising (EAA-I), and mistake correcting advising (EAA-M). EAA-Median indicates the median performance of all EAA variants. The heuristics are the same as in the Action Advising baselines and are defined below.

Action Advising: We compare EAA to four heuristic-based Action Advising algorithms [144] in all environments:

1. *(AA-E) Early* : Continuously advise at each state.
2. *(AA-A) Alternative* : Advise with a fixed frequency.
3. *(AA-I) Importance* : Advise when the state importance $I(s) >$ threshold.
4. *(AA-M) Mistake Correcting* : Advise if $I(s) >$ threshold and $a^{student} \neq a^*$.

Memory-based Action Advising: For the discrete state environments, we compare EAA to three recently introduced [172] benchmark algorithms: Q-change Per Step, Decay Reusing Probability, and Reusing Budget. These works are conceptually similar to ours in that students have a memory and reuse teacher advice in the form of state-action pairs, so they serve as a strong state-of-the-art comparison.

Action Advising with Adaptive Teacher: In the USAR environment, we further compare EAA against an approach in which reinforcement learning is used to learn an optimal teacher [110, 177], and denote this as *Learn to Teach*. This algorithm learns over multiple sessions, within each of which the teacher learns to allocate its advice budget with the constraint that the student's policy must converge. All methods except for Learn to Teach are trained for 5 random seeds. Due to the computational demand of the inner reinforcement learning loop, Learn to Teach (100x slower) is only evaluated once for USAR.



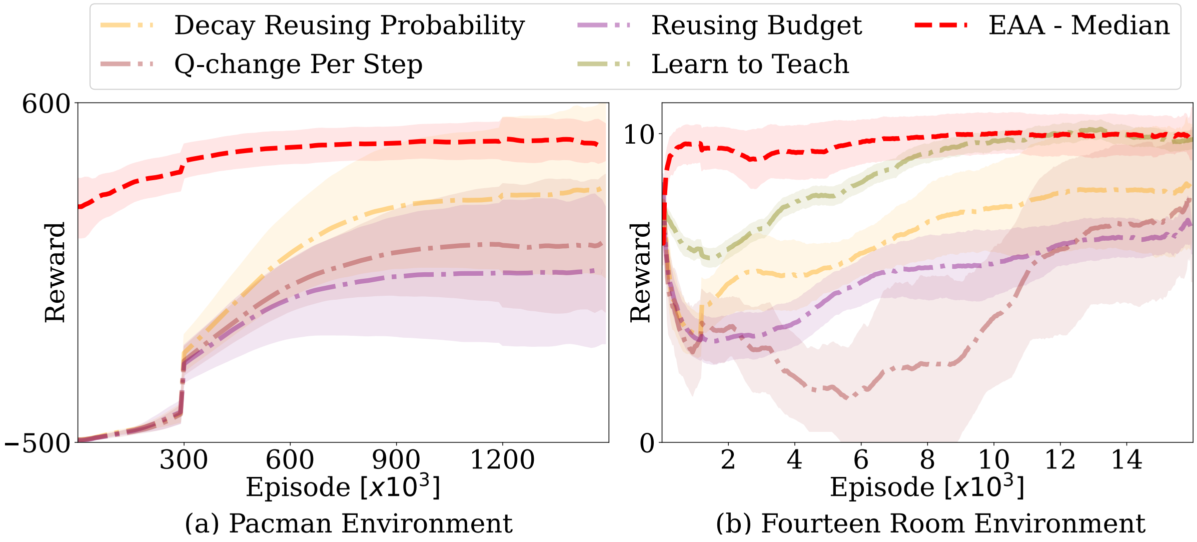
(a) Early (b) Alternative (c) Importance (d) Mistake Correcting

Figure 2.4: Comparing Explainable Action Advising (Early: Red, Alternative: Purple, Importance: Yellow, Mistake Correcting: Green) with Action Advising (Blue) on the PacMan, HFO, and USAR (four and fourteen room) environments.
Dash-Dotted lines indicate when advising budget is exhausted.
Results are averaged over 5 independent trials except for 14-room over 3.

2.3.3 Results

Figures 2.4 and 2.5 show the results of our EAA variants to the action advising baselines described above, and are intended to demonstrate improved policy return and sample efficiency owing to advise re-use from explanations.

Comparison to Action Advising: In Fig. 2.4, we compare our four EAA variants described in Sec. 2.3.2 to AA [144] in all environments. We observe that EAA converges to the teacher's reward significantly faster than the AA baselines and with lower reward variance, indicating more training stability. Of note is PacMan, where the lack of initial state randomization enables heavy advice re-use and offsets the small advice budget, allowing EAA to perform remarkably well. In HFO, EAA achieves only a slight improvement over AA; we conjecture that this is due to the simplicity of the task – it was designed for classical methods such as SARSA, which enables AA to perform well. Advise re-use and generalization is evident in the HFO and USAR environments where EAA results in later budget exhaustion (dashed lines, Fig. 2.4) across all heuristics. This is not evident in PacMan simply due to the extremely small advice budget; it is spent quickly at the start of training. All algorithms except for EAA-A (Fig. 2.4 (b)) also suffer from a large drop in reward when the teacher's budget is exhausted. While this is expected, it is interesting to observe that our EAA variants tend to recover quicker from this drop than AA.



(a) Pacman Environment (b) Fourteen Room Environment

Figure 2.5: Baseline of Pacman (left) and 14-room USAR comparing EAA to SOTA algorithms. Results are averaged over 3 random seeds except for Learn to Teach which uses 1 due to run-time.

Comparison to Memory-based Action Advising: In Fig. 2.5, we see that EAA again demonstrates both improved convergence rate and maximal policy return when compared to the memory-based action advising baselines, achieving optimal performance in both the Pacman and USAR 14 room environments. The memory-based method simply maps from observations to actions by storing teacher's previous advice, so the advice from student model always matches teacher's. Q-change Per Step displays the worst performance because it assumes a strict increase in Q-values when the student receives advice, otherwise discarding it. However, this assumption is often violated, especially in USAR where an action such as removing rubble does not directly result in an increase in Q-values – only healing victims does. Decay Reusing Probability and Reusing Budget are able to achieve optimal performance, yet take significantly longer to converge than EAA.

Comparison to Action Advising with Adaptive Teacher: Figure 2.5 (b) shows the comparison the adaptive teacher action advising method, Learn to Teach, in the USAR 14-room layout. While Learn to Teach converges to the teacher's reward, it takes considerably longer to do so, requiring nearly $12,000$ episodes compared to EAA's $6,000$.

2.4 Conclusion

We propose Explainable Action Advising (EAA), a novel action advising algorithm in which a teacher agent issues explanations in addition to action advice to student agents. Our results show that EAA improves both policy reward as well as convergence speed with agents trained via MARL when compared to eight state-of-the-art benchmark algorithms. We demonstrate that our method is especially effective in budget-constrained advising, as each piece of advice now conveys more information and helps reduce communication frequency with respect to advice in the case of artificial agents. EAA also enables explainable transfer to unseen scenarios, which will be discussed in the next chapter.

Chapter 3

Student Rejecting Bad Advice Based on Explanations and its Internal Model

We present how Explainable Action Advising (EAA) leads to explainable generalization in this chapter, as part of our work in [54]. Recall that EAA improves the transparency of teacher-student action advising by utilizing the decision tree paths of the distilled decision tree policy, from the teacher's "black box" policy where we sample demonstration trajectories with VIPER. Through collecting those paths where the nodes are features forming explanations leading to the leaves as the resulting action advice, students may have its internal model to reconstruct the teacher's decision tree policy.

If the student is faced with an unseen scenario, 1) It may tell if the teacher's advice is generalizable in this new scenario by examining the features in the explanation, which allows the student to identify the unhelpful advice and selectively ignore them to avoid being misled; 2) The student may query its internal model to seek for a solution when faced with any observation, and only explore the environment when its own knowledge is not applicable. In short, every time the student receives advice, it examines the explanations and internal model, to decide if the teacher's advice should be accepted or rejected. Therefore, our method helps improve the generalizability with the student taking the role of rejecting unsuitable advice and reflecting on its internal knowledge. We performed experiments to demonstrate its improvement empirically, and compared with common transfer learning approaches.

3.1 Background

A fundamental property of decision trees is their ability to perform feature selection [119, 170]. Decision trees operate by partitioning the data along certain feature dimensions in a way that best separates the target classes. Notably, only a subset of features, known as decision features, are employed in this process, while others are ignored. This property lends decision trees their interpretability, as the decision path from the root to a leaf node provides a clear explanation of the classification decision based on a subset of features. Leveraging this characteristic, EAA enhances the transparency and adaptability of action advising in reinforcement learning.

The importance of decision tree features is intrinsically determined by their position and frequency of appearance within the tree's paths. A feature that frequently appears close to the root of the tree is generally more important for classification, as it contributes to the majority of decision paths. On the opposite, a feature sits far away from the root or not shown in the tree is less important. Let's denote the set of all features as $\mathcal{F}$ and the set of decision features, which appear in the decision paths of the tree, as $\mathcal{F}_D$, where $\mathcal{F}_D \subseteq \mathcal{F}$. These decision features $\mathcal{F}_D$ form the basis of classification decisions. Consequently, understanding these decision paths allows for an effective and interpretable transfer of knowledge, which is a crucial aspect of our proposed EAA approach.

3.2 Methodology

Using a decision tree representation for the internal model allows the student to generalize the advice to novel states which differ from that in which it was issued. We do this by treating the decision tree as a classification model, in which the state s is the input and the action a is the target, and extracting a generalizable classification tree during distillation. Formally, let the set of all possible observed features in the teacher's environment, i.e. a source environment, be $\mathcal{F}_S$ and the set of features modeled by our reconstructed decision tree be $\mathcal{F}_R$ where $\mathcal{F}_R \subseteq \mathcal{F}_S$. Then a given decision path corresponding to a feature vector $f \in \mathcal{F}_S$ will also generalize to any other feature vector $f' \in \mathcal{F}_S$ as long as the features in which they differ are not contained in $\mathcal{F}_R$. Such generalization leads to improved advice reuse, and as a result improved learning performance as is empirically shown in Sec. 3.3.2.

Similarly, a decision tree representation also lends itself to transfer learning, by allowing a student to recognize situations in which advice is *not* applicable and as such should not be followed, e.g. when the teacher is trained in a different environment. Let us denote the set of features in the teacher's source environment as $\mathcal{F}_S$ and the set of features in the student's target environment as $\mathcal{F}_T$. We impose an additional constraint on the student such that it *rejects* advice if the decision path explanation contains features in $\mathcal{F}_T - \mathcal{F}_S$, which means the advice is applicable in the source environment but not in the target, and the student instead follows its own policy to explore to deal with this unseen scenario. Intuitively, we only want to follow advice if it is dependent on features found in both environments. We refer to this process as *learning through reflection*, and empirically demonstrate that this yields significantly improved policy transfer in Sec. 3.3.2.

The overall generalization procedure is shown in Algorithm 2. Ideally, the student will need to seek for exploration when the advice is not generalizable. This happens when there is one decision feature given from the teacher or its internal model, where the feature is never observable by the teacher, and definitely not observable by the student's own internal model as it is a subset of teacher's knowledge. However, if the student will need this feature to make a decision, then clearly the advice from the teacher is not applicable. If such an feature exists when student is faced with, the student will reject the advice and explores.

Algorithm 2 EAA - Generalization

```
Require: $\mathcal{F}_S$, $\mathcal{F}_R$, $\mathcal{F}_T$
Initialize EAA normal process.
while not End of Training do
    $s \leftarrow$ current state
    $a^* \leftarrow$ action advice from teacher
    $f \leftarrow$ extract every decision feature from advice $s$
    if $\exists f \notin \mathcal{F}_R$ and $f \in \mathcal{F}_T - \mathcal{F}_S$ then
        Follow student's own policy
    else
        Follow advice $a^*$
    end if
    Update the student internal model
end while
```

3.3 Experiment

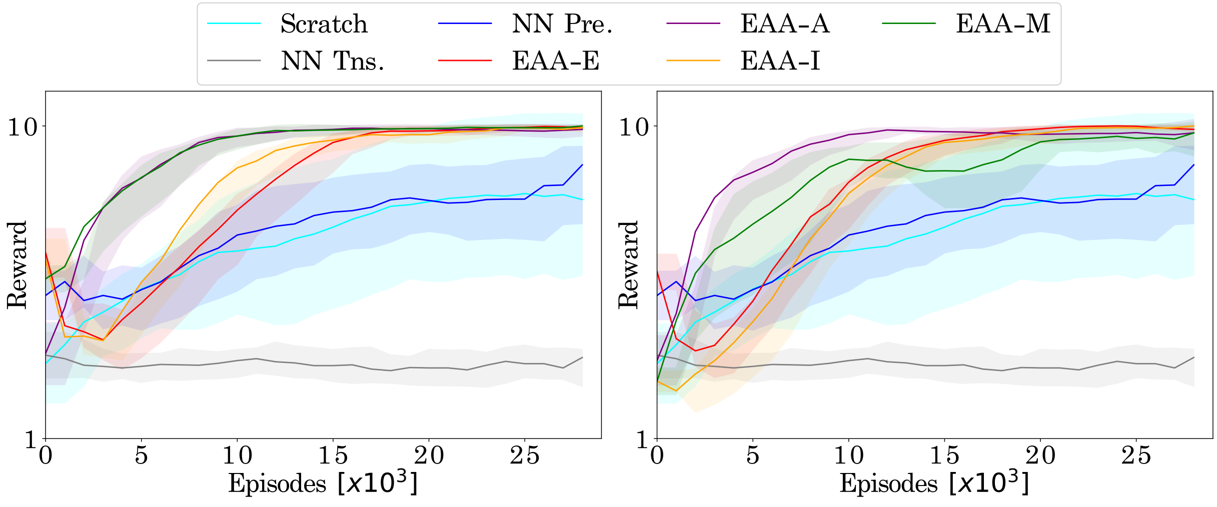
3.3.1 Experiment Setup

We use the same environment of USAR in the last chapter. Here we let the teacher train in a USAR variant where there is no rubble but only victim(s) to be healed. However, the student is faced with a USAR variant where the team must remove the rubble first before healing any victim. Rubble existence is an important feature where the student would reject any advice if in the teacher's explanation the student does not see this critical feature.

More specifically, the teacher is a trained expert in a USAR 4-room source environment *without* rubble, and is used to advise a student in a target environment *with* rubble, in which it is a non-expert due to the additional coordination requirement of removing it. The student rejects advice if the teacher's advised explanation contains no features related to rubble when encountering a room with rubble. The team is rewarded only when the medic has healed a victim, which may be hidden behind rubble and thus requires the engineer to remove. Rubble is scattered randomly in various rooms without replacement. The action set is meta-actions with respect to navigation, healing victims, and removing rubble (for the entire room). In the transfer learning experiments presented in the manuscript, the teacher is trained with the same room layout but without rubble (0 rubble, 1 victim), and is used to teach students who learn in the standard layout with rubble (3 rubble, 1 victim).

In the experiments, we first compare EAA against the standard transfer learning technique in which a policy is pre-trained in a source environment, and then fine-tuned in a target environment, denoted as NN Pre. We also include results for the pre-trained policy without fine-tuning (NN Tns.) and a policy baseline trained from scratch (Scratch).

Experiment plots show the results when self-reflection is used to selectively transfer advice from a teacher trained in a different environment than the student.



(a) Teacher with DT & NN (b) Teacher with DT

Figure 3.1: Transfer learning performance for students training in the USAR environment with rubble by a teacher who was originally trained without rubble. The teacher only has access to the extracted decision tree where the teacher NN is not present in (b), compared to (a) where the teacher NN is present. Results are averaged over 5 independent trials.

3.3.2 Comparison to Fine-tuning

Figure 3.1 shows EAA's performance when transferring advice from a teacher's source environment to a student's target environment, as compared to standard transfer learning approaches.

In Fig. 3.1 (a), EAA has access to both the distilled and original policy and does not store explanations in which the action has a low probability, as discussed in Sec. 2.2.1. This is the default behavior used for prior experiments in Sec. 2.3.3, and represents the best-case transfer learning scenario. In Fig. 3.1 (b), the EAA variants are trained by a teacher who only has access to its distilled decision tree and not its original policy. This represents a more difficult task, and establishes a baseline for the later case where there is no teacher and the student relies only on its internal model trained in the source environment (Fig. 3.2).

In Figure 3.1 all four variants of EAA significantly outperform the neural network baselines and converge to the optimal reward within $10,000$ episodes. The transferred teacher policy (NN Tns.) serves as a baseline of the non-expert teacher behavior in the target environment without any additional training; this is the upper limit on the expected sub-optimal advice performance. Notably neither training from scratch (Scratch) nor pre-training on the source environment followed by fine-tuning (NN pre.) consistently converges to the optimal reward. Given that EAA performs nearly as well in Fig. 3.1 (b) as Fig. 3.1 (a), we conclude that maintaining the original black-box teacher policy is unnecessary.

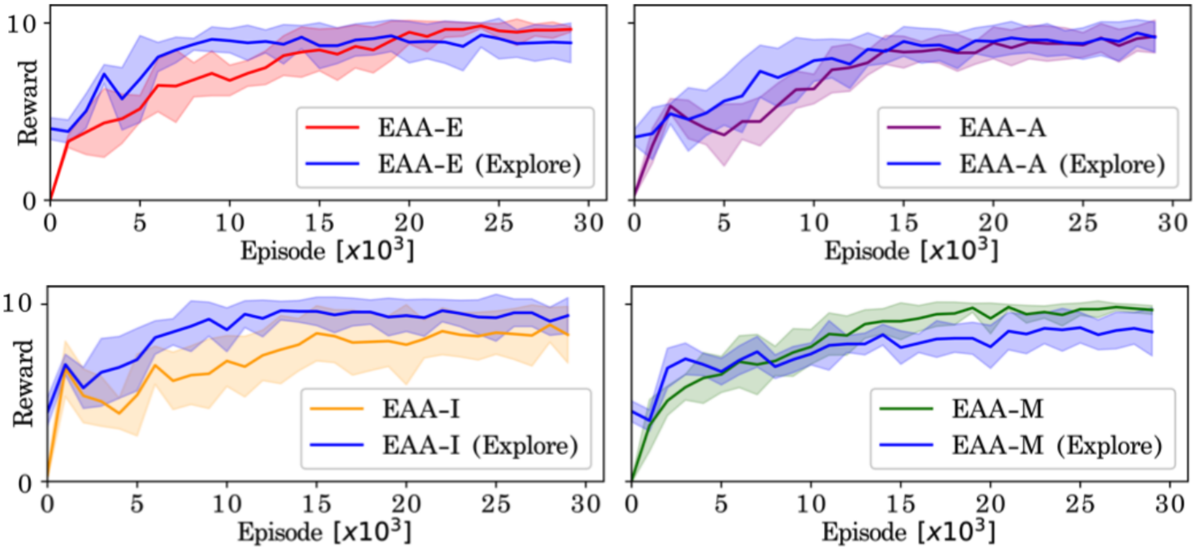
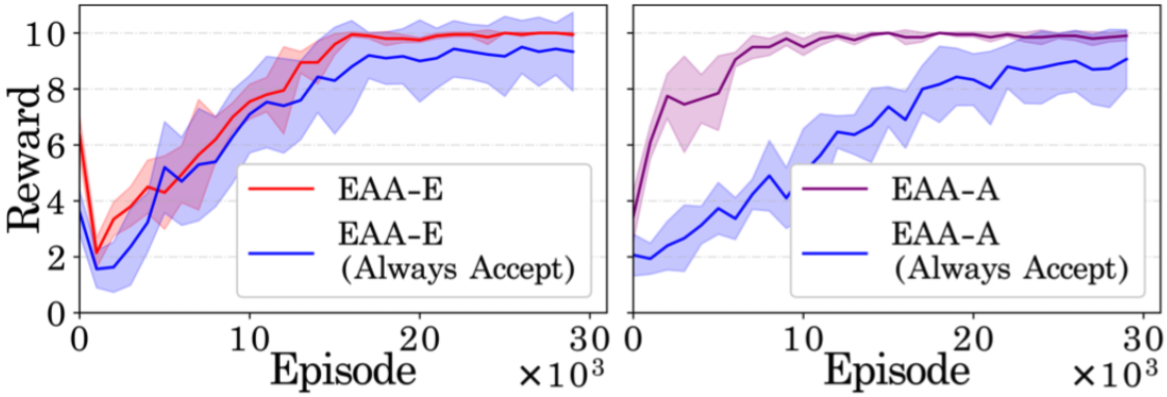


Figure 3.2: A comparison of EAA with reflection, called EAA (Explore), to standard EAA. The student is pretrained in a source environment (4-room no rubble) and transferred to a target environment (4-room with rubble). Results are averaged over 5 independent trials.

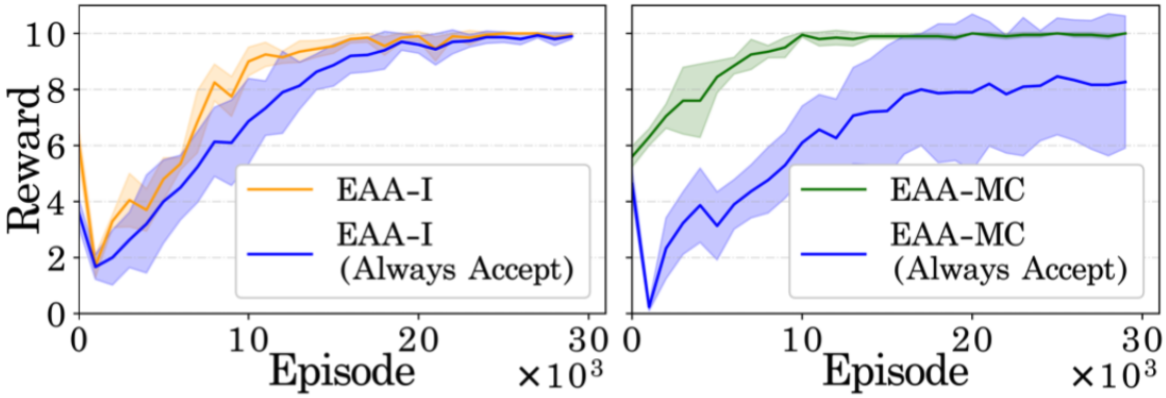
3.3.3 Self-Reflection

Fig. 3.2 shows the results when a student trained via EAA in a source environment continues training in a target environment, when no teacher – and therefore no advising – is present. We implement a version of EAA with reflection in which the student examines its own reconstructed decision tree and ignores actions learned from unhelpful advice and instead executes an exploratory action (EAA Explore).

When compared to an ablated student without reflection, we find that reflection not only leads to improved rewards at the start of training, but also an improved convergence rate for EAA-E, EAA-A, and EAA-I variants. These results show that not only does EAA improve policy transfer to novel environments in the presence of a teacher, but it even yields benefits when the student operates in novel environments *by itself*.



(a) EAA Early (b) EAA Alternative



(c) EAA Importance (d) EAA MC

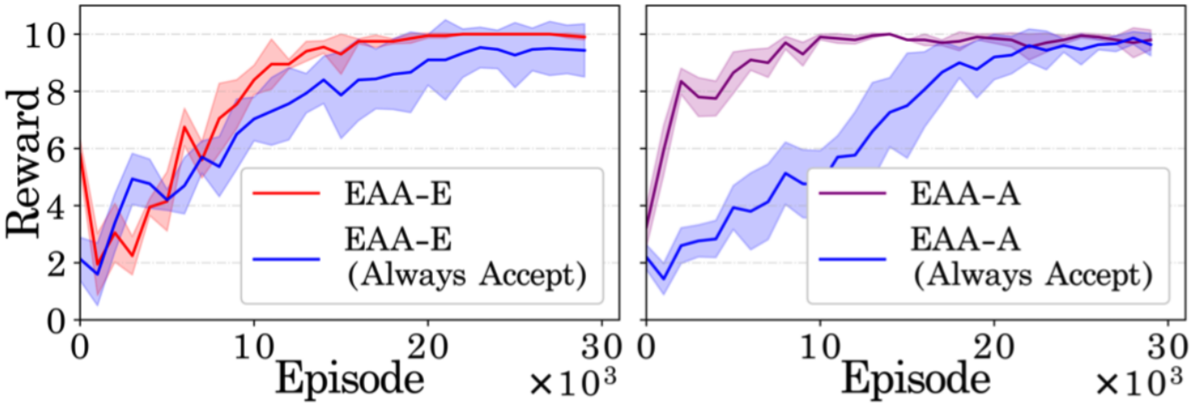
Figure 3.3: A comparison of an ablated version of EAA which always takes the teacher's advice, EAA (Always Accept), to standard EAA when the teacher has access to both its original neural network policy and the extracted decision tree (DT + NN).

3.3.4 Comparison to Ablated Version

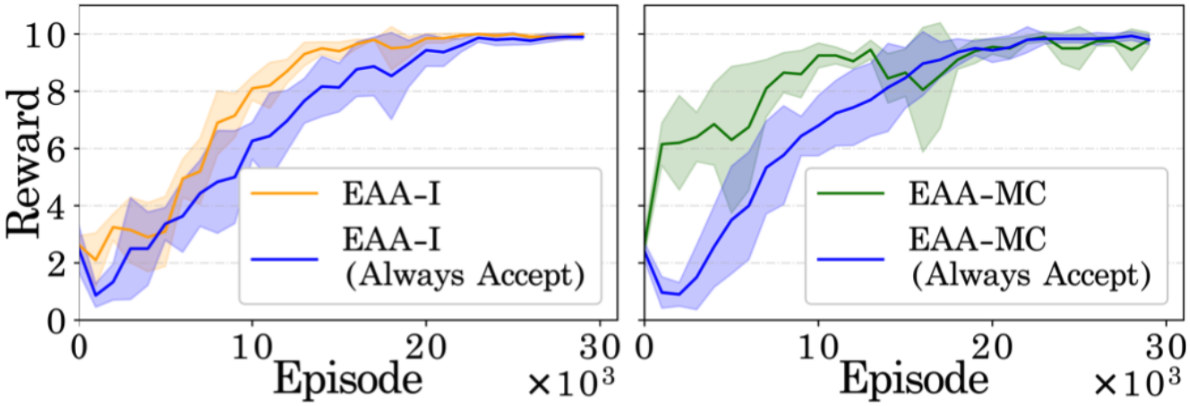
In this section, we include an experiment in which we compare standard EAA to an ablated version which lacks the transfer learning check. The results are shown in Figure 3.3 and Figure 3.4.

In both cases – a DT + NN teacher and a DT-only teacher – always accepting the advice from a non-optimal teacher performs worse than standard EAA. Since the teacher is trained in an environment with no rubble, and thus when the student encounters a room with rubble, it is more advantageous for the student to execute an exploratory action rather than listening to the teacher, which happens in the standard EAA version. In the (Always Accept) versions, this exploratory action is not taken and instead the student always accepts the teacher's (potentially non-optimal) advice, resulting in worse performance.

For EAA-Early, the fact that always accepting advice is similar to standard EAA may be due to advice being given continuously resulting in the budget being used up at a very early stage.



(a) EAA Early (b) EAA Alternative



(c) EAA Importance (d) EAA MC

Figure 3.4: A comparison of an ablated version of EAA which always takes the teacher's advice, EAA (Always Accept), to standard EAA when the teacher only has access to its extracted decision tree (DT).

This is similar to the Importance variant as the threshold for giving advice is rather low. It can be observed that EAA is better at the beginning when it ignores non-optimal advice, but this effect is limited if the advice is given too densely in a limited period of time as in the Early variant. We suspect that this is due to limited updates in the network parameters per iteration. Instead, if the advice is given more sparsely, for example in EAA-Alternative, ignoring non-optimal advice has a more significant effect. Empirically, the later the budget is used up the better the performance, something that is also seen in the Mistake Correcting variant.

In the ablated EAA version which always accepts advice, we notice that in some cases the student fails to converge to the optimal reward. We conjecture that this occurs when the budget lasts a long time – Alternative and Mistake Correcting variants – which prevents the student from learning to converge to optimal on its own. Additionally, the variance in the policy returns is quite a bit larger for the ablated versions – again likely due to taking occasionally poor actions resulting in the policy being more likely to execute non-optimal sequences.

Moreover, removing the original policy NN from the teacher slightly improves the ablated EAA variants (Fig. 3.4). When the teacher's advice is based solely off the extracted DT, the advice is more likely to be erroneous, which ironically means more likely to be correct if the initial advice was incorrect (as in this case). So in a roundabout way, due to the lower quality advice from the teacher, the ablated version which always accepts advice ends up performing better than the ablated version in which the teacher has access to both its DT and NN policies.

3.4 Conclusion

We demonstrate that EAA enables the advice to be transferable to other unseen scenarios, by rejecting teacher's unhelpful advice and learning through reflection. Limitation exists that the important feature from the student must be given in advance. Like our experimental scenario where the rubble existence is an important factor that differs the student and teacher's environment, this key feature must be given to EAA. Otherwise, it is difficult to distinguish if the non-existence of a particular feature is due to the fact that the teacher has no experience of it, or regards it as not important. More robust transfer methods are introduced in the next two chapters.

Chapter 4

Teacher's Introspection for Preventing Bad Advice

This chapter introduces the work on Introspective Action Advising (IAA) [23], in which the teacher determines when its advice is transferable from its source task to the student's target task, in order to only give out beneficial advice. When the teacher's advice originates from a policy trained in a different source task, ensuring its transferability is crucial to avoid misguided exploration by the student, which could result in misleading the student into low-reward state space regions. In IAA, advice is regarded as transferable if the resultant expected discounted returns are considerably similar across tasks. This means the advice is considered suitable if it triggers an outcome in the target task that is sufficiently "close enough" to the outcome observed in the source task.

Algorithm 3 INTROSPECTIVE ACTION ADVISING (Actor-Critic)

```
Require: $\pi^T$, $\pi^S$, $V_{\pi^T}^{\text{Src.}}$, $V_{\pi^T}^{\text{new}}$
for iteration $= 1, 2, \ldots$ do
    Roll out student's policy for $T$ timesteps
    $X \leftarrow \emptyset$
    for $t = 1, 2, \ldots, T$ do
        $h_t \leftarrow$ INTROSPECT$(s_t, V_{\pi^T}^{\text{Src.}}, V_{\pi^T}^{\text{new}}, t)$
        if $h_t = 1$ then
            $a_t \sim \pi^T(a_t|s_t)$
        else
            $a_t \sim \pi^S(a_t|s_t)$
        end if
        $X \leftarrow X \cup (s_t, a_t, s_{t+1}, r_{t+1}, h_t)$
    end for

    Teacher critic update over $X$
    $\rho^T, \rho^S \leftarrow$ CORRECT$(X, \pi^S, \pi^T)$, $\theta_{V^T} \leftarrow \theta_{V^T} + \alpha\rho^T \nabla_{\theta_{V^T}} \mathcal{L}_t^V$

    Student actor-critic update over $X$
    $\theta_{V^S} \leftarrow \theta_{V^S} + \alpha\rho^S \nabla_{\theta_{V^S}} \mathcal{L}_t^V$, $\theta_{\pi^S} \leftarrow \theta_{\pi^S} + \alpha\rho^S \nabla_{\theta_{\pi^S}} \mathcal{L}_t^\pi$
end for
```

4.1 Methodology

Using standard definitions [139] for the state-value and action-value functions,

$$V_\pi(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t \right],$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value given that an agent follows policy π. An agent's goal is to learn π through training such that the expected discounted return V_π is maximized, given an initial state distribution.

Formally, the advice action a_t is transferable for a given state s_t if

$$\left| V_{\pi^T}^{\text{Tar.}}(s_t) - V_{\pi^T}^{\text{Src.}}(s_t) \right| \leq \epsilon, \tag{4.1}$$

Our approach involves accurately estimating $V_{\pi^T}^{\text{Tar.}}$ in the target task. However, we assume advice is only beneficial in the early stages of training when $t < \tau$ and it is unlikely that our estimate of $V_{\pi^T}^{\text{Tar.}}$ has sufficiently converged to the optimal value function in the target task, $V_{\pi^T}^*$:

$$\left| V_{\pi^T}^*(s_t) - V_{\pi^T}^{\text{Tar.}}(s_t) \right| > \epsilon, \tag{4.2}$$

which makes it difficult to determine whether advice is not transferable due to a mismatch in state-values or due to the student not having a sufficiently accurate estimate of the state-value.

4.1.1 Introspection:

We let the teacher itself directly estimate the state-value function in the target task, with the assumption that refining the teacher's existing estimate of the state-values in the source task will lead to quicker convergence to $V^*_{\pi^T}$ than estimating it from scratch. Specifically, we fine-tune a copy of the teacher's state-value function learned in the source task, $V^{\text{new}}_{\pi^T}$, with observed returns from the student's exploration in the target task as estimate of $V^*_{\pi^T}(s_t)$. Therefore, Eq. 4.1 becomes

$$\left|V^{\text{new}}_{\pi^T}(s_t) - V^{\text{Src.}}_{\pi^T}(s_t)\right| \leq \epsilon. \tag{4.3}$$

We introduce a decay hyperparameter, $\lambda \in [0, 1)$, to approximate the temporal cut-off τ. The teacher's advice is issued according to a probability that decreases over time, λ^t, which gives a low probability of advice issuance when $t > \tau$ because while a specific action might lead to similar results in both the source and target tasks, it may not always remain the *optimal* choice. The student thus should be gradually permitted to explore on its own as time progresses. Furthermore, a trade-off exists between how much returns the teacher should observe and incorporate into $V^{\text{new}}\pi^T$, and the necessity to provide advice early in the training for maximal effect [54]. The *burn-in* hyperparameter $\delta \in \mathbb{R}\geq 0$ manages this balance, specifying the number of timesteps the teacher should observe prior to giving advice.

4.1.2 Off-Policy Correction:

IAA fundamentally alters the nature of samples used for training updates as they are no longer solely derived from the student's policy rollouts, π^S. Instead, actions suggested by the teacher's policy intermingle with those produced by the student's policy, violating assumptions for on-policy algorithms. To make an off-policy correction for the student ρ^S, let's denote a trajectory of n samples, obtained through the student's interaction with the environment, as $X = x_0, \ldots, x_n$, where each $x_t = (s_t, a_t, s_{t+1}, r_{t+1})$ represents a single timestep observation. If $\mathcal{L}$ is a loss computed over X, off-policy correction is applied to optimize $\rho^S\mathcal{L}$ with respect to the student's model parameters. The off-policy correction ρ^S is a vector composed of n scalar values.

Each value in ρ^S is an importance sampling ratio $\frac{\pi^S(a_t|s_t)}{\pi^T(a_t|s_t)}$ when a specific action comes from the teacher's policy, or 1 when it is from the student's. Similarly, we also apply an off-policy correction, ρ^T, when optimizing the teacher's state-value function estimate $V^{\text{new}}_{\pi^T}$. This correction ρ^T contains an importance sampling ratio $\frac{\pi^T(a_t|s_t)}{\pi^S(a_t|s_t)}$ when an action comes from the student's policy, and 1 from the teacher's.

4.1.3 Algorithm:

During rollout of the student's policy (Lines 3-12), actions are from either the teacher's or student's policy (Lines 7, 9). The decision whether to issue advice is determined by our introspection function (Alg.4), replacing $H(\cdot)$. An off-policy correction, calculated in Alg. 5, is applied to the teacher's critic loss $\mathcal{L}t^V$ (Line 15), and to the student's actor and critic losses ($\mathcal{L}t^\pi$ and $\mathcal{L}t^V$) in (Lines 17, 18). These losses are then optimized in relation to the model parameters of the teacher's critic ($V\pi^{T^{\text{new}}}$), θV^T, as well as the student's actor and critic parameters: $\theta\pi^S$ and θ_{V^S},

Algorithm 4 INTROSPECT

```
Require: $s_t$, $V^{\text{Src.}}_{\pi^T}$, $V^{\text{new}}_{\pi^T}$, $t$, $\epsilon$, $\lambda$
$h_t \leftarrow 0$
$p \sim \text{Bern}(\lambda^{\max(0,t-\delta)})$
if $t > \delta$ and $p = 1$ then
    if $\left|V^{\text{new}}_{\pi^T}(s_t, a_t) - V^{\text{Src.}}_{\pi^T}(s_t, a_t)\right| \leq \epsilon$ then
        $h_t \leftarrow 1$
    end if
end if
Ensure: $h_t$
```

Algorithm 5 CORRECT

```
Require: $X$, $\pi^S$, $\pi^T$
$\rho^T \leftarrow \emptyset$, $\rho^S \leftarrow \emptyset$
for each $a_i$, $s_i$, $h_i$ in $X$ do
    if $h_i = 1$ then
        $\rho^T \leftarrow \rho^T \cup 1$, $\rho^S \leftarrow \rho^S \cup \frac{\pi^S(a_t|s_t)}{\pi^T(a_t|s_t)}$
    else
        $\rho^T \leftarrow \rho^T \cup \frac{\pi^T(a_t|s_t)}{\pi^S(a_t|s_t)}$, $\rho^S \leftarrow \rho^S \cup 1$
    end if
end for
Ensure: $\rho^T$, $\rho^S$
```

with a learning rate α. This formulation allows for the application of IAA to numerous on-policy algorithms, demonstrated in our experiments with Proximal Policy Optimization (PPO)[130], and extends to off-policy algorithms, excluding the proposed off-policy correction.

4.2 Experiments

We quantitatively and qualitatively analyze our proposed algorithm over two different reinforcement learning domains: Gridworld and Atari. In each domain we examine learning performance when transferring between tasks of varying similarity with the goal of understanding: 1) does our proposed approach yield positive transfer performance between tasks; 2) how does our approach compare to methods based on fine-tuning and action advising; and 3) does our approach allow us to understand what knowledge has been transferred between tasks.

Gridworld: The Gridworld environment consists of four inter-connected rooms and an agent which must learn to navigate to a goal position as shown in Fig. 4.1. In the source task, there are no doors and the agent is free to navigate to any room in order to reach the goal. In the target task, there is exactly one locked door separating the agent from the goal; a key must first be picked up and used on the door in order to unlock it and provide access to the goal.

Atari: The Atari environment consists of a subset of games selected from the Arcade Learning Environment [96]. This is a challenging domain, and has previously been shown to yield negative transfer for fine-tuning-based transfer methods [126]. We have chosen a set of nine games sharing similar visuals and strategies to evaluate for transfer – Air Raid, Assault, Asteroids, Beam Rider, Carnival, Centipede, Demon Attack, Phoenix, and Space Invaders – as well as two games which substantially differ – Ms. Pacman and Pitfall. Air Raid was arbitrarily chosen as the source task and the remaining ten games serve as the target tasks as shwon in Fig. 4.2.

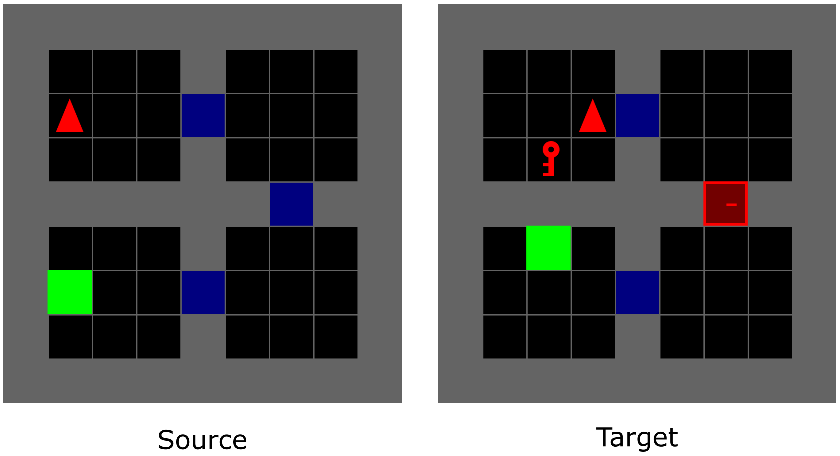


Figure 4.1: The source and target Gridworld tasks. Agent start (red triangle), goal (green square), and key positions are randomized. The agent must learn to navigate to the goal. The source task contains no locked doors while the target task contains exactly one.

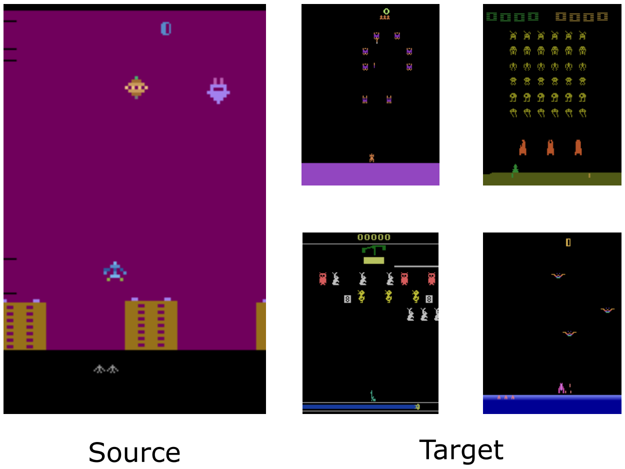


Figure 4.2: The source task (Air Raid) and a subset of target tasks in Atari (Phoenix, Space Invaders, Carnival, and Demon Attack shown). The nature of the task is dependent on the game.

4.2.1 Experimental Setup

In each domain we define a fixed source task for which to train a source policy and then evaluate transfer learning performance to a policy trained in one or more target tasks. We compare our approach to the following six methods:

- **Fine-tuning variants**: (a) Fine-tune (All Layers) - all layers have weights copied from the source policy but are fine-tuned in the target task. (b) Fine-tune - only the input layers have weights copied from the source policy while the remaining are randomly initialized. (c) Frozen - the same as Fine-tune, except the input layer is frozen and not updated.
- **Action Advising**: Action advising at every step with no budget constraint.
- **Action Advising (Decay)**: Action advising which has been augmented with our proposed advice decay rate λ. Rather than issuing advice at every timestep, it is issued according to a decaying probability over time.
- **Policy Distillation (Loss)**: Policy distillation through the introduction of a cross-entropy auxiliary loss between the teacher's and student's policies [128].
- **Policy Distillation (Reward)**: Policy distillation through a reward shaping term which captures the difference in the teacher's critic between the current and previous timesteps [29].
- **Baseline**: A policy trained from scratch in the target task.

In addition to fine-tuning methods used in prior works [126], we chose to compare to two widely-used policy distillation methods which transfer knowledge from a teacher policy through auxiliary losses and reward shaping terms. The source and target policies are trained using Proximal Policy Optimization (PPO) [130].

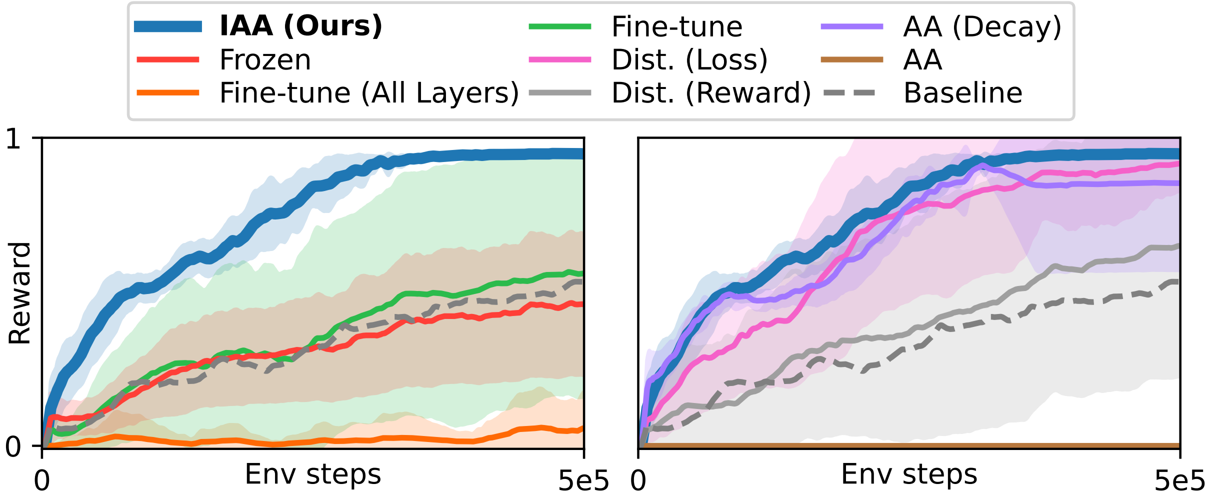


Figure 4.3: The training curves between our proposed method and fine-tuning methods (left) and transfer methods methods (right) given a Gridworld environment where the source and target tasks are similar enough such that the teacher issues helpful advice.

4.2.2 Results - Gridworld

Transfer Performance: In Fig. 4.3 we can see that IAA significantly out-performs all of the fine-tuning based variants. In particular, we can observe that the fine-tuning methods fail to yield positive transfer and under-perform the Baseline, while IAA results in positive transfer with a 66% increase in reward. This is an interesting observation and seems to indicate that the target policy is unable to separate features useful for navigation in an environment without doors to one with locked doors. Additionally, this is one of the only cases we encountered where fine-tuning with a frozen input layer outperforms any of the other fine-tuning variants, indicating that randomly initialized layers after the input are more beneficial – likely due to over-fitting to the source task. Our decayed version of action advising, AA (Decay), fares better in this regard, producing positive transfer with a 48% reward increase. Contrast this with the non-decayed action advising variant, AA, which always gives advice at every timestep and fails to produce any meaningful policy. It is clear that while *some* of the advice is useful in the target task, the target policy must still be allowed to explore on its own. We hypothesize that this is also why Distillation (Reward) yields no performance improvement; the shaping term as proposed in [29] does not decay over time and the student never becomes fully independent. Distillation (Loss) does decay over time, however, and subsequently we see performance gains in line with IAA and AA (Decay).

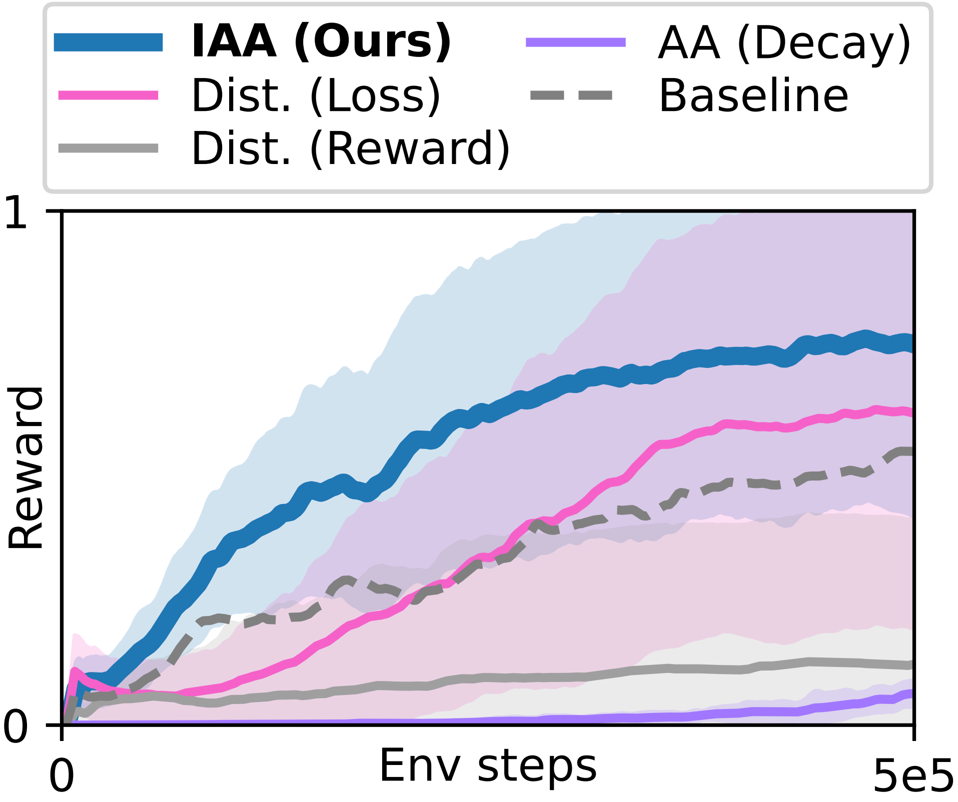
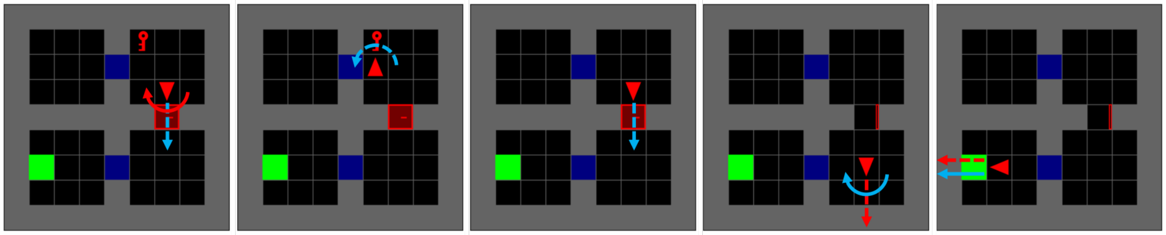


Figure 4.4: The training curves for a Gridworld environment where the source and target tasks differ such that the teacher largely gives unhelpful advice.

Figure 4.4 shows performance when the source task differs more significantly – the agent starts in the bottom rooms and must navigate to the top rooms – and consequently the teacher issues more unhelpful advice. This is where we see the advantage of IAA, as it is able to successfully filter out unhelpful advice and still achieve modest performance improvements, at the expense of increased variance due to the sparse reward. The policy distillation methods and AA (Decay) all perform substantially worse than before due to transferring all knowledge, helpful or not.



Teacher: Forward
Student: Turn right

Teacher: Turn left
Student: Pick up

Teacher: Forward
Student: Toggle

Teacher: Turn right
Student: Forward

Teacher: Forward
Student: Forward

Figure 4.5: A sequence from a single training episode showing the student's sampled action (red) and the teacher's advised action (blue) in Gridworld. Bold text indicates the executed action as determined by IAA given $\epsilon = 0.1$.

Interpretability: Figure 4.5 shows a sequence of states from a training episode for IAA. In the first three states, the teacher gives seemingly unhelpful advice as it advises the student to move in the direction of the locked door before the student has picked up the key needed to unlock it. This is expected behavior, as the source task in which the teacher was trained contains no doors and keys and so the teacher is unaware of the key. However, this advice is filtered out by IAA and the student instead performs its own action and collects the key. Once the agent enters the bottom rooms, the teacher's advice becomes helpful again and is issued to the student in the fourth and fifth states. The fourth state in particular demonstrates helpful advice, where the student would have moved in the wrong direction with its own action, but instead moves towards the goal by following the teacher's advised action.

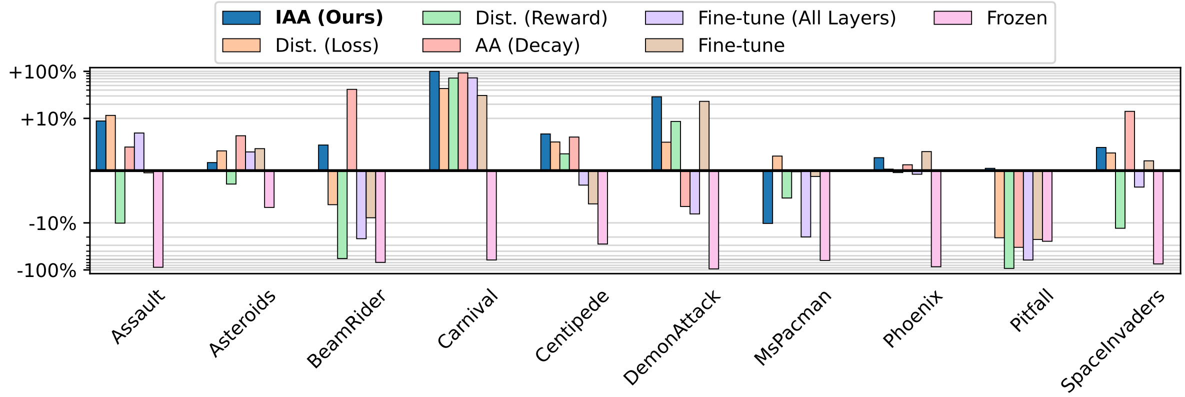


Figure 4.6: The relative performance of our proposed approach and the transfer learning baseline models to the Baseline policy learned in the target task without any transfer. Each value represents the percentage of improvement in the mean reward after 10M environment steps over four independent training runs with randomly initialized weights. The target task is shown along the x-axis with Air Raid used as the source task in all scenarios.

Metric	**IAA**	Loss	Reward	AA	All Layers	Fine-tune	Frozen
PTR	**9/10**	8/10	3/10	7/10	3/10	5/10	0/10
MinT	**–10%**	–21%	–93%	–48%	–90%	–33%	–95%
MedianT	4.6%	3.6%	–3.9%	**5.4%**	–3.0%	0.7%	–66%
MaxT	**99%**	43%	72%	93%	73%	31%	–7%

Table 4.1: Summary of transfer performance shown in Fig. 4.6. Positive Transfer Ratio is the number of tasks for which performance improved, Minimum Transfer is the smallest relative performance achieved over all tasks, and Maximum Transfer is the greatest relative performance achieved. Bold represents the best value for each metric. PTR, MinT, MedianT, and MaxT refer to Positive Transfer Ratio, Minimum Transfer, Median Transfer, Maximum Transfer.

4.2.3 Results - Atari

Transfer Performance: Figure 4.6 shows the relative performance values for each method over each target task after 10M environment steps. Across all target tasks we can observe several trends: IAA consistently yields positive transfer between tasks, achieving the most number of tasks with improved performance; IAA achieves the best *negative* transfer performance for a single task, producing the least worst result when a task fails to transfer; and IAA also results in the best positive transfer performance for a single task. This means that while IAA does not produce the best *median* transfer performance – that falls to AA (Decay) – it does provide the highest positive transfer rate while minimizing performance losses in the event transfer fails. We opted not to show the results for AA as they are not meaningful; this reduces to simply running the source policy in the target task which consistently yields performance worse than the Frozen baseline.

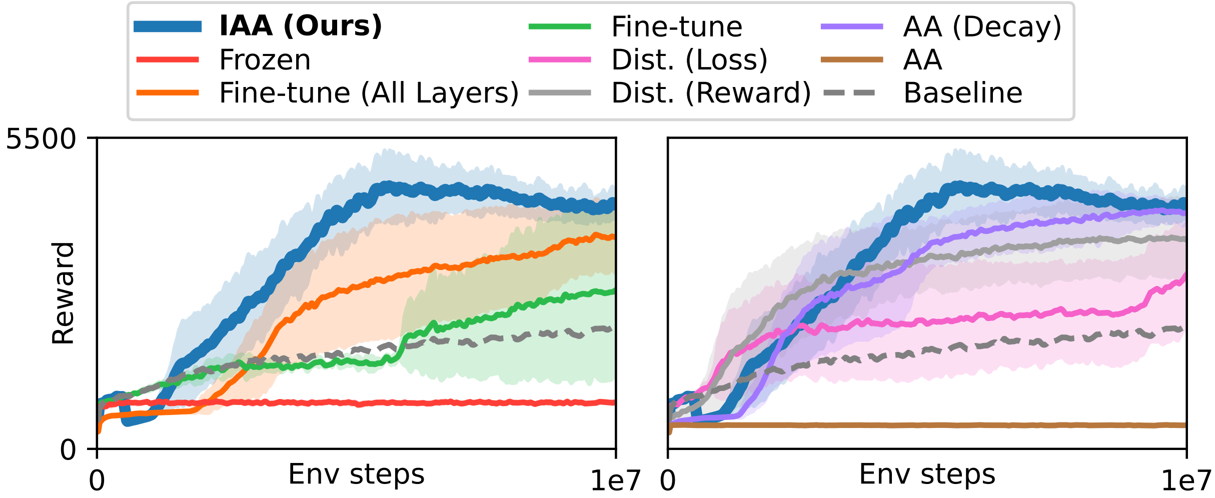


Figure 4.7: The training curves between our proposed method and transfer learning methods (left) and advising methods (right) for Carnival. Each line is the average reward over four independent training runs for 10M environment steps, with the shaded region indicating standard deviation.

The training returns for Carnival are shown in Fig. 4.7 where we can see that IAA produces a significantly higher return than the Baseline while dramatically improving the convergence rate. This task is interesting in that it is one of the few examples where fine-tuning over all layers performs quite well, out-performing fine-tuning over just the input layers. This seems to indicate that the state-action distribution is quite similar between source and target tasks, which would also contribute to the high performance of both IAA and AA (Decay). Another interesting trend is present which sometimes emerges for IAA, which is the drop in performance once the burn-in period is over and advice begins to be issued. This drop occurs after 500K steps – corresponding with $\delta = 500$K – and has two potential causes: 1) the burn-in period wasn't long enough for $V^{\text{new}}_{\pi^T}$ to sufficiently converge to $V^*_{\pi^T}$, and 2) the behavior distribution shift induced by issuance of advice means that even if it *had* converged for the prior behavior policy distribution, it may not be anymore. This invites the possibility of future work which investigates an adaptive introspection threshold conditioned on distribution shifts in behavior policy.

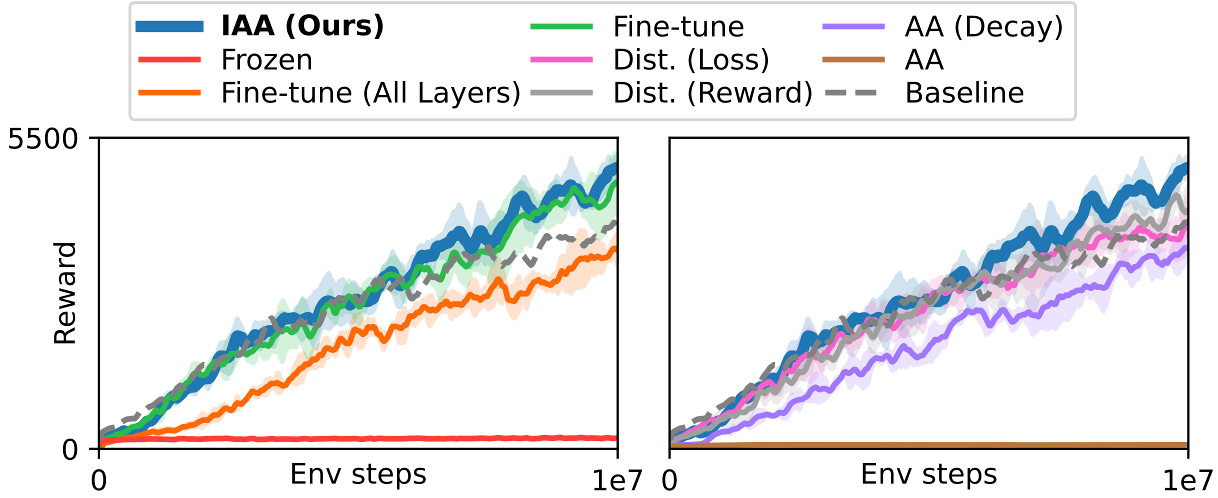


Figure 4.8: The training curves between our proposed method and transfer learning methods (left) and advising methods (right) for Demon Attack. Each line is the average reward over four independent training runs for 10M environment steps, with the shaded region indicating standard deviation.

Figure 4.8 shows training performance from another game, Demon Attack, exhibiting more modest improvements. Rather than the significant increase to convergence rate as seen in Carnival, this task produces slower, more consistent improvements at the same level as fine-tuning. From these two tasks we can observe a more general trend: fine-tuning with a frozen input layer nearly always results in negative task transfer and poor policies after 10M steps. This is unsurprising given that the features encoded from the observation are likely over-fit to the source task, but adds further evidence that issuing advice in the form of state-action pairs which is agnostic to the underlying visual features (like in IAA) may be a desirable strategy.

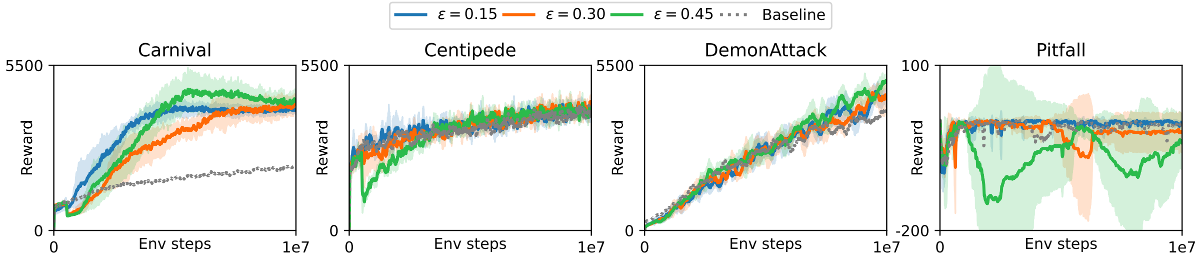


Figure 4.9: Training curves for multiple introspection thresholds of IAA for different Atari tasks.

Figure 4.9 shows the sensitivity of IAA to different values of the introspection threshold. While in general the tested thresholds yield similar results – suggesting that comprehensive hyperparameter tuning may not be necessary – an overly high threshold can cause large performance drops for certain tasks. In Centipede for instance, $\epsilon = 0.45$ produces a sharp drop in performance right after the burn-in period has elapsed and advice is issued. Similarly, in Pitfall, a target task which differs significantly from the source task, we can observe increasing variance in the returns as more advice is issued. Intuitively, we observe that IAA should be used with a more conservative introspection threshold for target tasks which differ significantly from the source task, which consequently leads to a decrease in expected performance improvement ($\epsilon = 0$ reduces to baseline performance).

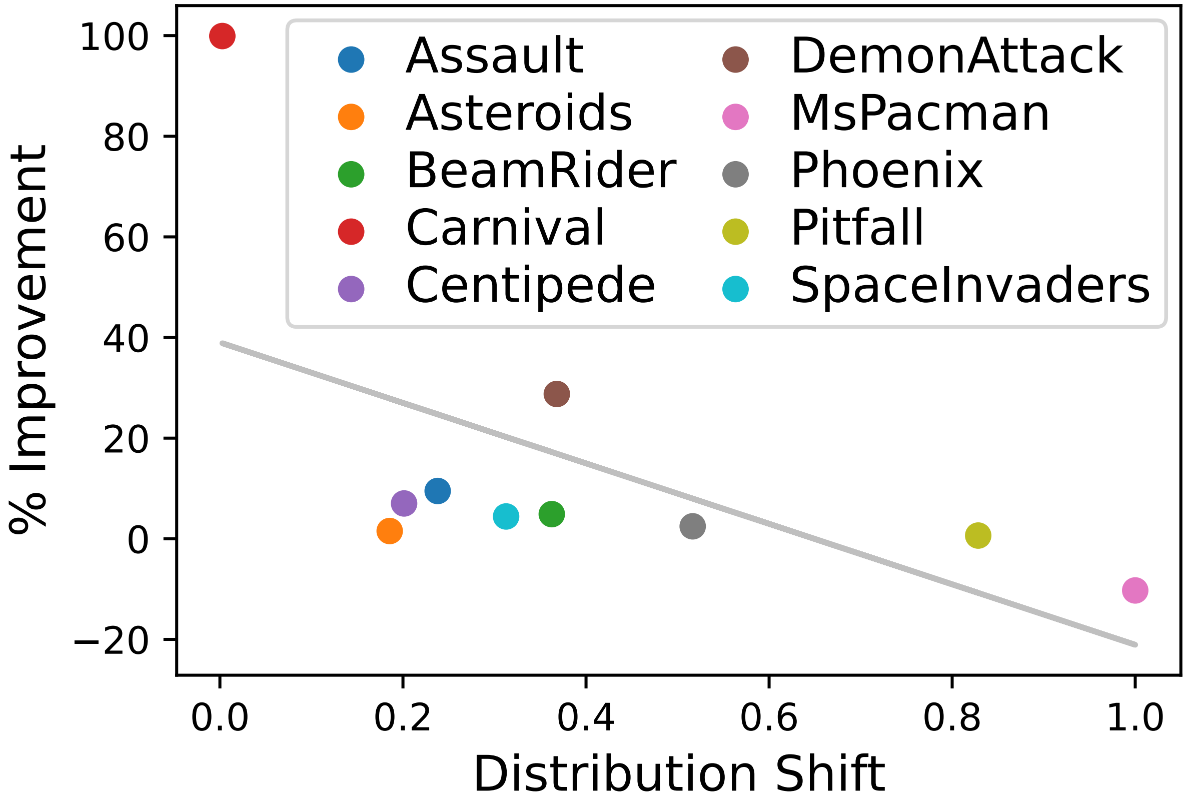


Figure 4.10: The relative performance of IAA compared to the distribution shift of each Atari task.

Figure 4.10 examines this in further detail. We collect states and actions that are sampled in the teacher's game and the student's game respectively. They are then computed to form distributions, and the difference of the distributions is the distribution shift. The result shows that there is indeed a negative correlation between performance and distribution shift between source and target observation distributions, with $R^2 = 0.34$.

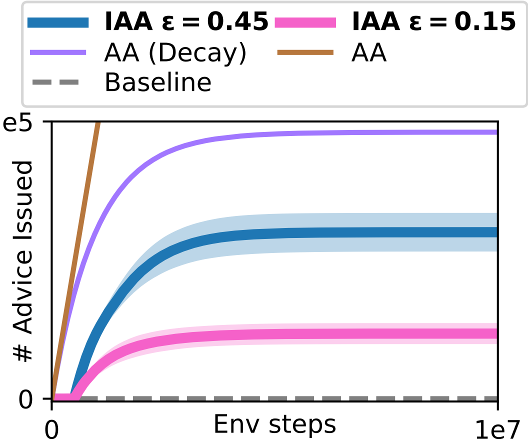


Figure 4.11: The amount of advice issued by each advising algorithm for the game Carnival.

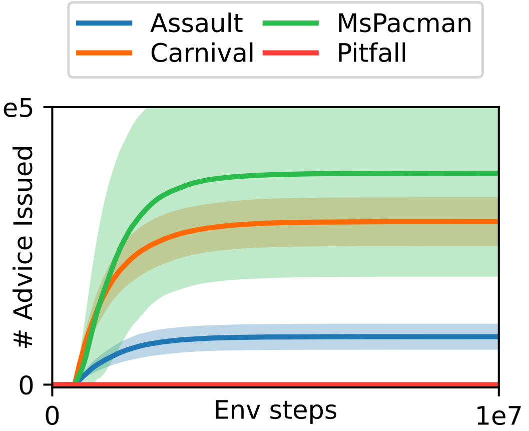
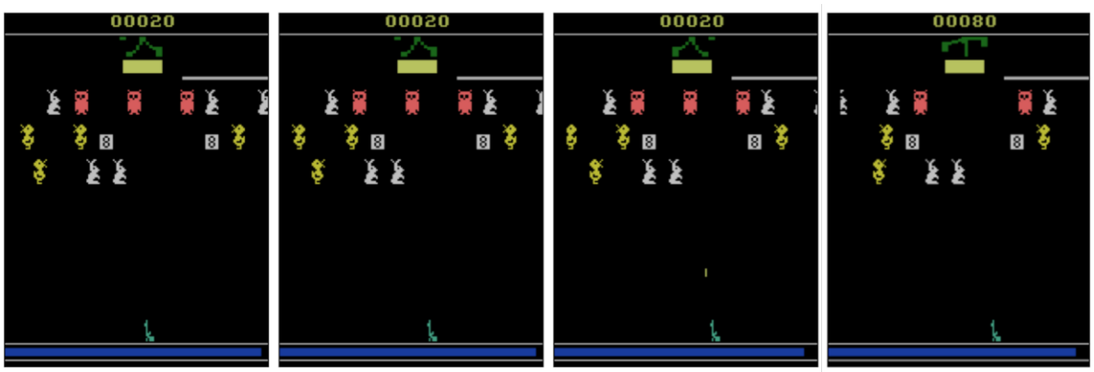


Figure 4.12: The amount of advice issued across Atari games by IAA given $\epsilon = 0.15$.

The amount of advice issued by the advising algorithms for a single game, Carnival, is shown in Fig. 4.11. As in the Gridworld environment, we see again that IAA is successful in identifying only useful knowledge and achieves higher returns by giving fewer pieces of advice. Figure 4.12 shows the amount of advice issued for IAA across target tasks when the introspection threshold is fixed. Unsurprisingly, this indicates that some target tasks have an expected reward distribution more similar to the source task than others, resulting in more advice being issued by IAA. However, this figure also indicates that $V^*_{\pi^T}$ may be more difficult to learn in some tasks, resulting in a large variance in issued advice as seen in Ms. Pacman. This variance is undesired and as seen in Fig. 4.6 led to the only negative task transfer for IAA, likely due to the introspective teacher failing to discriminate beneficial advice from non-beneficial advice.



Teacher: Fire
Student: Move

Teacher: Fire
Student: Move

Teacher: Move
Student: Fire

Teacher: Move
Student: Move

Figure 4.13: A sequence of states from a single training episode showing the student's sampled action (red) and the teacher's advised action (blue) in Carnival. Bold text indicates the executed action as determined by IAA given $\epsilon = 0.45$.

Interpretability: Figure 4.13 shows a sequence of states from a training episode for IAA in Carnival. We can observe that in the first two frames the teacher advises the student to fire, an action which is filtered out in the first state but issued and executed in the second. Afterwards, the teacher advises the student to perform movement actions in order to reposition the agent so further fire actions can be taken. In the fourth state we can observe that the advice is indeed helpful, as the advised action results in a high-value target being destroyed in the furthest row and a 60 point increase to the score. This example is demonstrative of the insight which IAA provides into which knowledge is transferred between the teacher and the student; an external observer need only examine the states for which advice is issued. While prior methods such as policy distillation also transfer knowledge, it is much more difficult to analyze an auxiliary loss term for a particular state or even a reward shaping term – particularly as rewards require one to sample actions and perform a comparative analysis to other state-action rewards.

4.3 Conclusion

In this chapter, we have introduced Introspective Action Advising (IAA), that is capable of distinguishing which knowledge is beneficial to transfer from the teacher to the student, while at the same time providing an opportunity for more fine-grained knowledge transfer.

Chapter 5

Teacher's Adaptation to Improve Advice based on Estimating Student's Rewards

One of the challenges inherent to the teacher-student paradigm, however, is issuing helpful advice in the presence of reward misalignment between the teacher's source and student's target tasks. This means the teacher can be an expert that is trained with a reward different from the student's. A teacher which blindly issues advice without considering task differences can harm the student's learning by actively guiding it to regions of the state space which yield low rewards [54]. Prior approaches have attempted to address this by identifying the subset of states which is helpful when following advice, and only issuing advice in these states [54], but this leads to a more conservative teacher and is thus less beneficial to the student [23]. In this chapter, we instead propose an approach in which the teacher adapts its own advice such that it maximizes the reward associated with the target task. Rather than identifying *which* advice is helpful as in prior methods, our approach adapts the advice such that *all* of it is helpful. Upon publication of this thesis, the manuscript has not been officially published yet, but is accepted and preparing for camera-ready version and code release within the next month. We refer readers to the original paper [56].

Our proposed teacher, the Reward-Adaptive Teacher (RAT), performs the following steps, it: a) estimates the unknown target reward function based on the student's observed returns, b) uses an internal world model to optimize its own policy with respect to the estimated reward function, and c) issues action advice from the updated policy. In line with prior action advising formulations, we assume that our teacher lacks direct access to the environment when advising. Thus, the key to our approach is the usage of a learned state-action mapped to reward function, to adapt the teacher's policy with respect to the target task. The primary contributions of our work are as follows: 1) We introduce a novel reward-adaptive teacher which is robust to differences in rewards between the source and target tasks, leading to improved advice quality. 2) We validate our approach through empirical testing in different environments: a simulated urban search and rescue (USAR) scenario, a skiing game drawn from the Arcade Learning Environment [96] and Pacman. We empirically show that our method produces advice of higher quality than existing methods, improving the sample efficiency of reinforcement learning methods. We also examine calculating the transition functions in a simulated urban search and rescue (USAR) scenario and a real-world single-agent navigation task, to estimate the upper limit of the teacher. Our method delivers superior performance compared to existing state-of-the-art methods in both environments.

5.1 Background

Action advising, which, at its core, leverages the knowledge of a *teacher* to expedite the training process of a *student* agent, has a long-standing history in the field of RL [83, 98, 143]. Particularly, in action advising, the teacher provides advice to the student agent by suggesting actions in a given state, helping the student agent discover the correct actions faster than by relying purely on their own exploration. To this end, Torrey et al. [145, 146] proposed methods that map human knowledge to RL tasks by utilizing a user-provided mapping while Maclin and Shavlik [97] discuss giving advice by an external observer.

Human teachers have the remarkable ability to evaluate whether or not their advice should be adjusted given the student's observed experience. Recent work in the domain of action advising that utilize an expert policy in order to provide advice, struggle when the domain of the teacher differs from the domain of the student, as most approaches assume the task to be the same [31, 111]. To decide whether or not to give advice, Torrey and Taylor [144] utilizes a heuristic approach while later work formulates the advising task as a separate learning problem [177]. In such setting in which the teacher is sub-optimal, as the domain of the student shifted from the teacher's, the utility of the teacher heavily depends on the quality of its advice [165]. This work addresses this problem with a curriculum of teachers that constantly observe and subsequently adapt their behavior in an approach inspired by DAGGER [125]. However, such an approach requires the existence of multiple teachers. An alternative approach that addresses this domain shift is presented in Guo et al. [54], in which the teacher does not only provide the advice itself, but also an explanation as to why the advice was given. While this enables the student to reflect on the given advice, the teacher does not improve its performance and continues to provide fixed advice [172]. The work presented in Campbell et al. [23] introduces Introspective Action Advising, which allows the teacher to reflect upon its own advice, ultimately learning to withhold advice if the teacher's expected outcome does not align with the student's experience.

In this chapter, we directly address the domain shift between the student's and teacher's tasks by quickly adapting the teacher's reward model given the student's observed behavior and returned reward. This allows our teacher to not only withhold advice that is expected to negatively impact the student, but also allows it to go beyond prior work by actively improving the quality of issued advice.

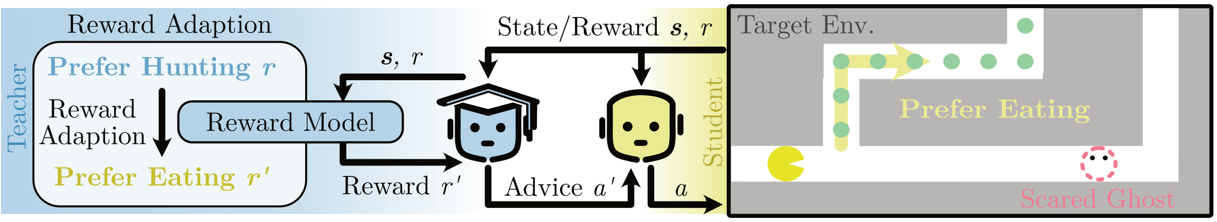


Figure 5.1: An overview of Adaptive Reward Teaching. The teacher initially issues action advice to the student from a policy trained in the source task – leading to low rewards in the target task due to differences in the reward functions. For example, the teacher could be a ghost chaser while the student is a conservative player that always avoids ghosts and eats food. The teacher has a much higher reward of eating ghosts than the student, however, it would not find this out until examining the data collected by the student. After observing the student's trajectory returns, the teacher adapts its policy using its so as to maximize the student's reward in the target task.

5.2 Methods

In this chapter, we consider a scenario in which a teacher that has already learned a policy π^T in a source task seeks to advise a student learning a (randomly initialized) policy π^S in a target task. We assume that the state-action spaces of the source and target tasks are the same, however, the tasks exhibit different reward functions. This presents a problem for action advising in that the teacher may issue advice that leads the student to regions of the state space associated with low rewards in the target task, despite yielding high rewards in the source task. Our goal in this chapter is to adapt the teacher's policy π^T such that it maximizes the reward associated with the target task, rather than the source task in which it is trained, side-stepping the above problem.

Our approach consists of three main steps:

1) learning a model of the reward function of the source task during teacher training, involving the teacher collecting data in its own environment;

2) estimating the unknown reward function associated with the target task when at the beginning of advising the student, so as to collect the true reward through the student's interaction with its environment different from the teacher's, and

3) adapting the teacher's policy via fine-tuning so as to maximize the target task's reward, which can then be used to issue action advice. While step 1 is only performed once prior to student training, steps 2 and 3 are performed throughout the student's training process, leveraging the student's interactions with the environment as training data.

Intuitively, our approach can be understood as the teacher observing the student's returns in the target task, estimating how they differ from those in the source task in which it was trained, and then tailor its policy to the target task and afterwards advising the student. The process is shown in the Figure 5.1.

The teacher observes everything the student observes/receives from the environment. Therefore, it knows the reward of students at every step. However, it does not know about the student's current policy or the entire reward space of the student. If the teacher does not advise, the student takes the action that is computed from its own neural network/internal model. Our method is different from vast of the domain adaptation work in that the teacher collects the ground-truth reward from the student's, so that it only observes a small amount of data as the student interacts with the environment, and also needs the teacher's reward network as the base model to get corrected. In most domain adaptation tasks, there is no need to consider the agent exploring the environment, as they are simply supervised learning questions. Moreover, we don't require access to the data features that summarize the student's reward space.

We majorly focus on the reward difference instead of other components to simulate an environment, such as transition functions, as reward functions may reflect the preference that can be understandable. Intuitively, the states and actions to gain high rewards are the desirable behaviors for the agents, while studying other challenging components may lead to more intricacy and confusion.

Algorithm 6 Adaptive Reward Teaching

```
1: Input: Trained teacher policy $\pi^T$, advising strategy $H$, adaptation period $\mu$.

2: Prior to student training
3: Teacher collects $\mathcal{D}_T = \{(s_t, a_t, r_t)\}_{t=1}^{\omega}$ in its environment.
4: Teacher trains reward model $\mathcal{M}(\mathcal{D}_T)$.

5: During student training
6: if iteration $\leq \mu$ then
7:     Teacher collects $\beta$ student trajectories $\mathcal{D}_S = \{(s_t, a_t, r_t)\}_{t=1}^{\beta}$
8:     Reward adaptation: $\mathcal{M}' = \mathcal{M}(\mathcal{D}_S) + \mathcal{E}(\mathcal{D}_S)$
9:     Policy adaptation: $\pi^T = \text{argmax}_\pi \, \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{M}'(s_t, a_t)\right]$
10: end if
11: Teacher issues advice $a_t^T \sim \pi^T(a_t|s_t)$ if $H(s_t) = 1$
```

Learning Environment and Reward Dynamics: Prior to advising a student in the target task, the teacher learns an approximation of the source task's reward function $\mathcal{M}$. Formally, for each time step t, the teacher observes its state s_t, takes an action a_t under policy π^T, and receives a reward r_t from the environment. The collected data takes the format of $\mathcal{D}_T = (s_t, a_t, r_t)_{t=1}^{\omega}$, where ω represents the total number of time steps in the teacher's data sets. (Algorithm 6 Line 3). $\mathcal{D}_T$ is utilized to train a reward prediction model, which is a neural network $\mathcal{M}$ parameterized by ϕ (Algorithm 6 Line 4).

$\mathcal{M}$ aims to estimate the reward r given a state-action pair (s, a): $\mathcal{M}(s, a; \phi) \to r$. Thus the training objective for $\mathcal{M}$ is to minimize the difference between the predicted reward and the actual reward:

$$L(\phi) = \mathbb{E}\left[||\mathcal{M}(s_t, a_t; \phi) - r_t||^2\right], \tag{5.1}$$

Reward Adaptation:

Once the student begins training, ART observes the student's interactions with the target environment for β episodes, gathering the student's state-action-reward data, denoted by $\mathcal{D}_S = (s_t, a_t, r_t)$, where $t \in [1, \beta]$ (Algorithm 6 Line 7). To accommodate the differences in the rewards of the source and target tasks, ART introduces an error prediction model $\mathcal{E}$, parameterized by ψ. This model is trained to predict the discrepancy between the rewards estimated by the initial reward model $\mathcal{M}$ and the actual rewards obtained by the student in the target task.

The error prediction model $\mathcal{E}$ aims to estimate the error ϵ_t given a state-action pair (s_t, a_t): $\epsilon_t = \mathcal{E}(s_t, a_t; \psi)$. The error ϵ_t represents the difference between the predicted reward and the actual reward observed by the student: $\epsilon_t = r_t - \mathcal{M}(s_t, a_t; \phi)$. The training objective for $\mathcal{E}$ is to minimize the difference between the predicted error and the actual error:

$$L(\psi) = \mathbb{E}\left[||\mathcal{E}(s_t, a_t; \psi) - (r_t - \mathcal{M}(s_t, a_t; \phi))||^2\right], \text{where} (s_t, a_t, r_t) \in \mathcal{D}_S. \tag{5.2}$$

Following the training of the error prediction model, the adapted reward at each step is computed as the sum of the predicted reward and the predicted error: $\mathcal{M}'(s_t, a_t) = \mathcal{M}(s_t, a_t; \phi) + \mathcal{E}(s_t, a_t; \psi)$. This adaptation allows the teacher to more accurately align its reward predictions with the actual rewards in the target task, facilitating more effective policy adaptation and advice to the student.

Policy Adaptation: After adjusting its reward prediction model, the teacher uses the updated $\mathcal{M}'$ to perform a limited number of reinforcement learning iterations and improve its policy π^T with respect to the target task (Algorithm 6 Line 9), by modifying the parameters of π^T based on the standard update rule: $\theta' = \theta + \alpha \nabla J'(\theta)$, where $J'(\theta)$ maximizes the expected cumulative reward under the new reward distribution:

$$J'(\theta) = \mathbb{E}\pi^T \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{M}'(s_t, a_t; \phi) \middle| s_t, a_t \right] \tag{5.3}$$

This adaptation process allows the teacher to improve its policy and may be performed numerous times throughout the student's training process. The updated policy is subsequently used to issue future action advice to the student according to an advising heuristic $H(\cdot)$ – our approach is agnostic of the specific heuristic which is used. The effectiveness of this adaptation step is contingent on how accurate the reward model is. As a result, adaptation steps performed early in the interaction when a limited number of student returns have been observed are likely to be more noisy than those performed later in training.

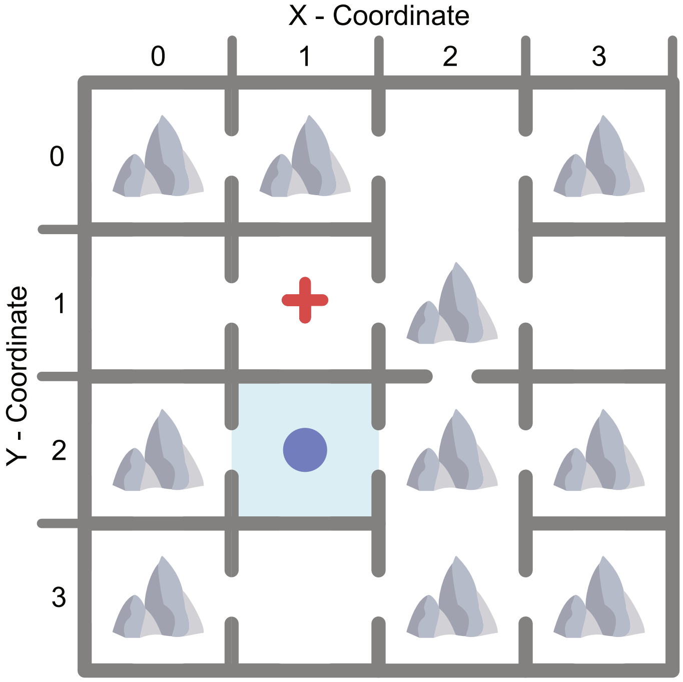


Figure 5.2: Search and Rescue: The agent (blue circle) navigates to the room where a rubble is removable, and a victim represented by red cross needs to be rescued.

5.3 Experiments

We empirically evaluate our proposed adaptive teacher in three separate scenarios. These domains have been selected such that: 1) the teacher and student have different rewards, and this leads to different policies and behaviors after the training convergence; 2) the student performance is better when advised by an adaptive teacher, compared to student's performance with a teacher that never adapts or when the student receives no advice; and 3) the student reward may be inferred from a small batch of data at the beginning of its training.

We use the following environments. USAR: A Gridworld which simulates an urban search and rescue environment in which the agent needs to remove rubble and rescue trapped victims. Skiing: The ALE [96] Skiing game game where the agent moves left or right or having no actions. The agent always sits on top of the screen and the background keeps rolling up to simulate the agent is moving downwards. Pacman: A variant of the Pacman game with a feature-based state representation. The maximum steps for are 500, 1500, and 600.

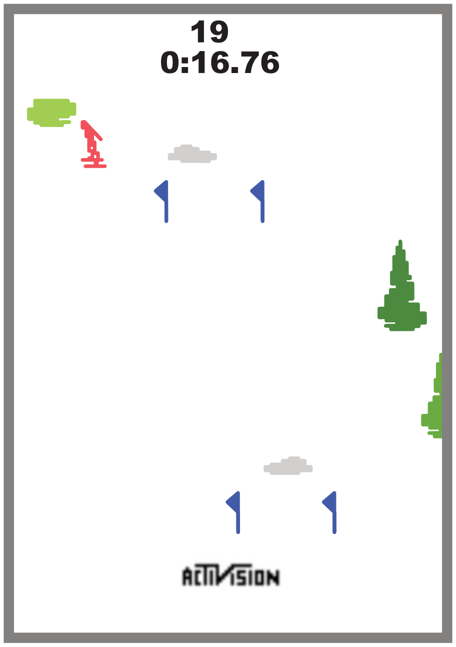


Figure 5.3: Skiing: The agent (red) needs to avoid the trees (green) and gates (blue). It can pass the gates if it goes through the empty space between gate pairs.

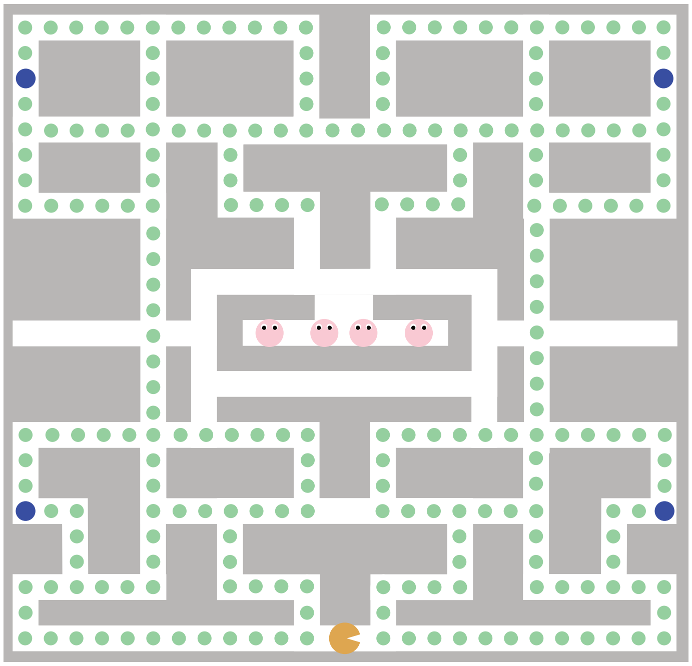


Figure 5.4: Pacman: The pacman (yellow) eats up food (green) and power pellets (blue). In the default mode, the ghosts (pink) chase the pacman and can eat it up; After the pacman eats up a power pellet, it can eat ghosts.

5.3.1 Action Advising Comparisons

To evaluate the effectiveness of our adaptive advising method, we first compare the learning performance of the student agent under the action advising of different types of teachers. The types of teachers we compare are:

Adaptive Reward Teaching (ART): The teacher is trained in the source environment and adapts its policy to align with the target task's reward function (used by the student). This adaptation occurs within one or a few iterations of the beginning of student's training, leading to a noticeable improvement or 'jumpstart' in the student's performance, and helps the student to converge faster.

Non-adaptive Teacher: The teacher is trained in the source environment and provides consistent advice throughout the learning process without adapting its policy. As a result, the advice may be suboptimal with respect to the target environment's reward function.

Ideal Teacher: This hypothetical teacher is an optimal policy within the student's target environment. The ideal teacher is included as a baseline to represent the upper limit of teaching quality.

No Advice: The student agent receives no advice and learns from scratch. This approach typically shows high variance in performance and, although it can potentially converge to an optimal solution, there is no guarantee of this outcome.

Additionally, our methodology involves a gradual decay in the probability of advice being given. When the probability goes to zero, it means no advice is given from the teacher, and thus the reward reflects the actual student's performance. Although there are a range of works focusing on when to issue the advice, we utilized the simple heuristic method and focused on adaptability. Key hyperparameters in this decay advising process include the initial probability of advice, the final period when the advice stops, and the rate at which the probability is reduced. We present the best empirical results from various combinations of these parameters to illustrate the effectiveness of our adaptive advising approach in different environment settings. This comparative analysis allows us to robustly evaluate the benefits of adaptive advising in enhancing the student's learning process across various environments. Each of the methods is averaged over 5 episodes. Typically the student advised by the ideal teacher is the best, and the student received no advice or advised by the non-adaptive teacher is the worst. We show ART can boost performance of the student after the teacher has finished improving its policy, to the level comparable with the ideal teacher.

USAR: We apply our method to a simulated search and rescue scenario [42, 86] modeled as a Gridworld environment. Shown in Fig. 5.2, , this is a single agent scenario in which the agent (blue circle) must navigate the environment, remove rubble, and rescue victims (red cross). In this variant, there is a single victim and single piece of rubble randomly located within the 14 room layout. The source environment only rewards the agent for rescuing the victim with a per-timestep penalty – no reward is given for removing rubble. The optimal behavior in the source environment is to ignore rubble and immediately locate and rescue the victim. To induce a change in rewards, the target task rewards the agent for both removing rubble and rescuing victims. Correspondingly, the optimal behavior in the target environment is to remove as much rubble as possible before rescuing the victim.

The baseline comparison result is shown in Figure 5.5 (left). The student receiving no advice learned the slowest and had the most variance but ultimately converged to the optimal reward. The student advised by the non-adaptive teacher received sub-optimal advice, yet it was still beneficial and resulted in an initial performance improvement over the baseline student with no advice. Despite the sub-optimality, this student was able to converge to the optimal reward due to the decaying probability of issuing advice. Our proposed method, ART, achieved performance comparable to the student advised by an ideal teacher. This ideal teacher is the agent that is trained to be optimal with the student's rewards, and we empirically have tuned the hyper-parameters (see supplementary material section) to ensure the most effective advising process. It can be observed that ART could soon boost the student's performance to reach the student's performance as advised by an ideal teacher. We measured if after the teacher finished adaptation the student's learning curve overlapped or surpassed the one advised by the ideal teacher. We note that the teacher issued no advice in the first 256K training steps, as observations were collected only to adapt the teacher's reward function.

Skiing: In the Atari [96] Skiing environment shown in Figure Fig. 5.3, the primary objective is to navigate a downhill course as quickly as possible while avoiding obstacles such as trees and navigating through gates. The source environment uses the default reward structure from ALE, where the agent receives $+10000$ score for passing through a gate and -1000 for hitting an obstacle, with a per-timestep penalty. The optimal behavior results in the agent passing through gates near the agent's position while avoiding obstacles. Empirically, this results in an agent trained in the source environment to pass through 58% of the gates on average. The target environment increases the reward for passing through gates by 10X, thereby encouraging the agent to focus on passing through every gate while avoiding collisions. This necessitates more conservative behavior as gates are worth relatively more compared to the per-timestep penalty. The resulting optimal agent behavior passes through 87% of the gates on average.

Empirical results are shown in Figure 5.5 (middle). The student receiving no advice exhibited training instability and often failed to converge to optimal behavior. The student with the non-adaptive teacher did converge to optimal behavior, however, suffered a noticeable drop in performance once the advising probability fully decayed to 0 and advice was no longer issued. This was not an issue for the student advised by the adaptive or ideal teacher, where the reward growth is consistent and smooth even when advising stopped.

Pacman: Pacman is shown in Figure Fig. 5.4, which is a widely used benchmark for reinforcement learning. The reward encourages the agent to eat food which increases the agent's score as well as power pellets, which cause the ghosts to entire a "frightened" state after which they

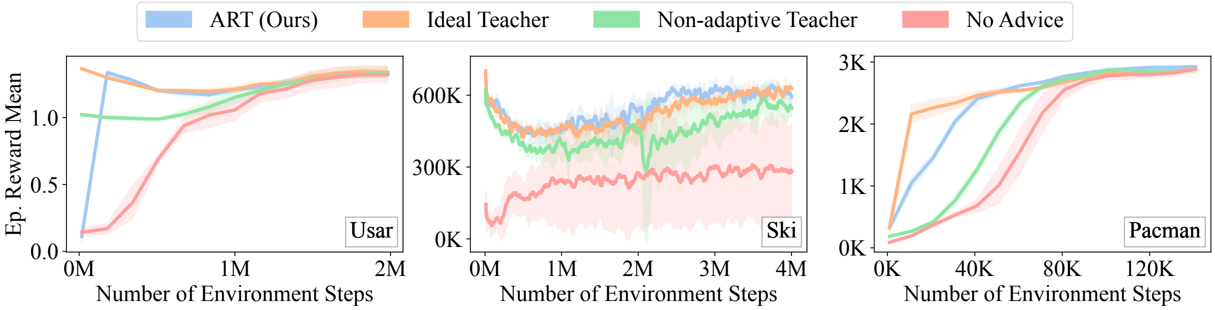


Figure 5.5: Training curves for a student which receives action advice from our proposed method as well as the action advising variants described in Sec. 5.3. Left-to-right: USAR, Skiing, and Pacman.

can be eaten for a large increase in score. The source environment highly rewards the agent for consuming ghosts, resulting in an optimal behavior that is characterized by a strong preference for chasing ghosts, sometimes leading to risky situations or hesitation between pursuing ghosts and consuming more pellets. On average, an optimal agent trained in the source environment consumed approximately 6.4 ghosts and 45% of the available food.

The target environment, in contrast, rewards the agent for collecting food while avoiding ghosts, as the agent is not rewarded for eating ghosts even when they are in the "frightened" state. On average, the optimal agent trained in the target environment consumed 1.4 ghosts and 85% of the food.

The results are shown in Figure 5.5 (right). Our experimental results reveal significant performance disparities between students advised by adaptive and non-adaptive teachers. The non-adaptive teacher, though helpful in directing the student to eat food, demonstrates a starkly inferior performance compared to the adaptive teacher as it encouraged chasing ghosts. In such cases, the student's performance aligns more closely with an untrained, random agent, often getting quickly eaten by ghosts with minimal food consumption. Notably, the difference in performance between students trained with adaptive and non-adaptive teachers in the Pacman environment was more pronounced than in the USAR or Skiing scenarios. While the adaptive teacher's performance gradually converges towards that of an ideal teacher, it demonstrates a clear advantage over other baseline approaches. Our experiments, also additionally indicated that an initial advising probability less than 1.0 (specifically 0.8 in our case) was optimal. This allowed for sufficient exploration by the student while still benefiting from the teacher's guidance. Overall, the Pacman environment is more challenging for the teacher to adapt since there is a large shift in dynamics after a power pellet is eaten. This challenge is further exacerbated in our scenario, given the difference in reward between the source and target environments.

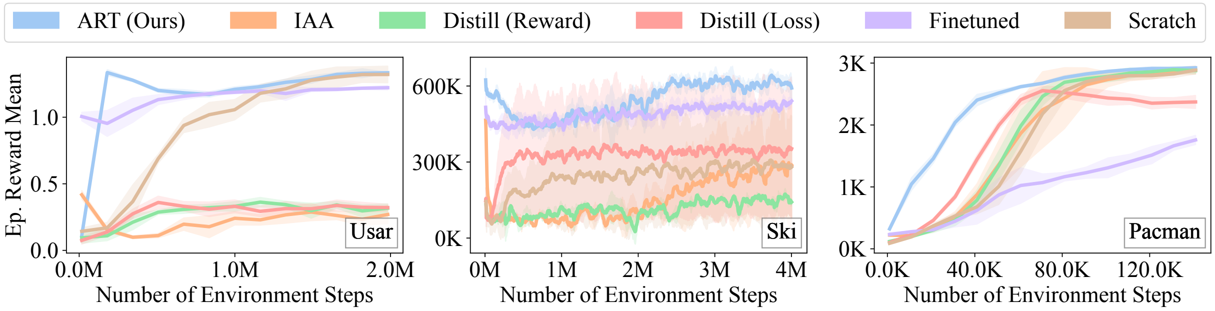


Figure 5.6: Training curves for a student which receives action advice from our proposed method as well as the methods defined in Sec. 5.3.2. Left-to-right: USAR, Skiing, and Pacman.

5.3.2 Transfer Learning Comparisons

We compare our approach to three state-of-the-art transfer learning methods. All methods use the same teacher policy which has been pre-trained with the reward in the source environment and used to transfer to a student learning from scratch with the reward in the target environment.

Introspective Action Advising (IAA)([23]): Action advising method in which the teacher identifies helpful advice based on the expected state-value in the target task.

Policy Distillation (Loss)([128]): Also known as kickstarting, this is a policy distillation method which adds an auxiliary loss term based on the cross-entropy between the student and teacher policies.

Policy Distillation (Reward)([29]): Policy distillation method which introduces a reward shaping term that captures the difference in the teacher's critic between the previous and current timesteps.

We also evaluate against several simple ablations.

Finetune: Student policy initialized with the teacher's policy and trained in the target task.

Random: A randomly initialized policy trained from scratch in the target task.

In these experiments, our approach uses a heuristic function such that the probability the advice is issued decays over time, and finally the student has learnt to perform tasks without teacher's advice.

The Adaptive Teacher strategy demonstrated superior performance in the USAR and Skiing scenarios when contrasted with other benchmarks in Figure 5.6. This advantage, however, was less pronounced in the more complex Pacman environment, though it still maintained a lead. The efficacy of the Introspective Action Advising (IAA) approach was particularly notable in environments where differences could be assessed through value function estimations. This allowed for more advanced advising strategies, as the teacher's insights were grounded in a robust understanding of the environment's dynamics. However, in scenarios where the reward changes leading to policy differences, the value-based insights provided by IAA became less effective. In such cases, the performance of IAA aligned more closely with that of traditional policy distillation methods, which rely on mimicking teacher behaviors either through reward or loss minimization. This convergence highlights the challenges posed by environments with altered rewards and underscores the need for adaptable advising.

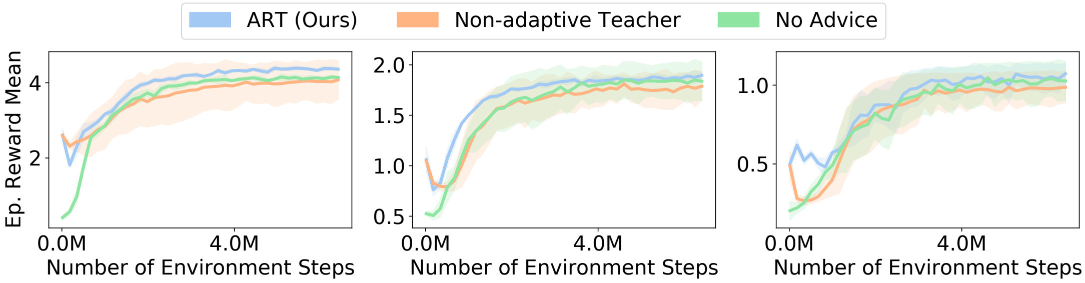


Figure 5.7: Training curves for our proposed method as well as a student advised by a non-adaptive teacher and a student with no advising. The reward for removing each piece of rubble varies from 0.5 (left), 0.25 (middle), and 0.125 (right).

5.3.3 Adaptability with respect to Policy Variance

We further explored the impact and limits of an adaptive teacher on a student's learning performance in a variant of the USAR environment which contained 10 rubble and 1 victim. The agent's task was to find the victim and clear the rubble, with the episode ending either upon finding the victim or after 500 steps. For the teacher, rewards were given for finding victims (+1) and removing rubble (+1), and these rewards decayed over time. The student's reward differed from that of the teacher, with varied levels of difference. While finding the victim always yielded a reward of 1.0 for the teacher and all the student policies, we varied the reward for rubble removal between 0.0625 and 0.5 for different students. This necessitated the agent to balance the trade-off between conserving steps to maximize the reward for finding the victim as soon as possible and exploring more rooms to locate all the rubble and remove them. As the teacher received high reward of removing rubble and would remove all of them before finding the victim, the varied smaller reward for removing rubble by the student indicated the priority of removing rubble decreased for different students. This setup introduced more variability in behavior and required longer times for the agent's policy to converge. We conducted experiments with the teacher adapting to each of the different student policies and advising them. Each scenario was tested over five episodes, and the results were evaluated with 1,000 trajectories. Instead of presenting each of the learning processes, we evaluated the performance to demonstrate adaptability, compared to no advice and non-adaptive decay advise. The performances are evaluated after the student's training has been converged, meanwhile the teacher has already finished its advising and the evaluation is done for the students' policies.

First, the adaptive teacher improved the student's policy convergence as shown in Figure 5.7. This improvement was particularly noticeable when the reward for rubble removal was set at moderate values (0.25 and 0.5), which is closer to the teacher's 0.1. The complexity of the task required the agents to balance between exploring and quickly finding the victim. For students where the reward was smaller, the adaptive teacher's impact was less pronounced. The non-adaptive teacher also helped improving convergence in general, but less so than the adaptive teacher.

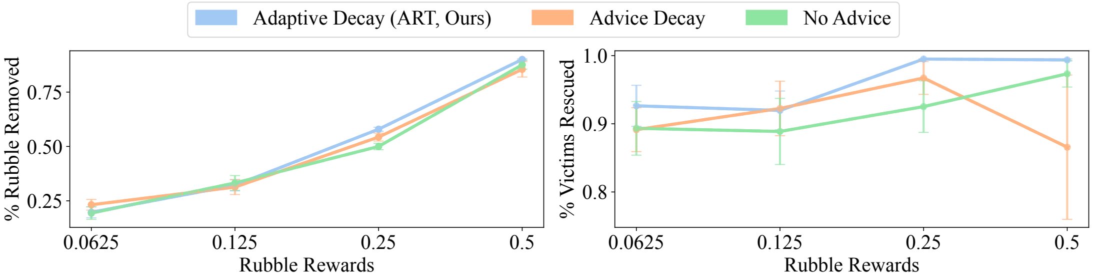


Figure 5.8: USAR 10 Rubble: Percentage of removed rubble (out of 10) and triaged victims (out of 1) over 1000 rollouts across five student seeds each.

Intuitively, as shown in Figure 5.8 (left), the decreased reward of removing rubble would lead to a decreased number of rubble removal of the converged student's policy. Therefore, from the general trend, we observed an increase of rubble removal as expected. Notably, when the reward for rubble removal was equal to that for finding the victim (1.0), the teacher and student achieved same performances after convergence. Both the adaptive and non-adaptive teacher would not affect the number of rubble removal after convergence at the end. However, the adaptive teacher consistently outperformed both the non-adaptive teacher and scenarios with no advice in helping the agent find victims as shown in Figure 5.8 (right). It guided all the students to find the victims and reduced the errors of the victims not being found. This was especially evident with reward settings of 0.5, where the trade-off of exploring and quickly finding the victims was the hardest to resolve. The non-adaptive teacher, which always prioritizing removing rubble, led to a high error and unstable performance of the students, while the adaptive teacher's guidance led to more stable and efficient victim-finding strategies. In summary, the experiment demonstrated the efficacy of an adaptive teacher in guiding an agent through varied students' rewards. The adaptive teacher not only enhanced the agent's ability efficiently but also expedited the policy convergence process, particularly in scenarios with moderate reward discrepancies.

5.4 Supplementary Material

5.4.1 Hyper-parameters

The hyper-parameters most relevant to the performance of ART are as follows:

1. **Initial Follow Teacher Probability (IFTP)** This is the same as the initial advising probability that the teacher issues advice. If IFTP=1.0, the teacher always gives advice, otherwise the teacher advises with a probability and the student has more freedom exploring its own environment. In our case, the student always follows the teacher's advice when issued.
2. **Burn In (BI)** This is the beginning duration where the teacher does not give advice but instead lets the student explore the environment, and collects the student's reward data.
3. **Advise Decay Rate (ADR)** The teacher gradually lowers its probability of giving advice,

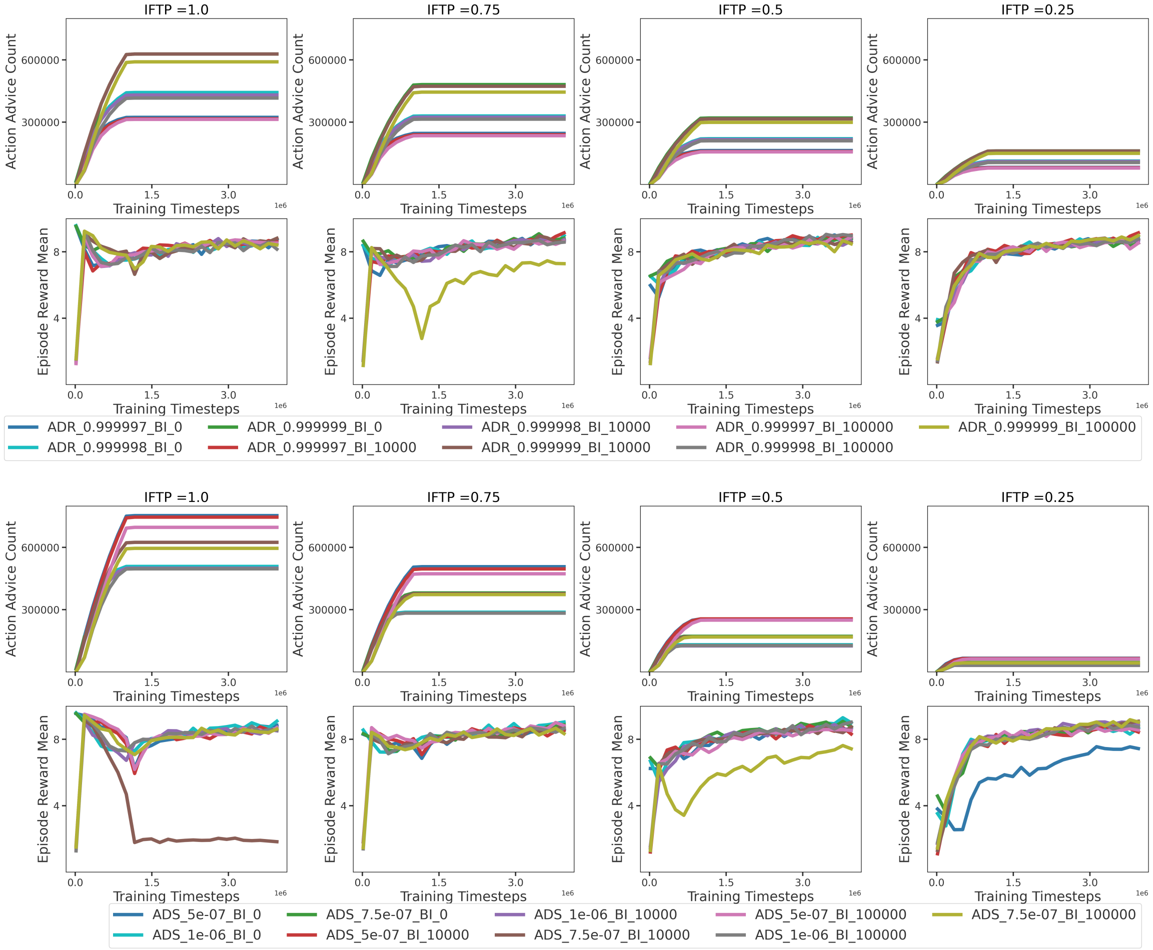


Figure 5.9: Hyper-parameters

until there is no advice at all given to the student. ADR is the decay rate γ where $Prob_{T+1} = \gamma * Prob_T$, where T is the time-steps of a training batch.

4. **Advise Decay Rate (ADS)** Similar as ADR, ADS is the decay step σ where $Prob_{T+1} = Prob_T - \sigma$, where T is the time-steps of a training batch.

Overall, the freedom that the student initially explores the environment, the amount of data the teacher collects before its fine-tuning, and the amount of advice issued empirically affect the performance of ART the most. Figure 5.9 shows an example of the teacher advising the student with the reward of removing rubble of 0.9 in the 10 rubble USAR scenario that we have shown in the experiment section. It can be inferred that with a smaller IFTP, longer BI, bigger ADR and bigger ADS, the advice given to the student is less, and thus the student is less affected by the teacher's advice. It may also allow the teacher to be more accurate when estimating the reward in the student's environment as it collects more data based on the student's exploration. However, the downside of advising less is that if the teacher's policy is beneficial, the student's own exploration maybe unnecessary, especially in this example where the teacher's advice on removing all the rubble aligns a lot with the student's reward.

As shown in Figure 5.9, from left to right, the student's performance with ART becomes more and more similar to the regular RL training where there is no advice. It is common in action advising that advising too much (left-most columns) may limit the student's exploration and when advising stops, the student's performance drops a lot and takes time to learn from the unexplored space. This issue empirically may be fixed with a smaller ADS or ADR, and a lower IFTP, depending on the specific tasks. Additionally, certain combinations of these important hyper-parameters may not work, as some outlier exist that limit the student's learning. We avoid the combinations that produce outliers when selecting important hyper-parameters.

The important hyper-parameters we used in the experiment section are listed as in the following table, and is shown in Figure 5.10 over timesteps. When the advising probability is reduced close to 0.2, stopping advising empirically does not cause the student performance to drop since the student has already learnt, therefore in the figure sharp drops to zero can be observed.

Environment	IFTP	BI	ADR	ADS
USAR	0.9	100000	0.9999985	N/A
Ski	0.9	0	0.999999	N/A
Pacman	0.8	0	N/A	0.000015

Table 5.1: Important Hyper-parameters

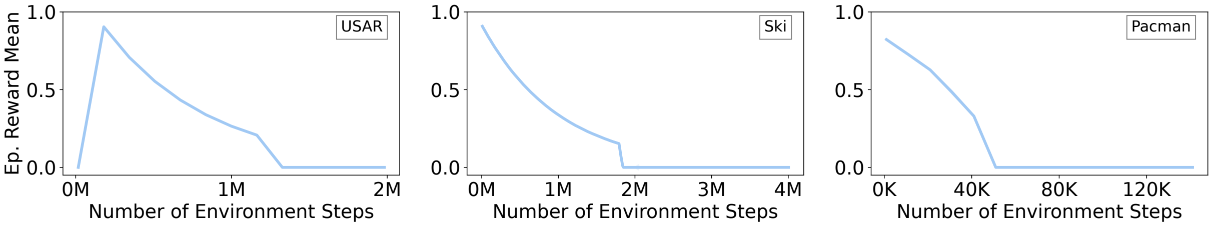


Figure 5.10: The advising probability of each environment.

There are other hyper-parameters such as when the teacher stops training, i.e. teacher training stop timestep (TTST), the teacher training frequency, i.e. inner loop teacher frequency (ILTF), the teacher training iterations (TTI), and basic RL training hyper-parameters for both teacher and student. All the experiments use the algorithm of PPO and the framework of Torch in RLlib. We report the teacher's training details in the following table, and the remaining could be identified in our code, which will be released upon acceptance. The other reported hyper-parameters are common RL training hyper-parameters: learning rate (LR), vf clip param (VFCP), vf loss coefficient (VFLC), kl coefficient (KLC), entropy coefficient (EC), train batch size (TBS), stochastic gradient descent mini-batch size (SGDS), and number of stochastic gradient descent iteration (NSGDI).

Env	TTST	ILTF	TTI	LR	VFCP	VFLC	KLC	EC	TBS	SDGS	NSGDI
USAR	100000	5000	150	0.002	10	0.5	0.5	0.01	8192	256	4
Ski	100000	5000	500	0.001	10	0.5	0.5	0.01	4196	256	4
Pacman	10000	500	150	0.001	10	0.5	0.5	0.012	1000	100	10
USAR10R	100000	5000	150	0.002	10	0.5	0.5	0.01	8192	256	4

Table 5.2: RL Training Hyper-parameters

5.4.2 Reward Similarity

Given two reinforcement learning tasks with the same state-action space, we quantify the average reward difference across all state-action pairs. For each state-action pair (s, a), let $D(s, a) = |R_1(s, a) - R_2(s, a)|$ denote the absolute difference in rewards between the two tasks. The average reward difference D is defined as:

$$D = \frac{\sum_{(s,a)\in S\cdot A} D(s, a)}{|S \cdot A|}$$

This formula calculates D by summing the reward differences for all state-action pairs and dividing by the total number of pairs, yielding an average of the reward discrepancies.

To facilitate a normalized comparison of reward differences across various environments, we define D_{norm} as:

$$D_{\text{norm}} = \frac{D}{\max_{(s,a)\in S\cdot A} D(s, a)}$$

Here, $\max D$ is the maximum possible average difference, which is determined by the structure of the tasks' rewards. Normalizing D to D_{norm} scales the average difference to a range between 0 and 1.

For USAR, each room is composed of 3 by 3 grids, the connection between rooms take 1 grid, rubble, victim, and the agent all occupy 1 grid. Moving to the wall grid, removing non-rubble grid, and healing non-victim grid are considered invalid. For each time the environment is reset, rubble, victim, and the agent are all relocated randomly. Therefore, $D_{(USAR)} = 0.5/422 = 0.0011848$ and $D_{\text{norm}_{(USAR)}} = 1/422 = 0.0023697$.

For Ski, there are 8 gates total, and the student received 10X reward for passing the gate compared to the teacher's +10000. The agent for each time step may choose left or right actions till the borderlines of 10 steps horizontally, and vertically it takes 600 steps. Therefore, $D_{(Ski)} = 720000/10800 = 66.67$ and $D_{\text{norm}_{(Ski)}} = 8/10800 = 0.000741$.

For Pacman, the teacher is an aggressive ghost eater, which means eating ghosts with a reward of 5000 may compensate other potential negative effects such as not finishing eating up foods or eaten by the ghosts when timer counts down to zero. There are 280 non-wall grids. The pacman may move only to non-wall grids, and can eat ghosts when has eaten one of the power pellets within 50 steps. We calculate with a probabilistic approach for this complex dynamics, where the teacher eats 6.4 ghosts on average. This leads to $D_{(Pacman)} = 5000 * 6.4/280 = 114$ and $D_{\text{norm}_{(Pacman)}} = 6.4/280 = 0.02286$.

Generally, the Pacman environment has a bigger reward difference compared to the other two, where critical objects such as rubble or gates are static, while eating ghosts can happen

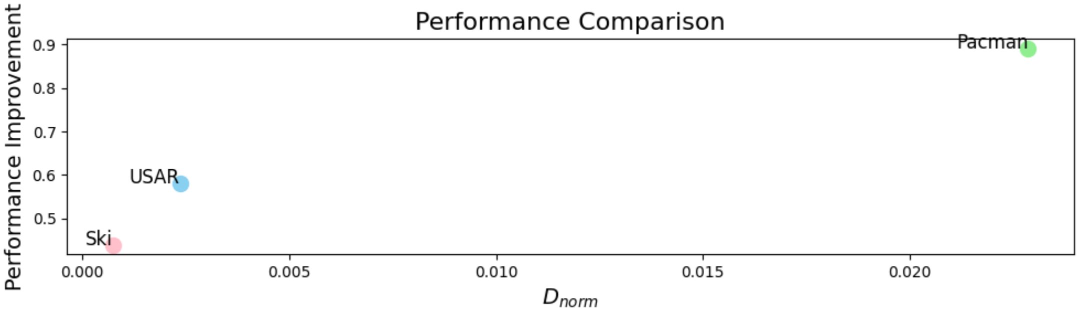


Figure 5.11: Improvement over the reward difference.

anywhere in the maze. In order to further quantify the influence, we calculate the performance difference between the teacher and the trained student with ART, when evaluated in the student's environment.

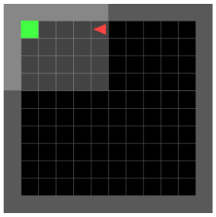
$$\text{Performance Improvement} = \frac{\text{perf}_{\text{ART}} - \text{perf}_{\text{teacher}}}{\text{perf}_{\text{teacher}}}$$

Figure 5.11 shows the result where the performance improvement is over the normalized reward difference. The Pacman who has the highest reward difference also yields a higher performance improvement. This means our ART is more helpful when the reward difference is bigger, and may also explain the small difference between ART and the finetune methods in Ski and USAR, because in these two scenarios the reward difference is smaller, and there is limited room to for the teacher to improve and provide the student with better advice. However, we also don't anticipate the performance improvement keeps growing when the reward difference becomes bigger. We acknowledge our study majorly focuses on the small D_{norm} space. The boundary exploration will be saved for future work.

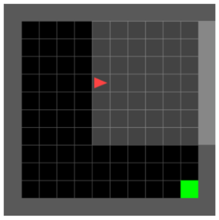
5.4.3 Bidirectional Navigation

We further present the results of an additional environment. In a grid environment, a square room is partitioned into 10x10 grids, serving for a navigation learning task. The source task involves a teacher agent navigating towards the top-left corner of the grid room. Conversely, the target task requires the student agent to reach an opposite goal, the bottom-right corner. The agent is initialized randomly in the room whenever the environment is reset to resample. This setup highlights a challenge when the source and target tasks have similar forms of reward functions that reaching the goal with a positive reward, but the ideal behaviors are completely opposite. Figure 5.12 shows the environment, where the green square is the target for the agent to navigate to, and the red triangle is the agent, whose angle points to the direction facing with. For both environments, the only positive reward of 1 is given when the agent reaches the target, discounted over steps.

The teacher agent excels within its native environment, requiring 15 steps on average to reach its goal, achieving a 100% success rate and a discounted reward of 0.89 over 100 trials. This performance contrasts starkly with its performance in the student's environment when not adapted, leading to an average of 85 steps to reach the target and a reduced average reward of 0.14. This



Teacher's Env



Student's Env

Figure 5.12: Environments of the teacher and student in the bidirectional navigation task.

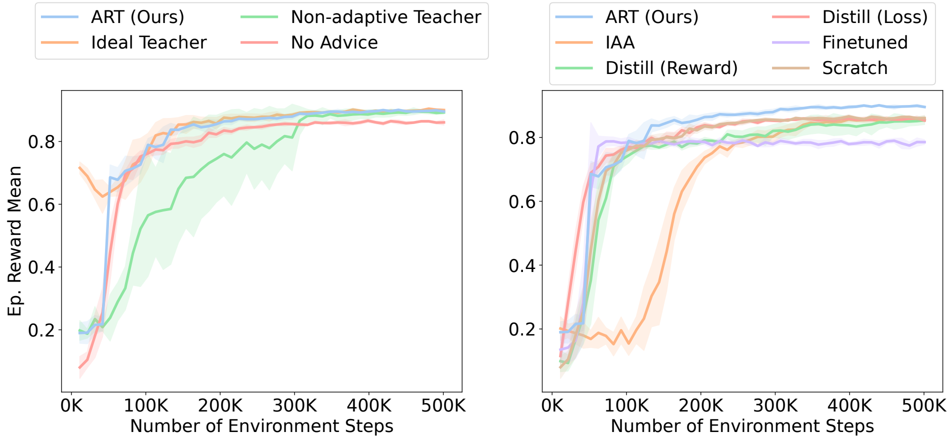


Figure 5.13: Baseline and benchmark results of the bidirectional navigation task

elucidates the challenges as the ideal behaviors are completely opposite. Even though the goal is on the bottom right corner, the non-adaptive teacher still keeps navigating towards its previous goal on the top left corner.

Similar to what we have done for the other three environments of USAR, Ski, and Pacman, we plot the baseline and benchmark results of this task. The baseline of the result in shown in Figure 5.13 (left). As the teacher's advice is for navigating to the opposite direction of the student's target, the teacher's advice can be harmful to the student's training. Therefore, the student advised by the non-adaptive teacher is worse than no advice for a long time, until finally the teacher stops advising and the student is trained by its own to exceed no-advice. Notably, the teacher's advice is not completely useless, as during the beginning of the student's training, student advised by the teacher who has not improved is still better than the student exploring with no advice. Additionally, student advised by any type of the teacher finally is slightly better than the student with no advice. For our ART method, the teacher quickly finetunes its policy to reach the performance of the ideal teacher, and then avoids giving the misleading advice anymore. Therefore, the student with ART is always better than the student with no advice over training.

The benchmark result is shown in Figure 5.13 (right). The difficulty of transfer can be further assured with the method of simple finetune, as the agent with a policy network that leads to the opposite behavior can be hardly changed. Thus the performance of the finetune method is the worst. Other benchmark methods may avoid this issue, however, our ART method has the best performance. This shows our ART is more robust in this scenario where teacher and student have opposite rewards that lead to opposite ideal behaviors.

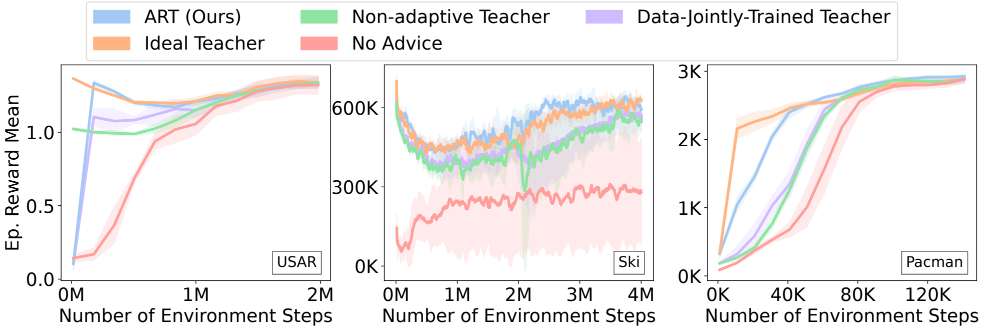


Figure 5.14: Compare with teacher using both its data and the collected student data to train its reward network.

5.4.4 Joint Data Training

We further conduct an experiment that trains a reward model jointly on teacher-collected data and student-collected data. For this data-jointly trained teacher, no error prediction model is used. We plot the results over the Figure 5.14. As shown in the figure, in all three environments, the new setting only allows the student to be trained slightly better than being advised by a non-adaptive teacher.

This is because the newly collected student data can hardly override the teacher data. To be more specific, in the USAR scenario, the teacher receives 0 as its reward when removing rubble, but when it collects student data, this becomes 0.5 instead. In the Ski scenario, the reward is 10x bigger when collected by the student, but from what the teacher has previously, it is not that significant. In the Pacman scenario, the reward of eating ghost as received by the teacher is much bigger than with the student. With data jointly trained, the reward network can hardly discern that is expected by the student, and handle the conflicts of reward values. Therefore, the teacher is only slightly improved, as the reward it estimates is still close to its own reward. This is no doubt worse than our method of ART where the estimation of the student reward is more accurate.

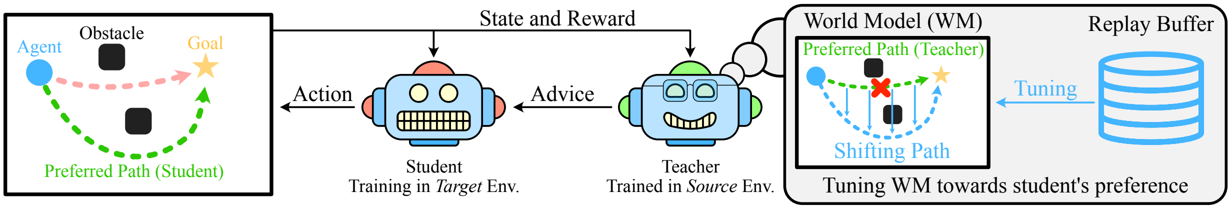


Figure 5.15: An overview of the Reward-Adaptive Teacher with the predicted transition function. The teacher initially issues action advice to the student from a policy trained in the source task – leading to low rewards in the target task due to differences in the reward functions. After observing the student's trajectory returns, the teacher adapts its policy using its internal world model so as to maximize the student's reward in the target task without requiring direct access to the environment.

5.5 Extending with Predicting Transition Function

5.5.1 Algorithm

Algorithm 7 Reward Adaptive Advisor

```
Input: Trained teacher policy $\pi^T$, advising strategy $H$, adaptation period $\mu$.

Prior to student training
Teacher collects $\mathcal{D}_T = \{(s_t, a_t, r_t)\}_{t=1}^{\omega}$ in its environment.
Teacher trains reward model $\mathcal{M}(\mathcal{D}_T)$ and world model $\mathcal{N}(\mathcal{D}_T)$.

During student training
if iteration $\leq \mu$ then
    Teacher collects $\beta$ student trajectories $\mathcal{D}_S = \{(s_t, a_t, r_t)\}_{t=1}^{\beta}$
    Reward adaptation: $\mathcal{M} = \mathcal{M}(\mathcal{D}_S)$
    Policy adaptation: $\pi^T = \text{argmax}_\pi \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{M}(s_t, a_t)\right]$ where $s_{t+1} = \mathcal{N}(s_t, a_t)$
end if
Teacher issues advice $a_t^T \sim \pi^T(a_t|s_t)$ if $H(s_t) = 1$
```

So far we have seen the teacher finetuning its policy with the ground-truth transition function. Intuitively, our approach in this section can be understood as the teacher observing the student's returns in the target task, estimating how they differ from those in the source task in which it was trained, and then using an internal model of the environment dynamics to tailor its policy to the target task *while advising the student*, without ever directly interacting with the target environment.

Learning Environment and Reward Dynamics: Prior to advising a student in the target task, the teacher learns a world model $\mathcal{N}$ of the environment dynamics in the source task. Under the assumption that the state-transition function remains constant between the source and target tasks, this model can be used to perform reinforcement learning without direct access to the actual environment [59, 60]. In parallel, the teacher learns an approximation of the source task's reward

Algorithm 8 World Model Rollout

```
Input: Reward model $\mathcal{M}$, world model $\mathcal{N}$, Policy $\pi$, Current state $s_t$, Max Episode Step $T$.
$s_t = s_0, t = 0$
for $t < T$ do
    $a_t = \pi(s_t)$, $r_t = \mathcal{M}(s_t, a_t)$, $s_{t+1} = \mathcal{N}(s_t, a_t)$
    if $s_{t+1} \notin S_{end}$ then
        $t = t + 1$
    end if
end for
Output: $s_{1 \dots T}, a_{1 \dots T}, r_{1 \dots T}$
```

function $\mathcal{M}$. For efficiency, we learn both of these models jointly with the teacher's initial policy in the source task, although there is no requirement to do so. The algorithm is shown in Algo 7.

Similar as previous, $\mathcal{D}_T = (s_t, a_t, r_t)_{t=1}^{\omega}$, where ω represents the total number of time steps in the teacher's training phase, is utilized to train two models representing its knowledge of the environment dynamics: a reward prediction model, which is a neural network $\mathcal{M}$ parameterized by ϕ, and a world model $\mathcal{N}$ parameterized by θ. $\mathcal{N}$ takes the state s_t and the action a_t at time step t as input, and then outputs the next state s_{t+1}: $\mathcal{N}(s_t, a_t; \theta) \to s_{t+1}$. Typically a world model does not include an estimate of the reward as recent works mostly focus on simulating the transition functions or modeling dynamics, however, we introduce such a mechanism in RAA so that it may be later adapted to the target task. $\mathcal{M}$ aims to estimate the reward r given a state-action pair (s, a): $\mathcal{M}(s, a; \phi) \to r$. Thus same as previous, the training objective for $\mathcal{M}$ is to minimize the difference between the predicted reward and the actual reward:

$$L(\phi) = \mathbb{E}\left[||\mathcal{M}(s_t, a_t; \phi) - r_t||^2\right], \tag{5.4}$$

The transition network $\mathcal{N}(s_t, a_t; \theta) \to s_{t+1}$ uses the loss function minimizing the difference between the network's predicted next state and the actual next state experienced in the environment:

$$L(\theta) = \mathbb{E}\left[||\mathcal{N}(s_t, a_t; \theta) - s_{t+1}||^2\right] \tag{5.5}$$

Policy Adaptation: After adjusting its reward prediction model, the teacher uses the updated $\mathcal{M}$, and its world model $\mathcal{N}$ to perform a limited number of reinforcement learning iterations and improve its policy π^T with respect to the target task, by modifying the parameters of π^T based on the standard update rule: $\theta' = \theta + \alpha \nabla J'(\theta)$, where $J'(\theta)$ maximizes the expected cumulative reward under the new reward distribution:

$$J'(\theta) = \mathbb{E}\pi^T \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{M}'(s_t, a_t; \phi) \middle| s_t, a_t \right], \text{where } s_{t+1} = \mathcal{N}(s_t, a_t). \tag{5.6}$$

This adaptation process, shown in Alg. 8, allows the teacher to improve its policy without requiring direct access to the target environment and may be performed numerous times throughout the student's training process. The updated policy is subsequently used to issue future action advice to the student according to an advising heuristic $H(\cdot)$ – our approach is agnostic of the specific heuristic which is used. The effectiveness of this adaptation step is contingent on how accurate the reward model is. As a result, adaptation steps performed early in the interaction when a limited number of student returns have been observed are likely to be more noisy than those performed later in training.

5.5.2 Experiments and Results

We empirically evaluate our proposed adaptive teacher with world model in two separate scenarios: a simulated urban search and rescue environment where the agent needs to remove rubble and rescue trapped victims; and a navigation task in which a mobile robot must reach a goal while avoiding obstacles, which we have implemented in both simulation and the real-world. These domains have been selected such that we can determine: a) whether our reward-adaptive advisor's advice adapts to the reward function associated with the target task, b) the degree to which this adaptation improves over prior state-of-the-art methods, and c) if this adaptation can be achieved in real-world environments.

5.5.3 Environments

USAR: The urban search and rescue environment [42, 86] is a challenging environment which enforces strict cooperation [53, 54] among agents with heterogeneous action spaces. We consider a combined agent role of medic, which can rescue victims, and engineer, which can remove rubble, who must cooperate such that victims which are trapped behind rubble can be rescued. In addition to a positive reward for rescuing victims, the source task rewards removing only pieces of rubble which block access to victims, while the target task rewards removing *all* pieces of rubble. Both the teacher and student policies are trained with DQN [109] and discrete state and action spaces. We evaluate over two variants: a simple 4-room layout and a more complex 14-room layout, as developed in [54].

Navigation to Goal: This environment is a navigation task in which a differential drive robot must navigate from a start location to a goal location while avoiding collisions with static obstacles. While the agent is rewarded for reaching the goal in both the source and target tasks while avoiding collisions, the reward function is modified in the target environment to penalize the agent if it gets too close to obstacles. In the source task, the penalty is low and so the agent's optimal behavior is to navigate directly to the goal by traveling between the obstacles. In the target task, the penalty is increased and the agent is encouraged to drive around the outside of the obstacles to reach the goal. This environment is implemented in both simulation [124] and in the real-world on the Khepera [76] robot platform; all policies are trained using PPO [129] with a continuous state space and discrete action space.

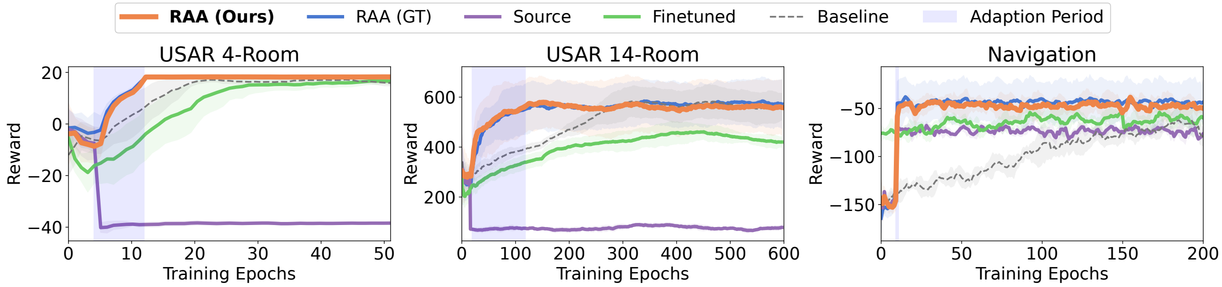


Figure 5.16: RAA (in orange) compared to ablated methods. The shaded region denotes the adaptation period, which happens every 2 iterations in USAR-4Room, 5 in USAR 14-Room, and only once in Navigation. No advising occurs prior to the first β episodes.

5.5.4 Results and Analysis

Teacher Performance: We first investigate RAA's ability to adapt the teacher policy to the target task, irrespective of a student. Figure 5.16 shows the performance of RAA in the target task along with several ablated methods. The key outcome from this analysis is that RAA's adaption process produces a teacher policy which achieves a higher reward in the target task than the other methods, substantially faster – with a *minimum* improvement to convergence rate of 2x (USAR 4-Room). The original teacher policy lacking any adaptation (Source) yields significantly lower returns across all three scenarios, while training a teacher from scratch (Baseline) or improving the existing policy (Finetune) requires more than double the amount of time to learn an optimal policy compared to our proposed adaptation process. Informally, RAA produces a better teacher policy, faster.

In addition, we measured whether using a learned world model to adapt the teacher's policy to the target task led to any performance degradation due to error accumulation. Adaptation using the ground truth dynamics model and target task reward function, RAA (GT), achieves approximately the same rewards as RAA, however. This indicates that despite a maximum episode length of 100 timesteps, policy adaptation via our learned world model is an effective strategy.

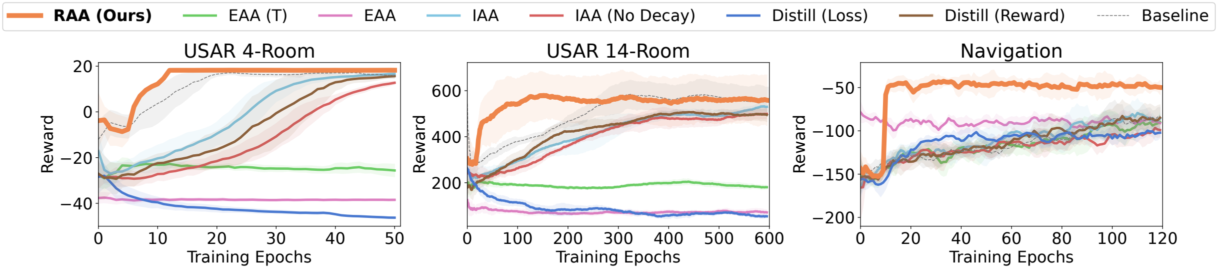


Figure 5.17: RAA (in orange) compared to state-of-the-art transfer learning approaches.

Comparison to Transfer Learning Methods: Next, we compare RAA to state-of-the-art transfer learning methods to determine whether our adaptive teacher policy is capable of issuing high-quality advice. The results, shown in Fig. 5.17, indicate RAA's teacher policy performance exceeds that of the compared methods for a significant duration of the training period. This suggests that the teacher is capable of issuing action advice leading which leads to improved performance; in the extreme case, if RAA were to issue advice at every timestep and the student were to blindly follow it, then the student would achieve higher rewards at least 5x faster than if it were trained according to another transfer learning approach. We note that the other transfer learning methods failed to meaningfully improve upon Baseline performance at all, indicating that a student trained from scratch (Baseline) is actively misguided by other approaches yet would benefit from RAA in our scenarios.

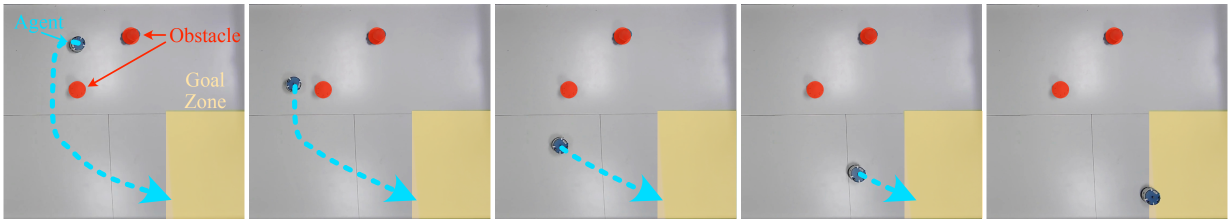


Figure 5.18: Rollout of an RAA-adapted policy given real-world student trajectories.

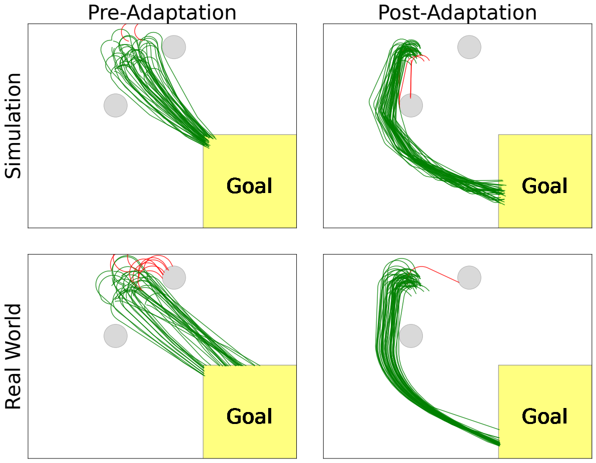


Figure 5.19: Policy adaptation results in simulation and real-world across 50 trajectories respectively. A consistent initialization of Pre-Adaptation and Post-Adaptation is kept.

Real-world Evaluation: We evaluate RAA's ability to adapt to real-world conditions by performing student training in a real-world equivalent of our navigation scenario. Using a Khepera differential drive robot as our embodied agent, we begin student training with PPO and perform RAA's adaptation step using a set of 150 real-world trajectories consisting of student interactions. The real-world trajectories are shown in 5.18. This experiment is designed to test, a) whether policy adaptation using a learned world model succeeds if the reward network has been fine-tuned using real-world data, and b) whether the resulting trajectories produce successful student trajectories despite the sim-to-real gap. Figure 5.19 (bottom row) shows a set of 50 policy rollouts using the adapted RAA teacher policy (right column) in comparison to the original teacher policy (left column). We observe that not only do the adapted trajectories exhibit the optimal behavior for the target task, but that they closely match the adapted policy produced purely in simulation (top row).

Adaptation Process:

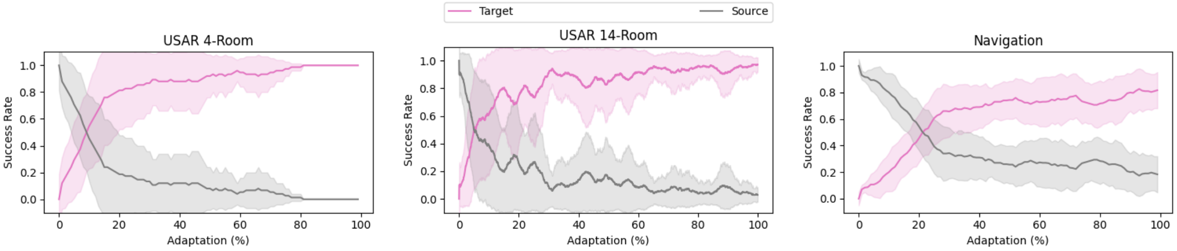


Figure 5.20: Throughout the teacher's fine-tuning process, its success rate in its source environment (gray) gradually decreases while the one in the target environment (pink), which reflects student's rewards, grows. This shows the teacher gradually learn to adapt to student's reward and is able to make alternative advice based on student's need.

The effect of the adaptation process on the teacher policy is visualized in Fig. 5.20. In all of our evaluated scenarios, RAA's adaptation step modifies the teacher's policy such that it gradually results in successful episodes with respect to the target task, at the expense of success with respect to the source task – under a mutually exclusive success definition.

5.6 Conclusion

In this chapter, we have introduced Adaptive Reward Teaching – an action advising method which produces a teacher capable of adapting its advice to a target task in the face of changing rewards. We empirically show that our approach results in significant policy improvement with respect to the target task's reward function in a far more sample efficient manner when compared to a number of state-of-the-art transfer learning approaches. We have also extended by calculating the transition functions and estimating the teacher's boundaries over the world model. In the future, we plan to further explore the adaptability of the teacher and quantify the distribution shift of the student's data from teacher's.

Chapter 6

Human Acceptance to Advice and Explanations from Artificial Teacher

So far, we have presented the work where the student as an artificial agent gets advice from the teacher, and the performance is improved. As we are motivated from the cognitive literature, that providing explanations and internal models would help human agents learn, we are interested in extending the advising framework to human students. Furthermore, due to the scope of this thesis that majorly focuses on policy transfer, and the complexity of human learning, we examine human's acceptance to the advice and explanations provided by an artificial teacher.

More formally, in this chapter we are aimed at using reinforcement learning (RL) policy to advise human players. This policy is distilled into a decision tree (DT) format to facilitate interpretable advice, particularly in explaining actions via DT path features. This process of providing explanations is the same as what we have presented in the chapter of providing explanations (Chapter 2), and modeling student's internal models. The selected environment for our application is the Pacman game, where the primary objective is to maximize food consumption. Our aim is to let the artificial teacher give advice and its explanations to human players, when the teacher is fed with the observation from a player. The IRB protocol is Pitt STUDY21030223, Crowd-sourced Behavioral Studies for Human-Agent/Robot Interaction.

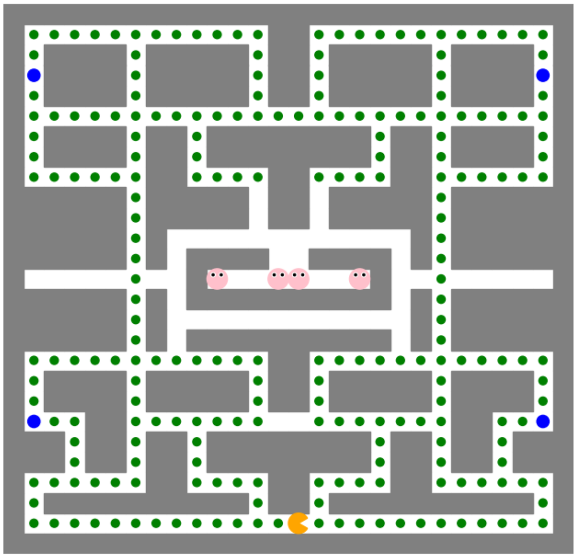


Figure 6.1: The start of pacman game.

Game Rules The start of the game is shown in Figure 6.1. The Pacman game is a classic video game that has been widely popular. In the game with the classical layout, players controls the Pacman (in orange) to navigate in a maze. The mouth of the pacman shows the current direction that the pacman is faced with. The maze is filled with dots (in green), known as foods, which Pacman can eat to gain points. The maze also contains four ghosts (in pink) that chase Pacman and attempt to catch it. The player's goal is to consume all the foods in the maze while avoiding being caught by the ghosts.

To aid Pacman in evading the ghosts, there are power pellets, i.e. capsules, (in blue) located in the maze. When Pacman consumes a power pellet, the ghosts temporarily become vulnerable and can be eaten by Pacman, providing a momentary respite from the chase. The game may progress in difficulty as the player advances, with the ghosts becoming faster and the power pellets' effect lasting shorter durations. The default maximum of the timer is 50 timesteps, when the Pacman just eats a capsule. When a ghost is eaten, its timer is set to 0 and sent to the center, i.e., not vulnerable, and starts its chasing.

6.1 Train the Teacher

6.1.1 RL Expert

The RL agent is trained with PPO. It has four actions of moving one grid up, down, left, and right. If the action is invalid, such as towards a wall grid, the agent won't change its location. The policy is based on the extracted features from the observation of each state. The features, represented as binary indicators (0s and 1s), have been refined to capture essential aspects:

- Directions (South, North, East, West) based on their potential to lead to the nearest food, power pellet, or ghost. If there are more than one closest items with the same distance, both corresponding directions are ones. It uses bfs to expand to find those closest items.
- Proximity to ghosts within a radius of 2. If the distance to ghosts is within the radius, the corresponding directions are ones.
- The fear status of the closest ghost (if scared for 10, 20, 30, or 40 timesteps) are calculated. For example, if the scared timer is 25, the first two features are ones and the rest are zeros.
- For each direction, if the next grid is a wall or passable.

Reward Structure The RL agent is expected to eat as many foods as possible, while at the same time strategically eat more power pellets and avoid ghosts. Unless it is encircled, it should keep itself safe while looking for foods. The reward function is designed to shape desired behaviors for the RL agent is as follows:

- Eating a food item: +10
- Consuming a power pellet: +150
- Being caught by a ghost: -500
- Consuming all available food: +1000
- Eating a ghost: -200

The balancing of reward-shaping is challenging, where in this case the agent may have a hard time choosing to prioritize power pellet to keep itself safe, or be risky to enter into regions where there are more foods as well active ghosts. It may also be risky to eat up a ghost, especially when the timer number is still big, as the ghost will be sent back to center and quickly comes towards the agent again. Additionally, when circumvented, such as multiple ghosts come from two ends of the road, and no power pellet remains, there is no way for the agent to escape.

Out of the total 229 foods, 4 power pellets, and maximum of 600 steps, the best agent is trained to reach the following performance, and it becomes our teacher. They are the mean numbers taken from 200 sample trajectories.

Policy	Food Eaten	Power Pellet	Ghost Eaten	Steps	Rewards
RL	194	3.5	0.33	248	2068

Table 6.1: Performance evaluation of the best RL policy.

6.1.2 Representable DT Teacher

Similar as what we have done in Explainable Action Advising, we used VIPER to extract the RL policy into a representable decision tree policy. Therefore, when giving action advice on any observation, the features on the decision tree paths are also given as the explanations leading to the action advice.

To be more specific, in the VIPER process, for each iteration, 20000 steps are sampled and the maximum iteration is 10. To evaluate if the DT policy is good enough, 200 trajectories are generated, and the best DT policy is selected among all the iterations over the 5 experiments.

The learnt DT teacher may eat 95% of the food as its performance compared to the RL teacher. With filtering on teacher's trajectories that have higher rewards, it might eat more food, however, it would lead to a smaller diversity of the DT policy. We don't intend to extend this discussion in this work.

Policy	Food Eaten	Power Pellet	Ghost Eaten	Steps	Rewards
DT	185	3.4	0.3	233	1872

Table 6.2: Performance evaluation of DT policy.

6.1.3 Keyboard Interaction Interface

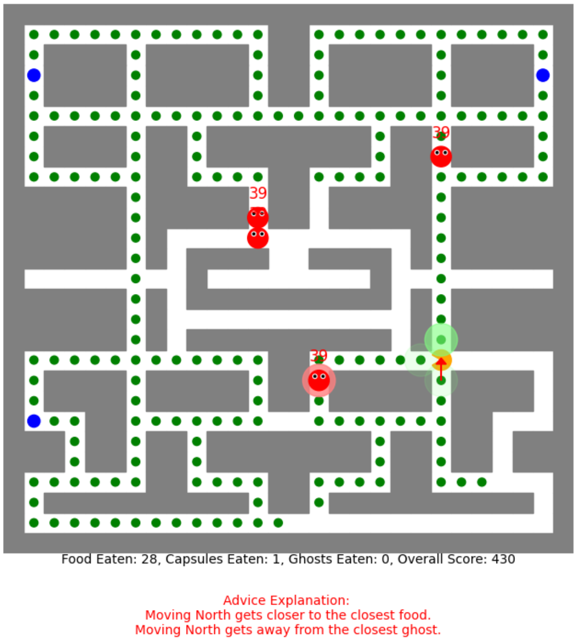
We have developed an interface so that human players may use keyboard arrows (up, down, left, right) to control the Pacman, aligning with the conventional gameplay mechanics. The reward setting also follows the conventional Pacman game, except for not receiving reward for eating the ghosts, as in our game we would like to focus on the goal where finishing eating up all the food is prioritized. The reward metrics for the game are as follows:

- Consuming food: +10 points.
- Consuming a capsule: +150 points.

The game's speed significantly influences its difficulty. Intuitively, if the speed is very fast, there is little time for the player to react, while slowing down the speed can give players enough time to think and respond. Even though enough time of thinking does not guarantee the success of each movement, it would trivialize the game and the advice given to players. Therefore, the speed is set to be 0.5 seconds as one step, which allows the designer, who is an expert playing the game to reach almost as good performance as the teacher.

Furthermore, we would like to emphasize that we don't directly ask participants to play the game with this interface. The human participants are served as observers to the clips of game playing from a human game expert. At each intersection, the human player receives advice from the teacher:

- A red arrow indicates the teacher's suggested direction.



Food Eaten: 28, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 430

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.

Figure 6.2: The advice and explanation received.

- Explanations are provided for the suggested action, focusing on the nearest food, ghost, and capsule.
- There are two types for these explanations:
    1. Red text describing the rationale.
    2. Visual highlights on game elements related to the advice (e.g., closest food, ghost, and capsule).

For example, as shown in Figure 6.2, the player is receiving advice from the teacher at an intersection. The player sees its number of food eaten, capsule eaten, ghost eaten, and the overall score. In this example, a capsule has just been eaten by the pacman, and the pacman has never eaten a ghost, so all of the ghosts are edible, with the timer on there top showing the time left to maintain this edible mode. The red arrow pointing towards north indicates the teacher advising the player to go north. The explanation that leads to the teacher's action advice are given from the teacher's decision tree path. In this example, the features related are getting closer to the closest food and getting away from the closest ghost. Only the closest food in the teacher's direction is highlighted with a colorful green circle, while other closest foods are highlighted with lighter colors.

Filtered Explanations We have filtered the features on the teacher's decision tree path, where only the related closest food, ghost, and capsule corresponding to the teacher's action direction are given to the player.

The features related to if the next step for each action is wall or passable are hidden as they are more straightforward in the game view. The features related to the nearby ghosts are also hidden, because the small radius is only useful for the teacher, while also straightforward in the game view. The features irrelevant to the directions are hidden, i.e. the timer for the closest ghost is not given in the explanation because the timers are shown on top of each edible ghosts.

The remaining features on the decision tree paths are given as explanations, in the same order as in the tree paths.

6.2 Amazon Mechanical Turk (MTurk) Experiment

6.2.1 Design of Experiment

Hypothesis: Explanations increase the acceptance of the teacher interventions, compared to the no-explanation condition.

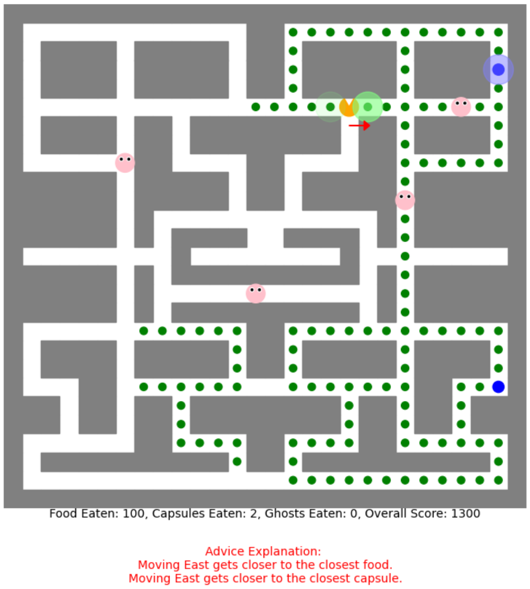
Dataset: Samples (s, a) are taken from an unknown (our expert human) policy. These are advised by the teacher policy and explained using the DT policy.

Design: Participants are divided into three between-subject groups based on the type of explanation they receive: visual-explanation, visual-and-text-explanation, or no-explanation.

Survey Questions: Participants are observers. They are asked the following questions after watching each of the intervention video clip, with the scores scale from 0 (strongly disagree) to 6 (strongly agree):

- The suggested action makes sense to me.
- If I were to play, I would comply with the suggested action.
- The provided explanation makes sense to me.
- The provided explanation helps me understand the suggested action.
- How do you like the suggested action and explanation?

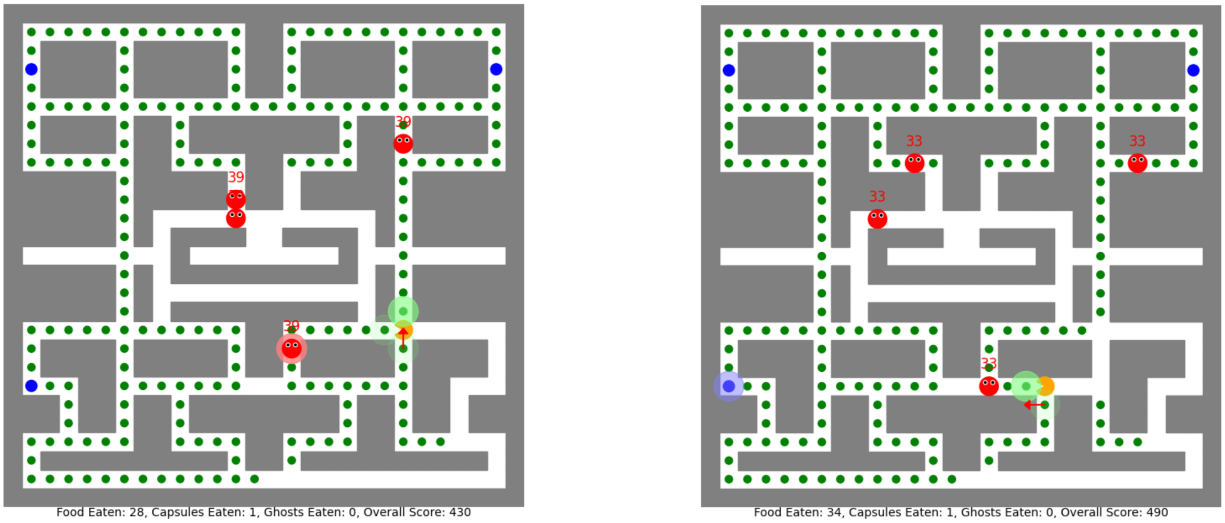
6.2.2 MTurk Samples Generation



Food Eaten: 100, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1300

Advice Explanation:
Moving East gets closer to the closest food.
Moving East gets closer to the closest capsule.

Figure 6.3: MTurk Samples (1)

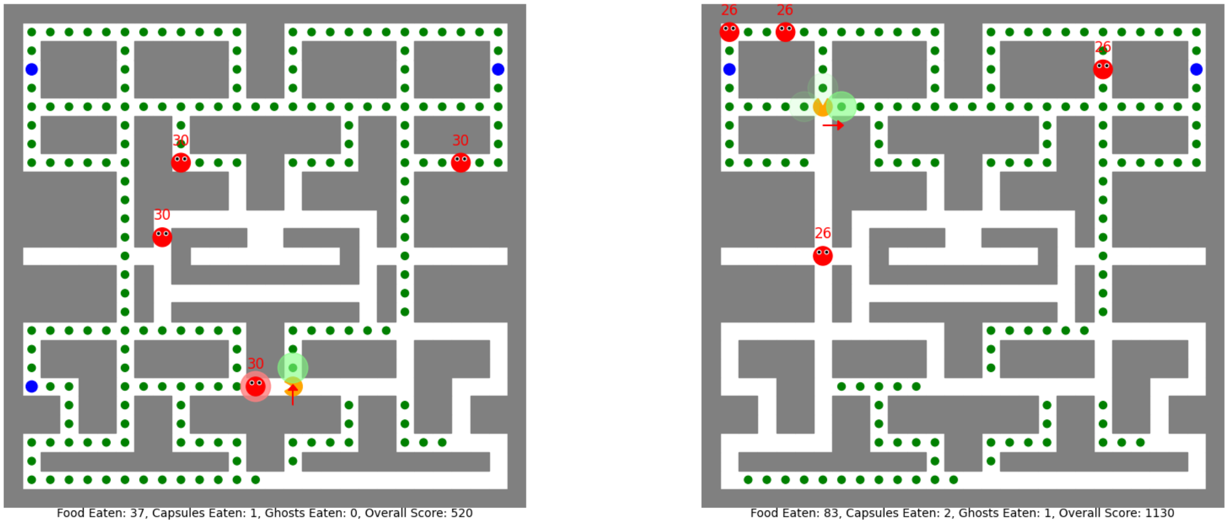


Food Eaten: 28, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 430

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.

Food Eaten: 34, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 490

Advice Explanation:
Moving West gets closer to the closest food.
Moving West gets closer to the closest capsule.

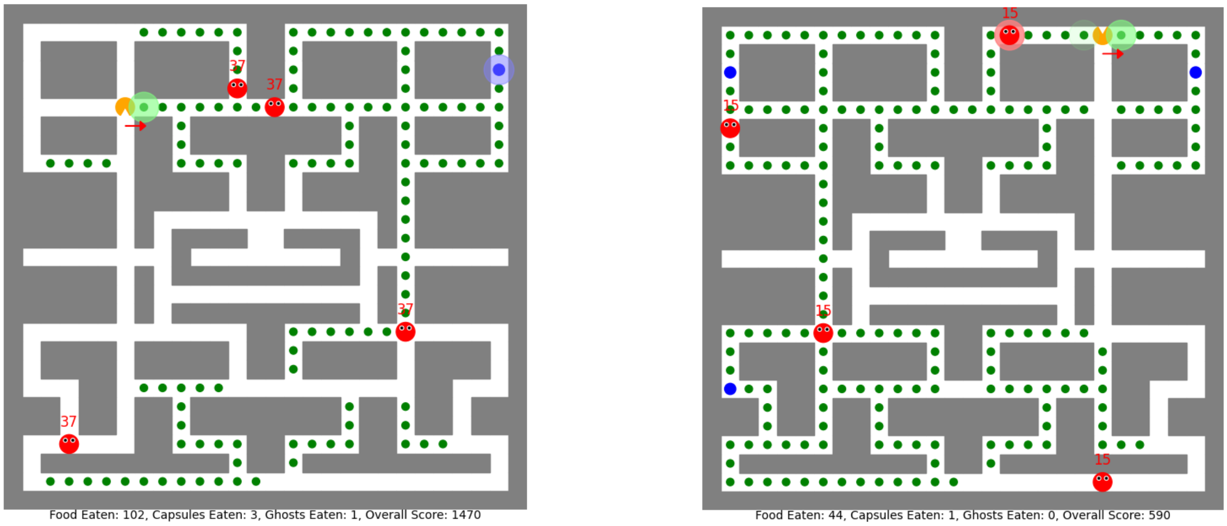


Food Eaten: 37, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 520

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.

Food Eaten: 83, Capsules Eaten: 2, Ghosts Eaten: 1, Overall Score: 1130

Advice Explanation:
Moving East gets closer to the closest food.



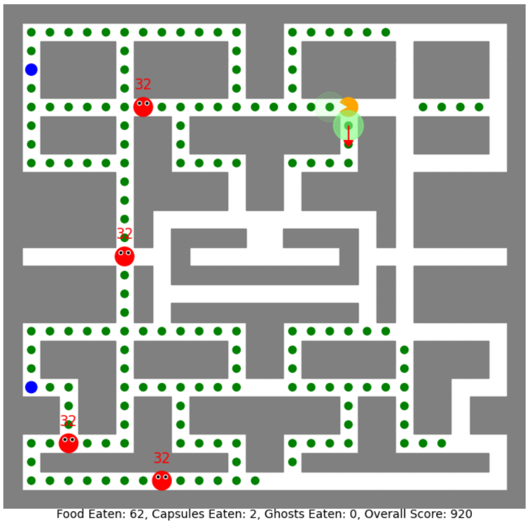
Food Eaten: 102, Capsules Eaten: 3, Ghosts Eaten: 1, Overall Score: 1470

Advice Explanation:
Moving East gets closer to the closest food.
Moving East gets closer to the closest capsule.

Food Eaten: 44, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 590

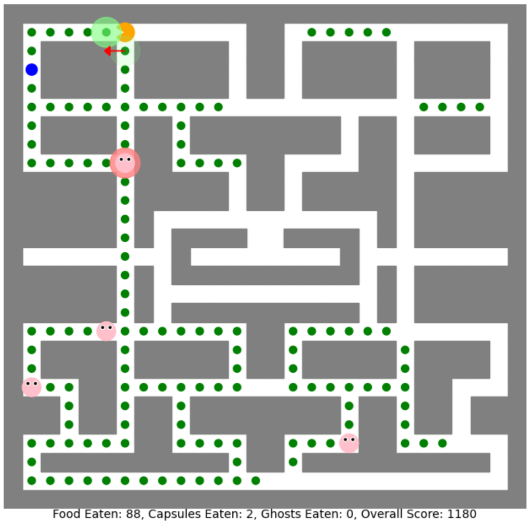
Advice Explanation:
Moving East gets closer to the closest food.
Moving East gets away from the closest ghost.

Figure 6.4: MTurk Samples (2)



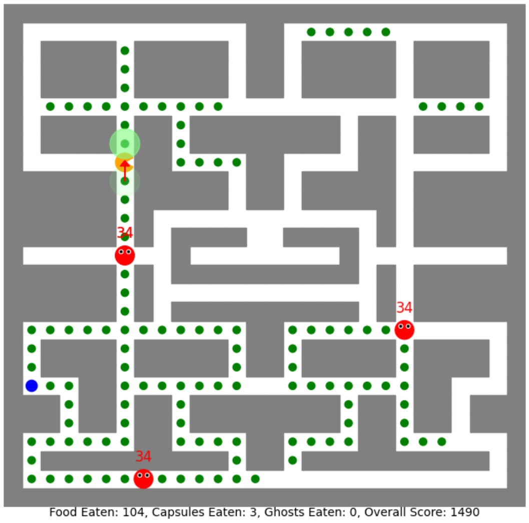
Food Eaten: 62, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 920

Advice Explanation:
Moving South gets closer to the closest food.



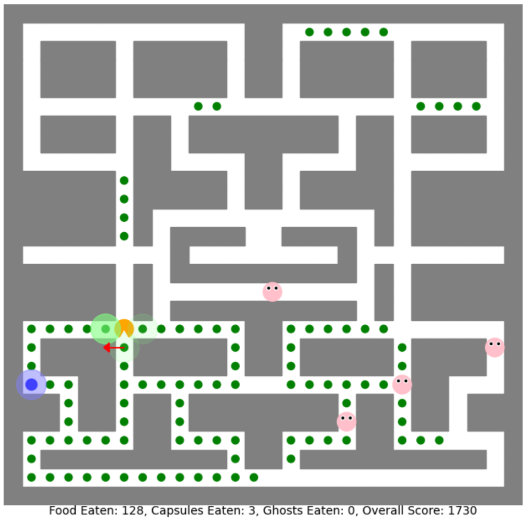
Food Eaten: 88, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1180

Advice Explanation:
Moving West gets closer to the closest food.
Moving West gets away from the closest ghost.



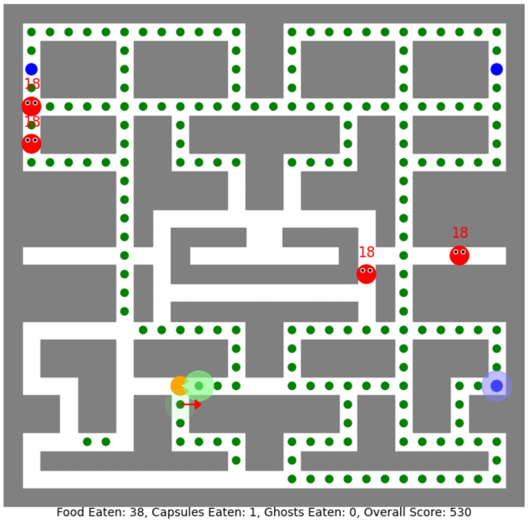
Food Eaten: 104, Capsules Eaten: 3, Ghosts Eaten: 0, Overall Score: 1490

Advice Explanation:
Moving North gets closer to the closest food.



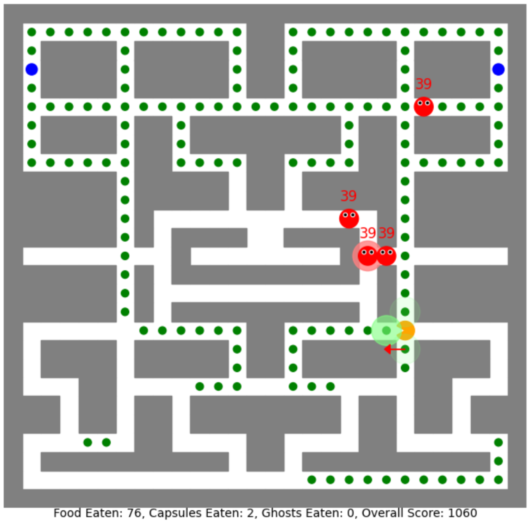
Food Eaten: 128, Capsules Eaten: 3, Ghosts Eaten: 0, Overall Score: 1730

Advice Explanation:
Moving West gets closer to the closest food.
Moving West gets closer to the closest capsule.



Food Eaten: 38, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 530

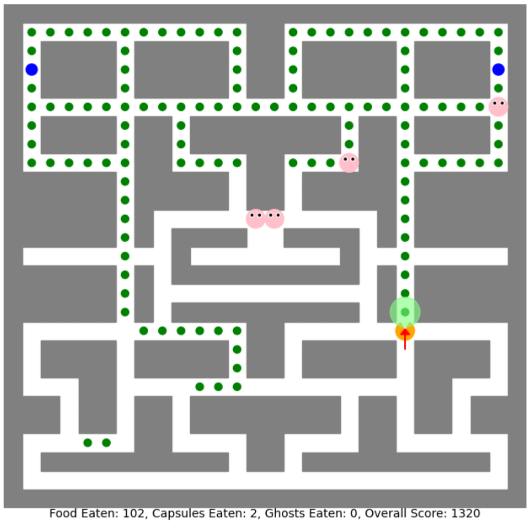
Advice Explanation:
Moving East gets closer to the closest food.
Moving East gets closer to the closest capsule.



Food Eaten: 76, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1060

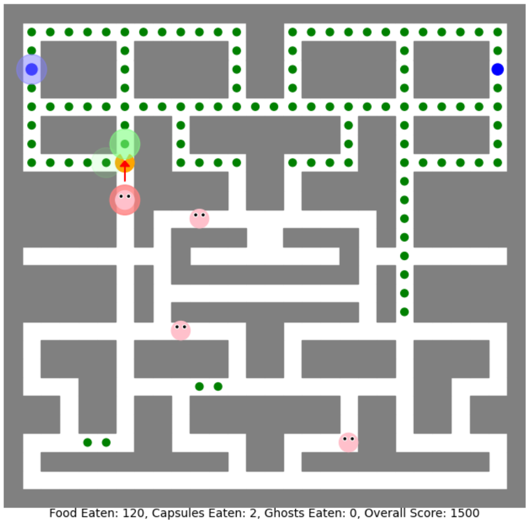
Advice Explanation:
Moving West gets closer to the closest food.
Moving West gets away from the closest ghost.

Figure 6.5: MTurk Samples (3)



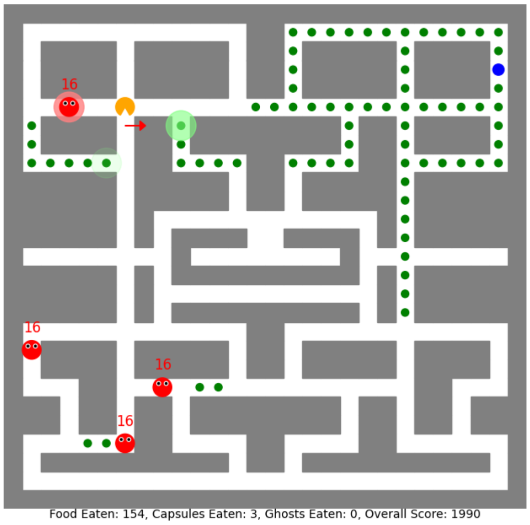
Food Eaten: 102, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1320

Advice Explanation:
Moving North gets closer to the closest food.



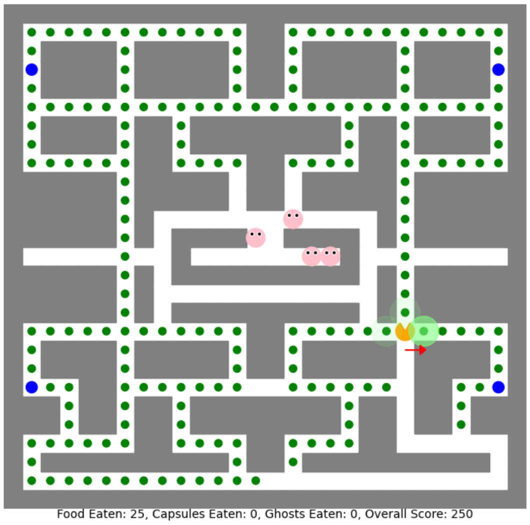
Food Eaten: 120, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1500

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.
Moving North gets closer to the closest capsule.



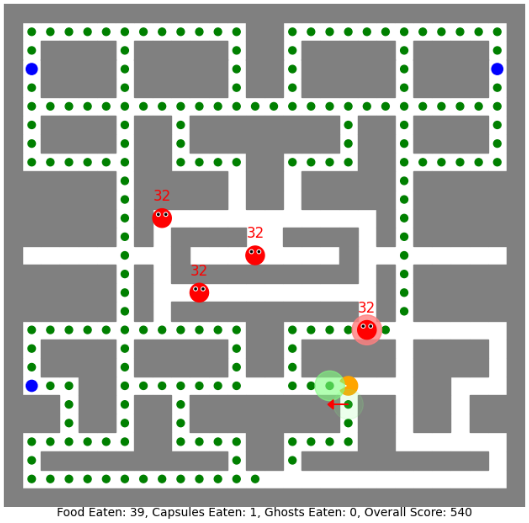
Food Eaten: 154, Capsules Eaten: 3, Ghosts Eaten: 0, Overall Score: 1990

Advice Explanation:
Moving East gets closer to the closest food.
Moving East gets away from the closest ghost.



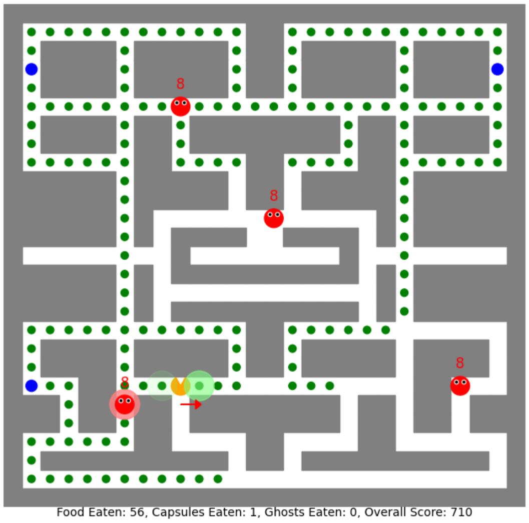
Food Eaten: 25, Capsules Eaten: 0, Ghosts Eaten: 0, Overall Score: 250

Advice Explanation:
Moving East gets closer to the closest food.



Food Eaten: 39, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 540

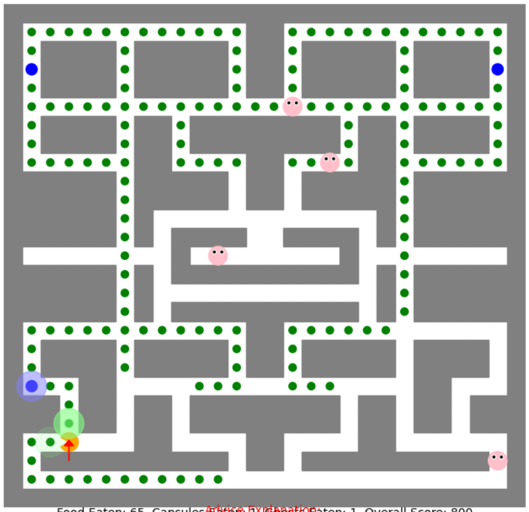
Advice Explanation:
Moving West gets closer to the closest food.
Moving West gets away from the closest ghost.



Food Eaten: 56, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 710

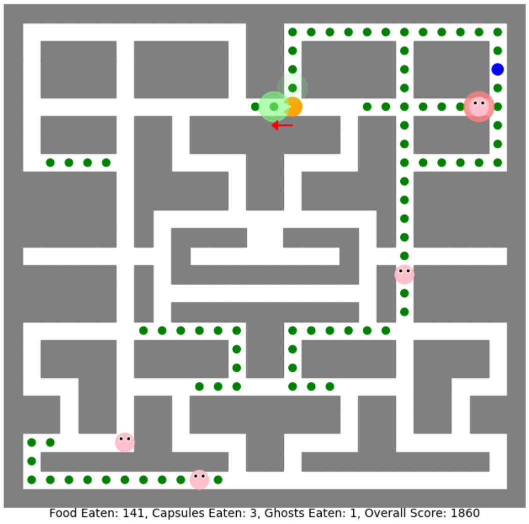
Advice Explanation:
Moving East gets closer to the closest food.
Moving East gets away from the closest ghost.

Figure 6.6: MTurk Samples (4)



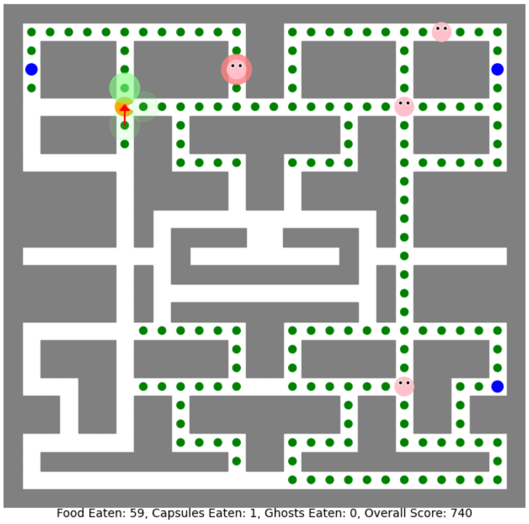
Food Eaten: 65, Capsules Eaten: 1, Ghosts Eaten: 1, Overall Score: 800

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.
Moving North gets closer to the closest capsule.
The closest ghost is not highlighted as it is still far away.



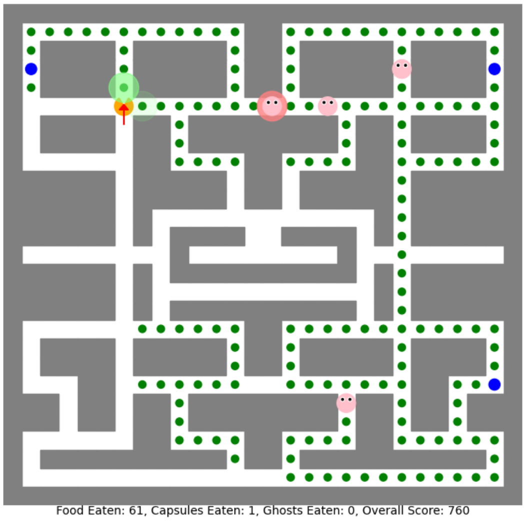
Food Eaten: 141, Capsules Eaten: 3, Ghosts Eaten: 1, Overall Score: 1860

Advice Explanation:
Moving West gets closer to the closest food.
Moving West gets away from the closest ghost.



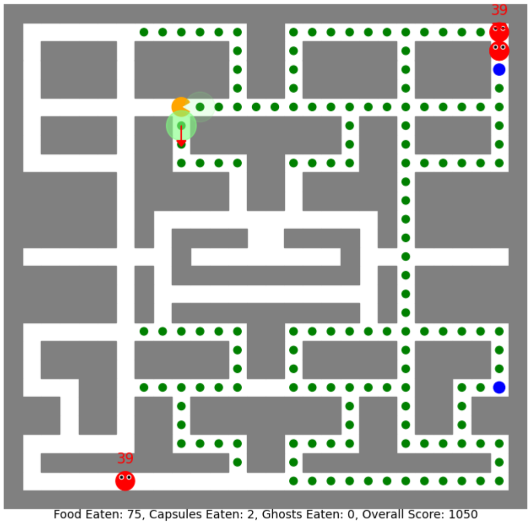
Food Eaten: 59, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 740

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.



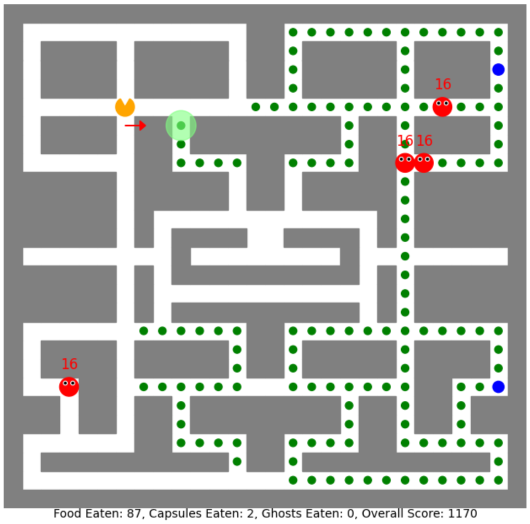
Food Eaten: 61, Capsules Eaten: 1, Ghosts Eaten: 0, Overall Score: 760

Advice Explanation:
Moving North gets closer to the closest food.
Moving North gets away from the closest ghost.



Food Eaten: 75, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1050

Advice Explanation:
Moving South gets closer to the closest food.



Food Eaten: 87, Capsules Eaten: 2, Ghosts Eaten: 0, Overall Score: 1170

Advice Explanation:
Moving East gets closer to the closest food.

Figure 6.7: MTurk Samples (5)

A total of 25 samples have been generated from 5 trials of an expert play. Each sample is a small video clip capturing gameplay from one intersection to the next, concluding with the teacher's advice. To present those clearly in the thesis, we include the last frames of the samples instead of the videos, who contain a few prior steps towards this current frame, as displayed from Figures 6.3, 6.4, 6.5, 6.6, 6.7.

Variations: For each clip, three different versions are prepared:

- no-explanation: The clip includes advice on action without explanations.
- visual-explanation: The advice is paired with visual highlights as explanations.
- visual-and-text-explanation: The advice features both text explanations and visual highlights.

The number of parcipants for each of the variations are also listed as follows:

- no-explanation: 30.
- visual-explanation: 30.
- visual-and-text-explanation: 31.

6.2.3 Experiment Results

With score of 6 as the strongest agree, we have an average score of 5.0 for human participants to accept the advice for all samples. This indicates the teacher's action advice is generally acceptable to participants. Among the 25 samples we have, 21 of them has an average score of 5.4, which we call as the "acceptance" group as they have a high score to accept the advice. However, for the rest 4 samples, they have an average score of 3.1, and they are all below a score of 4. The attitude from the participants are neutral towards the advice, and thus we call them the "neutral" group. We then break down to see the more details of the groups.

Scores breaking-down is shown in Figure 6.8 and Figure 6.9. The left images show average scores of action advice, and the right images show average scores of explanations. We test with three different conditions of no explanations, only visual explanations, and visual as well as text explanations, which are represented in the colorful bars of blue, orange, and green.

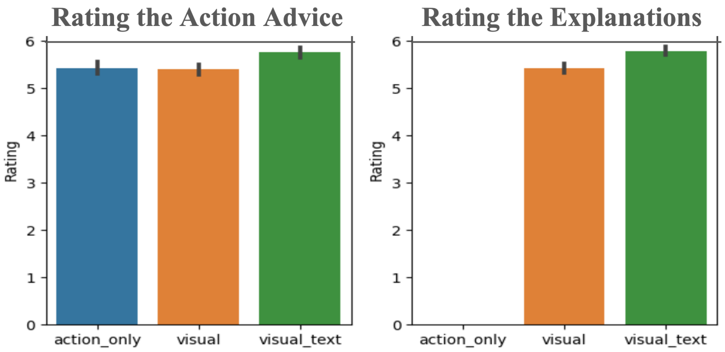


Figure 6.8: Acceptance Group

For the acceptance group in Figure 6.8, it can be clearly inferred that with visual and text explanations, the scores of action advice are higher than the other two conditions, while the other two conditions are similar. The p value is smaller than 0.05 and indicates the advantage is significant. Besides, scores of visual and text explanations are higher than only providing visual explanations. Therefore, we conclude that explanations, especially in the form of visual and text combination, make acceptable action advice more acceptable.

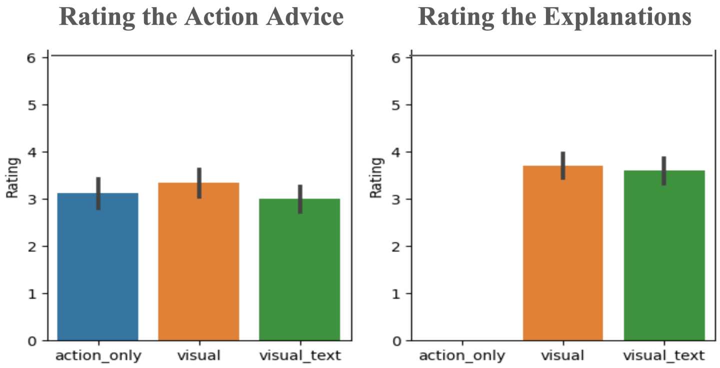


Figure 6.9: Neutral Group

However, if we examine the neutral group shown in Figure 6.9, first we can infer from the right image, that the two forms of explanations do not have much different scores. This means if humans feel neutral about the advice, the form of explanations does not matter. Besides, action advice with explanations do not have significant better scores, indicating the explanations do not help participants accept neutral advice.

6.3 Conclusion

In this chapter, we show how we evaluate human's acceptance when they receive action advice and explanations from an artificial teacher. The advice is generated from a pre-trained reinforcement learning agent, and explanations are from the corresponding decision tree policy, using the similar procedure as in the work of explainable action advising. The human participants are observers to examine various advice and explanations, and we examine if the explanations are helpful. Interestingly, only the explanations with acceptable advice make advice rated higher. The explanations of the advice neutral to human participants are rather less influential. Furthermore, as we only include a simple template for explanations, they are not as human languages. We examine the usage of a large language model to generate explanation in the next chapter.

Chapter 7

Utilizing a Language Model to Improve Explanations to Humans

In the ever-evolving landscape of artificial intelligence, intelligent agents, particularly robots, have transcended experimental settings to make substantial inroads into real-world applications, many of which are safety-critical. The ability of these agents to interact and collaborate effectively with humans hinges on their capacity to explain their decisions and actions in a manner that is comprehensible to their human counterparts. Despite the prowess of deep neural networks in driving the decision-making processes of these agents, the intrinsic opacity of such models poses a significant barrier to interpretability. Addressing this challenge, this chapter pivots on translating the intricate decision-making processes of agents into natural language explanations. By observing only the states and actions of these agents, we propose a method that is agnostic to the model's internal workings, focusing instead on the outcomes of its decisions. This method involves distilling the agent's policy into a decision tree, a representation that inherently lends itself to interpretability.

The crux of our chapter delves into how these decision tree-based policies can be eloquently articulated using a language model. Such a model does not merely decode the features on the decision tree paths but contextualizes them in a pedagogically meaningful manner. This process is not just about simplifying technical jargon but about fostering a two-way interaction where learners can query, seek clarifications, and even pose what-if scenarios. Through empirical validation and user studies, we aim to demonstrate that this methodology does not only make the agent's reasoning transparent but also empowering. The work has been published and we refer readers for more details in the original paper [169]. The IRB protocol is Pitt STUDY21030223, Crowd-sourced Behavioral Studies for Human-Agent/Robot Interaction.

7.1 Background

Rapid advances in artificial intelligence and machine learning have led to an increase in the deployment of robots and other embodied agents in real-world, safety-critical settings [19, 39, 88, 137, 171]. As such, it is vital that practitioners – who may be laypeople that lack domain expertise or knowledge of machine learning – are able to query such agents for explanations regarding *why* a particular prediction has been made – broadly referred to as explainable AI [5, 51, 156]. While progress has been made in this area, prior works tend to focus on explaining agent behavior in terms of rules [75], vision-based cues [28, 108], semantic concepts [164], or trajectories [23, 52]. However, it has been shown that laypeople benefit from natural language explanations [3, 104] since they do not require specialized knowledge to understand [154], leverage human affinity for verbal communication, and increase trust under uncertainty [47].

In this chapter, we seek to develop a framework to generate natural language explanations of an agent's behavior given only observations of states and actions. By assuming access to only behavioral observations, we are able to explain behavior produced by *any* agent policy, including deep neural networks. Unlike prior methods which exhibit limited expressivity due to utilizing language templates [62, 78, 154] or assume access to a large dataset of human-generated explanations [37, 92], we propose an approach in which large language models (LLMs) can be used to generate free-form natural language explanations in a few-shot manner. While LLMs have shown considerable zero-shot task performance and are well-suited to generating natural language explanations [87, 103, 157], they are typically applied to commonsense reasoning as opposed to explaining model behavior, and are prone to hallucination – a well-known phenomenon in which false information is presented as fact [105]. It is an open question as to how LLMs can be conditioned on an agent's behavior in order to generate plausible explanations while avoiding hallucination.

Our solution, and core algorithmic contribution, is the introduction of a *behavior representation* (BR), in which we distill an agent's policy into a locally interpretable model that can be directly injected into a text prompt and reasoned with, without requiring fine-tuning. A behavior representation acts as a compact representation of an agent's behavior around a specific state and indicates what features the agent considers important when making a decision. We show that by constraining an LLM to reason about agent behavior in terms of a behavior representation, we are able to greatly reduce hallucination compared to alternative approaches while generating informative and plausible explanations. An additional benefit of our approach is that it enables *interactive* explanations; that is, the user can issue follow-up queries such as clarification or counterfactual questions. This is particularly valuable, as explanations are social interactions conditioned on a person's own beliefs and knowledge [106] and thus, are highly individual and may require additional clarification to be comprehensible and convincing [79].

Our approach is a three-stage process (see Figure 7.1) in which we, 1) distill an agent policy into a decision tree, 2) extract a decision path from the tree for a given state which serves as our local *behavior representation*, and 3) transform the decision path into a textual representation and inject it into pre-trained LLM via in-context learning [20] to produce a natural language explanation. In this chapter we show how our framework can be applied to multi-agent reinforcement learning (MARL) policies – a particularly relevant setting given the complex dynamics and decision-making resulting from agent-agent interactions. Through a series of

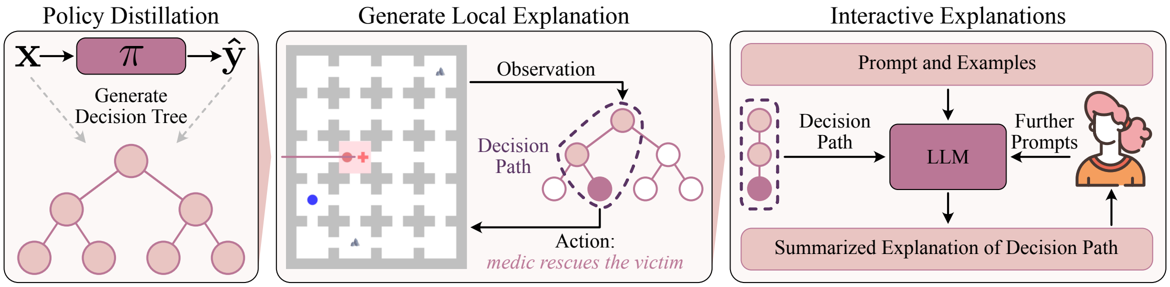


Figure 7.1: Overview of our three-step pipeline to explain policy actions: left: A black-box policy is distilled into a decision tree; middle: a decision path is extracted from the tree for a given state which contains a set of decision rules used to derive the associated action; right: we utilize an LLM to generate an easily understandable natural language explanation given the decision path. Lastly, a user can ask further clarification questions in an interactive manner.

participant studies, we show that, a) our approach generates model-agnostic explanations that are significantly preferred by laypeople over baseline methods, and are preferred at least as much as those generated by a human domain expert; b) when an agent policy is sub-optimal, participants find the ability to interact with our explanations helpful and beneficial; and c) our approach yields explanations with significantly fewer hallucinations than alternative methods of encoding agent behavior.

Explainable Agent Policies: Many works attempt to explain agent behavior through the use of a simplified but interpretable model that closely mimics the original policy [90, 117, 132, 151], a technique which has long been studied in the field of supervised learning [17, 123]. Although approaches that directly utilize inherently interpretable models with limited complexity during the training phase [22, 35, 135] exist, many researchers avoid sacrificing model accuracy for interpretability. We follow a similar approach in the second chapter on explainability, in which we leverage a distilled interpretable model to gain insight into how the agent's policy reasons.

Natural Language Explanations: Outside of explaining agent behavior, natural language explanations have received considerable attention in natural language processing areas such as commonsense reasoning [102, 118] and natural language inference [116]. Unlike our setting in which we desire to explain a given *model's behavior*, these methods attempt to produce an explanation purely with respect to the given input and domain knowledge, e.g., whether a given premise supports a hypothesis in the case of natural language inference [21]. Although self-explaining models [1, 66, 103] are conceptually similar to our goal, we desire a model-agnostic approach with respect to the agent's policy and thus seek to explain the agent's behavior with a separate model. While recent works have investigated the usage of LLMs in explaining another model's behavior by reasoning directly over the latent representation [18], this approach has yielded limited success thus far and motivates the usage of an intermediate behavior representation in this chapter.

7.2 Methods

We introduce a framework for generating natural language explanations for an agent from *only* observations of states and actions. Our approach consists of three steps: 1) we distill the agent's policy into a decision tree, 2) we generate a behavior representation from the decision tree, and 3) we query an LLM for an explanation given the behavior representation. We note that step 1 only needs to be performed once for a particular agent, while steps 2 and 3 are performed each time an explanation is requested. We make no assumptions about the agent's underlying policy such that our method is model agnostic; explanations can be generated for any model for which we can sample trajectories. Formally, we examine the setting in which an agent makes an observation of an environment and executes an action at each time step.

Notation: An agent observes environment state s_t at discrete timestep t, performs action a_t, and receives the next state s_{t+1}. Actions are sampled according to the policy $\pi(a_t|s_t)$ and a trajectory consists of a sequence of state-action pairs $\tau = s_0, a_0, s_1, a_1, \ldots, s_t, a_t$.

7.2.1 Distilling a Decision Tree

Our first step is to distill the agent's underlying policy into a decision tree, which acts as an interpretable *surrogate*. The decision tree is intended to faithfully replicate the agent's policy while being interpretable, such that we can extract a behavior representation from it. Given an agent policy π, we sample N rollouts $\tau_0, \ldots, \tau_{N-1}$ with which we learn a decision tree policy. While decision trees are simpler than other methods such as deep neural networks (DNNs), it has been shown that they are still capable of learning reasonably complex policies [16]. Intuitively, DNNs often achieve state-of-the-art performance not because their representational capacity is larger than other models, but because they are easier to regularize and thus train [7]. However, distillation is a technique that can be leveraged to distill the knowledge contained within a DNN into a more interpretable surrogate model [14, 43, 63].

7.2.2 Behavior Representation Generation

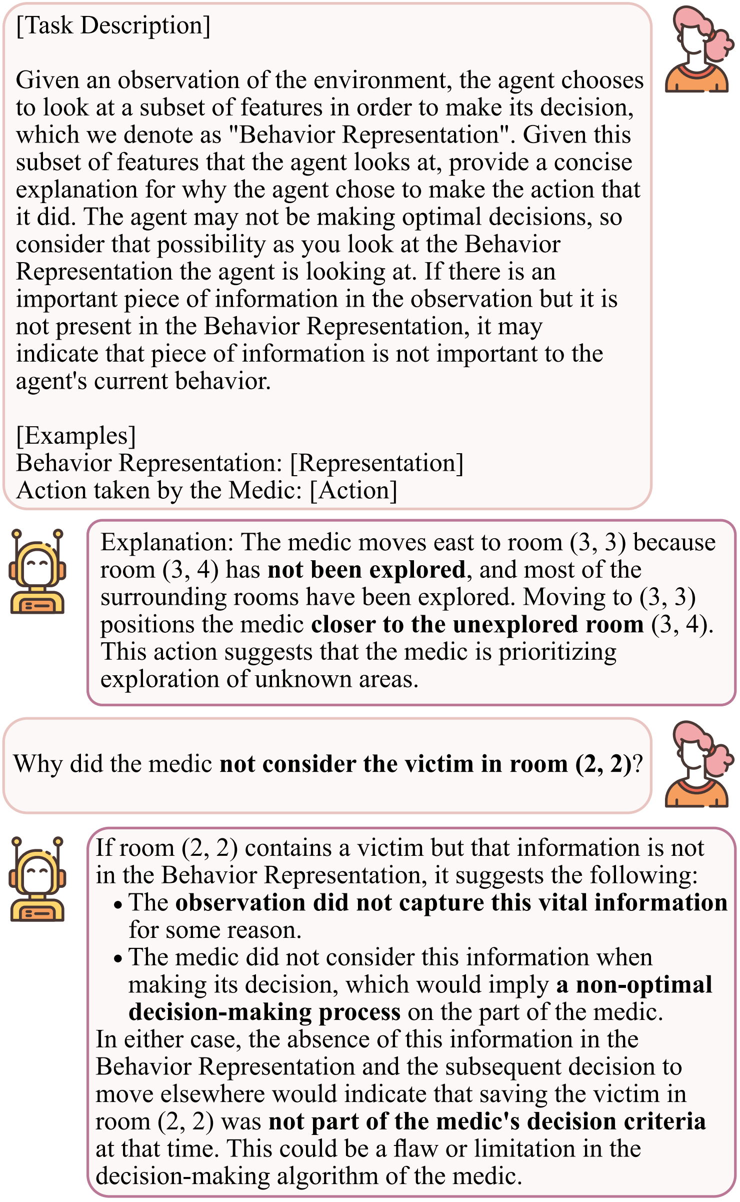
Given our surrogate decision tree, the next step is to generate a behavior representation for the agent given a particular state-action pair that we wish to explain. The behavior representation is a locally interpretable model of the agent's behavior at a particular point in the state space. We create a behavior representation by extracting the decision path from our learned decision tree corresponding to the given state. Intuitively, this decision path represents the decision rules that the agent may have reasoned with – we say *may* because the surrogate decision tree is only an approximation of the agent's true reasoning process – when taking the observed action. The decision path is an ideal behavior representation since we can algorithmically translate it to natural language for injection into an LLM prompt without requiring any additional fine-tuning.

7.2.3 In-Context Learning with Behavior Representations

The last step in our approach is to define a prompt that constrains the LLM to reason about agent behavior with respect to a given behavior representation. Our prompt consists of four parts: a) a concise description of the environment the agent is operating in, e.g., state and action descriptions, b) a description of what information the behavior representation conveys, c) in-context learning examples, and d) the behavior representation and action that we wish to explain. An example of this prompt is shown in Fig. 7.2. All parts except for (d) are pre-defined ahead of time and remain constant for all queries, while our framework provides a mechanism for automatically constructing (d). Thus, our system can be queried for explanations with no input required by the user unless they wish to interact and submit follow-up queries.

7.2.4 Interactive Explanations

Because our system leverages an LLM to generate explanations, it facilitates follow-up interactions that are conditioned on the preceding behavior representation and explanation. This is particularly valuable when the agent policy is sub-optimal, and the agent's actions are not aligned with the user's expectations. Two common types of interactions are clarification questions, e.g., "Why did the agent not consider feature X when making its decision?", and counterfactual questions, e.g., "What if feature Y were present instead of feature X?".



[Task Description]

Given an observation of the environment, the agent chooses to look at a subset of features in order to make its decision, which we denote as "Behavior Representation". Given this subset of features that the agent looks at, provide a concise explanation for why the agent chose to make the action that it did. The agent may not be making optimal decisions, so consider that possibility as you look at the Behavior Representation the agent is looking at. If there is an important piece of information in the observation but it is not present in the Behavior Representation, it may indicate that piece of information is not important to the agent's current behavior.

[Examples]
Behavior Representation: [Representation]
Action taken by the Medic: [Action]

Explanation: The medic moves east to room (3, 3) because room (3, 4) has **not been explored**, and most of the surrounding rooms have been explored. Moving to (3, 3) positions the medic **closer to the unexplored room** (3, 4). This action suggests that the medic is prioritizing exploration of unknown areas.

Why did the medic **not consider the victim in room (2, 2)**?

If room (2, 2) contains a victim but that information is not in the Behavior Representation, it suggests the following:

- The **observation did not capture this vital information** for some reason.
- The medic did not consider this information when making its decision, which would imply **a non-optimal decision-making process** on the part of the medic.

In either case, the absence of this information in the Behavior Representation and the subsequent decision to move elsewhere would indicate that saving the victim in room (2, 2) was **not part of the medic's decision criteria** at that time. This could be a flaw or limitation in the decision-making algorithm of the medic.

Figure 7.2: Interactive Conversation: The user is asking more clarification questions about the initial explanation generated by our method. The users are fluent English speakers and reside in the United States. From the right side, the first prompt of providing information to the agent is from the human researchers, and the second prompt of asking questions is an example question asked by human participants. The teacher agent's responses are from the left side.

7.3 Experiments

We conduct a series of participant studies and empirical experiments in order to answer the following questions:

- Does our approach generate plausible and useful explanations?
- Is the user interaction enabled by our approach helpful?
- Is our approach able to minimize hallucination?

We evaluate our task in a multi-agent Urban Search and Rescue (USAR) environment, implemented as a 2D grid spanning 4×5 rooms in which each room can contain rubble, a victim, or both (note that victims can be hidden under rubble). The goal of our two distinct agents, an engineer, who can remove rubble, and a medic, who can heal victims, is to rescue all victims scattered throughout the rooms. At each step, agents can take a navigation action to explore previously unexplored rooms, move through them, or conduct their unique action of removing rubble (engineer) or rescuing a victim (medic). All experiments utilizing an LLM use OpenAI's GPT-4 model with no additional fine-tuning [2].

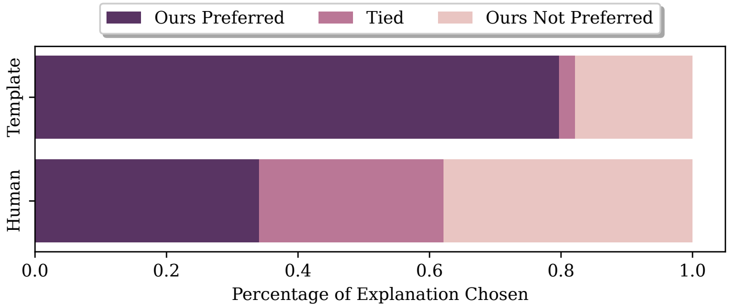


Figure 7.3: Participant preference when presented with two explanations and asked to choose which is most helpful to understand agent behavior: Ours vs. Template (top) and Ours vs. Human (bottom) over 320 responses.

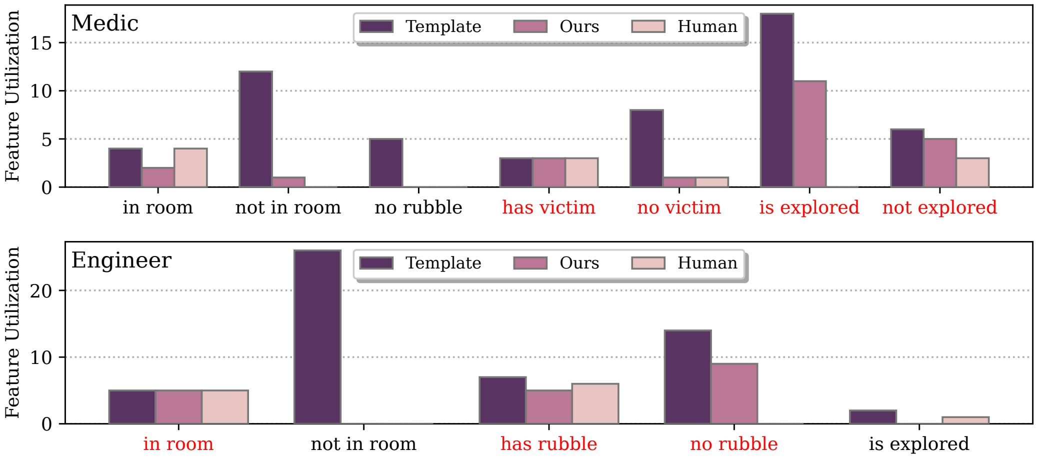


Figure 7.4: Frequency of selected features when explaining actions. Features highlighted in red were part of the examples provided in the LLM's prompt.

7.3.1 Does Our Approach Produce Useful Explanations?

We conduct an IRB-approved human-subject study with 32 participants in which we present ten environment states and actions – 5 each for the medic and engineer – to each participant alongside explanations generated by a) our proposed framework, b) an explanation generated via language templates directly from the behavior representation, and c) a human-generated explanation. The explanations are presented as two pairwise decisions and each participant is asked to choose which explanation in each pair is most helpful in understanding the agent's action. The subjects reside in the United States and are fluent English speakers.

The results are shown in Fig. 7.3 from which we can draw two conclusions: participants significantly ($p < 0.05$ with one-tailed binomial test) prefer the explanation generated by our approach over the baseline generated from language templates; and participants do *not* prefer the human-generated explanations over those generated by our approach. Analysis over the features which are referenced in each of the explanations (Fig. 7.4) reveals that our explanations and the human-generated explanations tend to emphasize the same features, while the template-generated explanations often refer to features that may be less helpful, e.g., "not in room". From these results we reason that our explanations are indeed useful to participants as they focus on helpful features and are as naturally acceptable as human-generated advice.

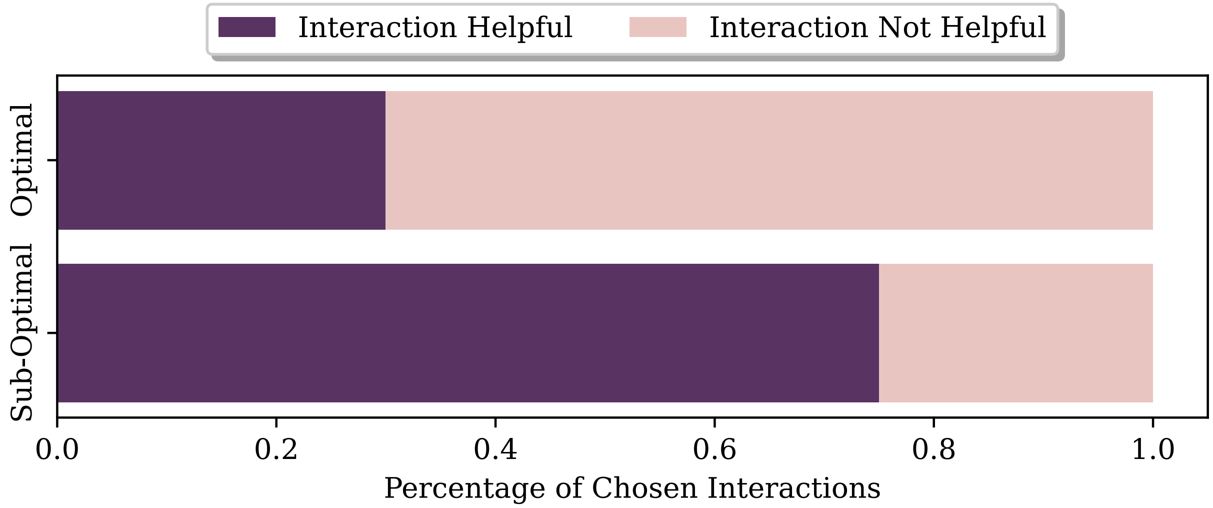


Figure 7.5: Helpfulness of interaction after presented with an explanation with respect to optimal and sub-optimal policies across 20 responses.

7.3.2 Is User Interaction Helpful?

We conduct a follow-up IRB-approved human-subject study with 5 participants in which we present 4 environment states, actions, and explanations generated by our framework and offer the user to further interact with the LLM via a chat window. Participants were asked to indicate whether they found the ability to interact helpful in understanding the agent's action. The results are shown in Fig. 7.5 and resulted in an interesting insight. When the agent's policy was optimal and the action aligned with the participant's expectations, interactions were largely found as not helpful; the initial explanation was sufficient for most participants. However, when the agent's policy was sub-optimal – which we achieved by creating a policy in which the agents explored all rooms first before rescuing victims or removing rubble – the participants found the ability to interact with the explanation and ask follow-up clarification questions helpful. An example of such an interaction is shown in Fig. 7.2.

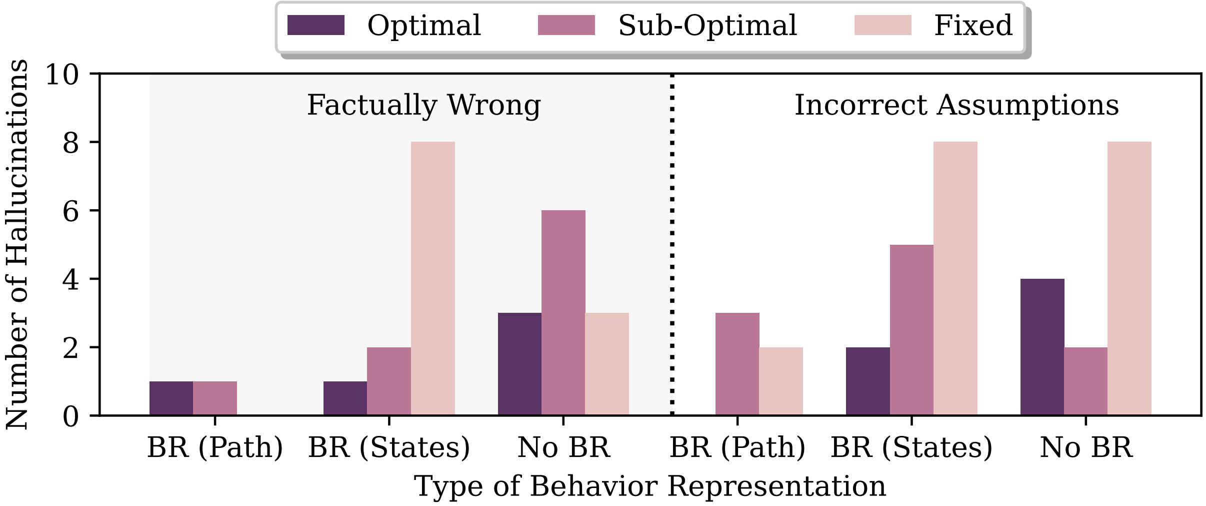


Figure 7.6: Number of hallucinations that are factually wrong (left) or have incorrect assumptions (right) across our samples.

7.3.3 Does Our Approach Decrease Hallucination?

In order to generate useful explanations, we wish to limit the amount of hallucination exhibited in the LLM's responses. We quantify the hallucination rate of our proposed approach over 30 generated explanations – 10 each from an optimal, sub-optimal, and "fixed" policy which exhibited state-invariant behavior, i.e. always traveled north. We compared our explanations with those generated by an LLM when the behavior representation was replaced by a list of state-action pairs randomly sampled from the agent's trajectories τ, and when the behavior representation was removed entirely. The former represents the case that, rather than producing a compact representation of the agent's behavior, we simply provide direct examples of the behavior and let the LLM reason over them. While seemingly intuitive, in practice this method does not work well because only a limited number of examples – 5 in this case – are able to fit in the LLM's context window.

The results are shown in Fig. 7.6 from which we can see that our approach yields significantly fewer numbers of hallucinations than either alternative. An interesting observation is that the explanations generated without a behavior representation (No BR) actually produces fewer factually wrong hallucinations than a behavior representation consisting of state-action samples (BR (States)) for the "fixed" policy. We find that this is because, in the absence of a behavior representation, the LLM tends to be more conservative in making claims and thus hallucinates less, at the expense of producing less useful explanations.

7.4 Conclusion

In this chapter we propose a model-agnostic framework for producing natural language explanations for an agent's behavior. Through construction of a *behavior representation*, we are able to prompt an LLM to reason about agent behavior in a way that produces plausible and useful explanations, enables a user to interact and issue follow-up queries, and results in a minimal number of hallucinations, as measured through two participant studies and empirical experiments. Although we think this is a promising direction, we emphasize that these results are also exploratory in nature and represent first steps. For example, the methods apply when the observations of the model are composed of meaningful features. For the pixel-level input, it does not seem to be able to provide good explanations, however, maybe LLM can directly tell the explanations from the image. We intend to follow up with more comprehensive studies and analysis in future work.

Chapter 8

A Naive Finetuning Approach to Transfer Prediction

So far we have presented our work on action advising, however, we would also like to introduce two of our previous works in transfer learning, to provide a more comprehensive background and motivations to demonstrate the advantages of approaches in action advising. In this chapter, we delve into the application of transfer learning to enhance the capabilities of agents assisting in urban search and rescue tasks. Understanding human actions and underlying beliefs is pivotal for developing agents that can effectively navigate and make strategic decisions in varying and complex environments. Traditional transfer learning, through direct model parameter transfer, allows us to leverage pre-existing knowledge from a source environment to improve agent performance in a novel, target environment. This method maintains core functionalities and adapts to new contexts without the extensive data collection and training typically required for each unique setting. We specifically examine the transition from a simpler, less varied environment to one that is more intricate and diverse, focusing on navigation and victim triage strategy adaptation. By utilizing a Transfer Learning Diffusion Convolutional Recurrent Neural Network (TL-DCRNN) alongside attention-based LSTM networks, we aim to capture and transfer spatial and strategic insights effectively, demonstrating the potential of transfer learning to expedite and refine the learning process in critical, life-saving applications. This approach not only underscores the adaptability and efficiency of transfer learning but also provides a comparative perspective to action advising, illustrating a more conventional yet robust method for neural network-based knowledge transfer [53].

8.1 Background

In an urban search and rescue (USAR) task, human rescuers may navigate better and rescue more victims with the help of an artificial agent that observes and predicts their navigation and rescue activities, and opportunistically intervenes to give them assistance. Simple assistance and guidance include reminding rescuers not to revisit an area already visited, how to efficiently go to desired places, and whether they are likely to find victims that they would consider high priority in saving. The utility of an agent's advice to rescuers is dependent on the accuracy of agent's predictions of the rescuer's intents; interventions based on incorrect predictions may be misleading.

Behavior prediction in an USAR task is challenging due to various factors, including but not limited to (a) incomplete information about where victims are, (b) changes in environmental and victim conditions , and (c) difficulty to obtain in obtaining data on rescue missions performed by humans. In a natural disaster scenario, such as after an earthquake, traversability of a building may change due to holes opening in walls or debris blocking passages, requiring rescuers to form ad-hoc navigation strategies during exploration. Rescuers also may be faced with decisions regarding *triage* (providing essential medical care to victims) priority. For example, a rescuer may decide to temporarily disregard lightly injured victims in order to search for and triage critical victims first, triaging lightly injured victims later.

In USAR tasks, real data is expensive to obtain, and is usually associated with unique configurations of the environment. This raises the question: how can an agent learn navigation and victim triage prediction in USAR task that can efficiently generalize and transfer to more complex environments and tasks? Representation and abstraction of spatial recognition in the humans has been studied widely. The knowledge human navigation relies on primarily can be primarily characterized by a labeled graph [27]. Instead of perceiving environments into a global coordinate system, this cognitive map built from local information cannot guarantee geometric consistency [155]. There is a general agreement that people have hierarchical representations of space [24, 48, 64, 141], and this typically leads to the wrong answer to trick questions such as "What direction is Reno from San Diego?" (Answer: Northwest). Many people know San Diego is in California and Reno is in Nevada, and the spatial perception on state locations misleads that on cities. Similar effects are found on a local scale where people cluster buildings and other landmarks together into regions. Distances between locations within a cluster are judged to be shorter than they actually are, while the distance between locations in different clusters are judged to be longer.

Simulations of learning and question answering based on this hypothesized hierarchical organization have been developed and analyzed in various works [121], [152], [99]. The goal of this chapter is to develop transfer learning methods that provide zero- or few-shot transfer for human navigation and triage strategies from a *source* domain to a more complex *target* environment and task. In particular, we developed effective transfer learning techniques for navigation prediction from a smaller to a larger indoor environment and for a larger number of victim injury criticality classes. In this work, we used the 3D Minecraft platform [36, 74] as a testbed. Both source and target environments represented building interior spaces. The two environments had different room layouts, and we collected human rescuer navigation trajectories while they performed the simulated USAR tasks. We augmented the trajectories collected from human participants with trajectories generated by rule-based simulated participants, in order to introduce more diversity to the set of rescue strategies for training. Our method has demonstrated several benefits of our approach: (1) the agent is able to make good predictions on the navigation and triage strategy of rescuers, (2) its prediction accuracy grows fast even with a small input number of trajectories, (3) the convergence time of training process is shortened significantly in the target domain while not affecting the accuracy, and (4) the experiment demonstrates the potential of transfer learning for USAR missions.

For the past decade, transfer learning has been studied extensively [114, 140, 142, 175, 176] . It has been recently used in reinforcement learning, where multiple tasks are learnt instead of a single one. Knowledge gained in some Markov Decision Processes can be leveraged to speed up the solutions of others [11, 12, 13], In computer vision, Joint Distribution Adaptation is proposed for robust transfer learning [93], which jointly adapts the marginal and conditional distribution. Domain invariant features of the source and target are extracted for visual object recognition [10]. Transfer learning applied to graphs has recently gained attention. [34] proposes a network transfer learning framework using the adversarial domain and graph convolution. Although training and test data still require having the same feature space and distribution, new tasks that share similar representations can be resolved easier in [85], and this work transfers the geometric information from source to target. Graph-based domain mapping is used to identify previously encountered games, and this provides a good starting place for learning [82]. With graph based skill acquisition, [85] and [131] capture community detection from a connectivity graph, and speed up learning using the transferred knowledge. The theoretical grounded framework for the transfer learning of GNNs can be found in [174].

There are works on training an agent to navigate while adapting to new environments, including [167] where knowledge is transferred from previous navigation tasks using a successor-feature-based RL algorithm. In [107], the autonomous agent is trained to navigate in diverse city environments, while performing transferred tasks of navigating in target new locations. We build upon our previous work on the observing and predicting agent [71] by incorporating a graphical representation of the environment, amenable to inference using graph embedded Recurrent Neural Network models and transfer to new domains (i.e., environment layouts). We refer readers to [67] [71] for the data collection process where rescuers are equipped with Minecraft skills.

Our approach to navigation prediction builds upon the Transfer Learning Diffusion Convolutional Recurrent Neural Network (TL-DCRNN) architecture presented in [101] to infer participant navigation, partitioning each domain's map utilizing the concept of a "clique" in a graph (a set of connected nodes) to form our clique group assignment for graph division. The original work predicts on traffic flow based on the Diffusion Convolutional Recurrent Neural Network (DCRNN) model proposed in [89]. It was able to predict the traffic flow of one city using the data collected from another. In our work, instead of using the partition for cities [100], we partition the graph based on the spatial recognition of the rescuers when performing the search task.

8.2 METHODS

The transfer-learning agent we developed is able to predict the navigation and triage activities of rescuers in the source task, using training data from human Minecraft players. The agent divides the space using graph methods and utilizes the TL-DCRNN to predict navigation when the rescuers search for victims; it also uses LSTM to predict triage strategies when the rescuers found victims and made decisions. In this work, we transfer the prediction model which is trained with the old scenarios, to predict the navigation and rescue activities of human rescuers in an unobserved situation. The implementation code has been released.[1]

8.2.1 USAR Domains

To train agents and evaluate the ability of agents to transfer to new domains, multiple USAR task domains were developed using two distinct building layouts (shown in Figure 8.1, and Figure 8.2): a smaller map with fewer rooms (referred to as *Sparky*), and a larger, more complex map (referred to as *Falcon*). For each domain, victims were placed in specific locations of the environment, and the environment was perturbed with blockage and holes on the walls, which are not initially known to rescuers, but are encountered as rescuers navigate the environment. Blockages in the hallways and rooms might make certain paths in the map impassable, while holes on the walls opened other possibilities for navigation. For prediction of rescuer navigation, a total of four task domains were created by varying the location of victims and perturbations: a single variant of the Sparky map, and three variants of the Falcon map of increasing difficulty (*Falcon-easy*, *Falcon-medium*, *Falcon-hard*), shown in Figure 8.3.

For prediction of rescuer triage strategy, we created two domains using the *Falcon* environment. The first domain, *Falcon-2victim*, contains victims of two severity levels, which are common to all domains: *regular* victims require 7 seconds to triage, and *critical* victims require 15 seconds to triage, and will expire 5 minutes after the start of the mission. The second domain, *Falcon-3victim*, introduces an additional victim severity level, *medium* victims, which require 12 seconds to triage and will expire 7 minutes into the mission.

The USAR mission required a human player/rescuer to work in an indoor environment in a limited time where victims were scattered. The rescuer is equipped with a medical kit which enabled them to triage victims in a few seconds. A device that beeps to report if one or more live victims were inside a room was utilized, however the human participant may or may not have understood the significance of the beeping. Knowledge of the beeping was a condition that was varied in different trials, i.e. in some conditions the participant was told the meaning of the beep at the beginning of the experiment, whereas in other conditions, the participant may not have been given this information.

---

[1] https://github.com/sophieyueguo/tl_navi

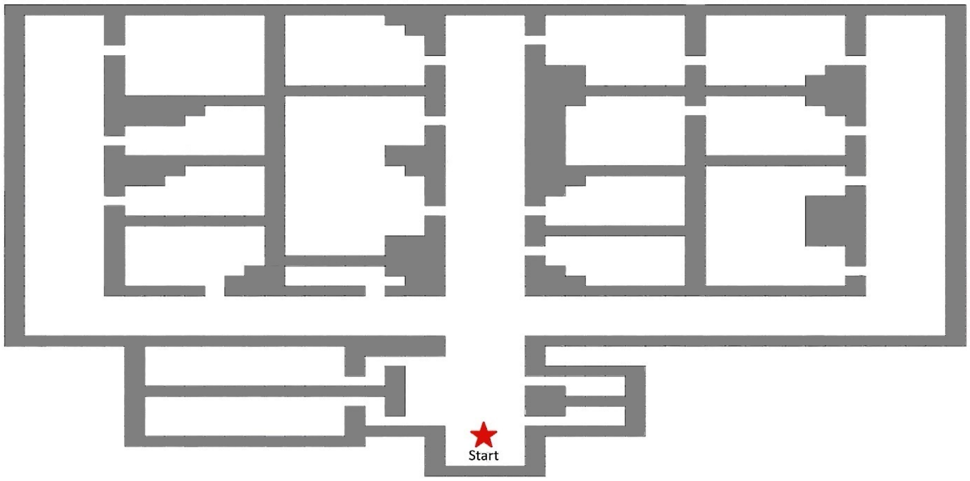


Figure 8.1: Sparky

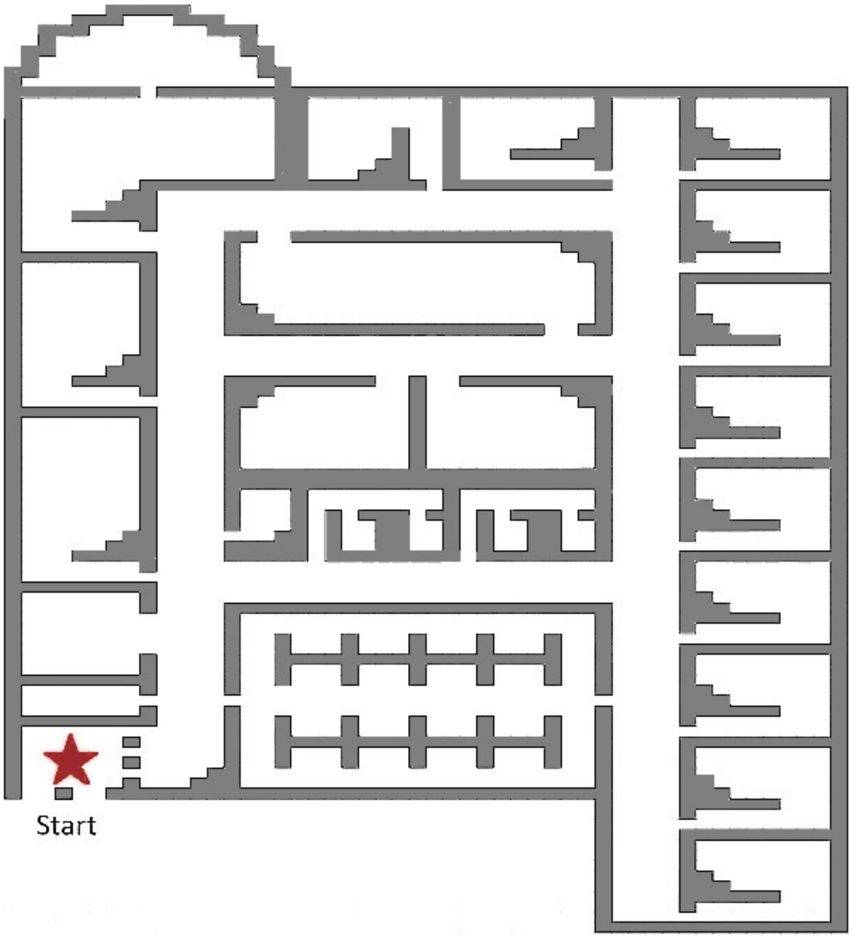


Figure 8.2: Falcon

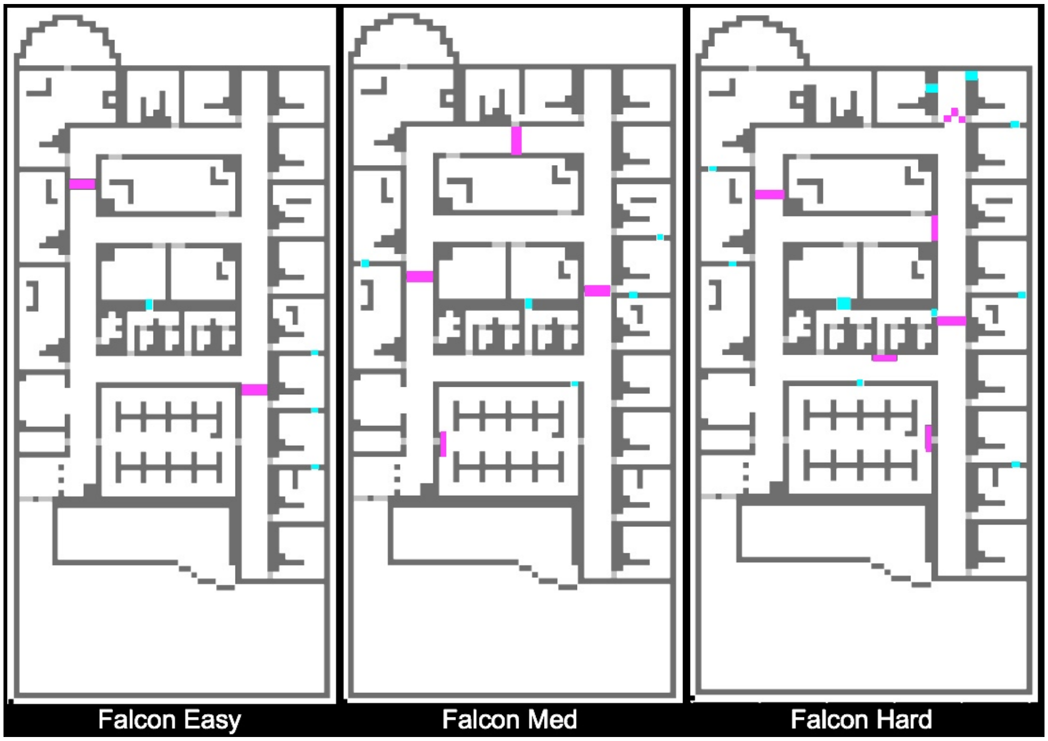


Figure 8.3: Map of the three versions of Falcon: easy, medium, and hard collected in [67]. Grey indicates walls, Magenta indicates blockages, and Cyan is for openings.

8.2.2 Agent Observations

The agent observes a stream of information containing the following values from the rescuer's trajectory:

- **Rescuer State:** The position and orientation of the rescuer in the environment,
- **Field of View Contents:** Blocks of interest (e.g., victims) that are present in the rescuer's field of view,
- **Victim Interaction Events:** Including starting, completing, and abandoning triage attempts,
- **Environment Interaction Events:** Including opening or closing doors, and entering or exiting rooms,
- **Victim Location Device:** Emitted beep when the rescuer is near a room containing an untriaged victim.

The Rescuer State and Field of View Contents update at a rate of 10Hz, all other observations are asynchronous events.

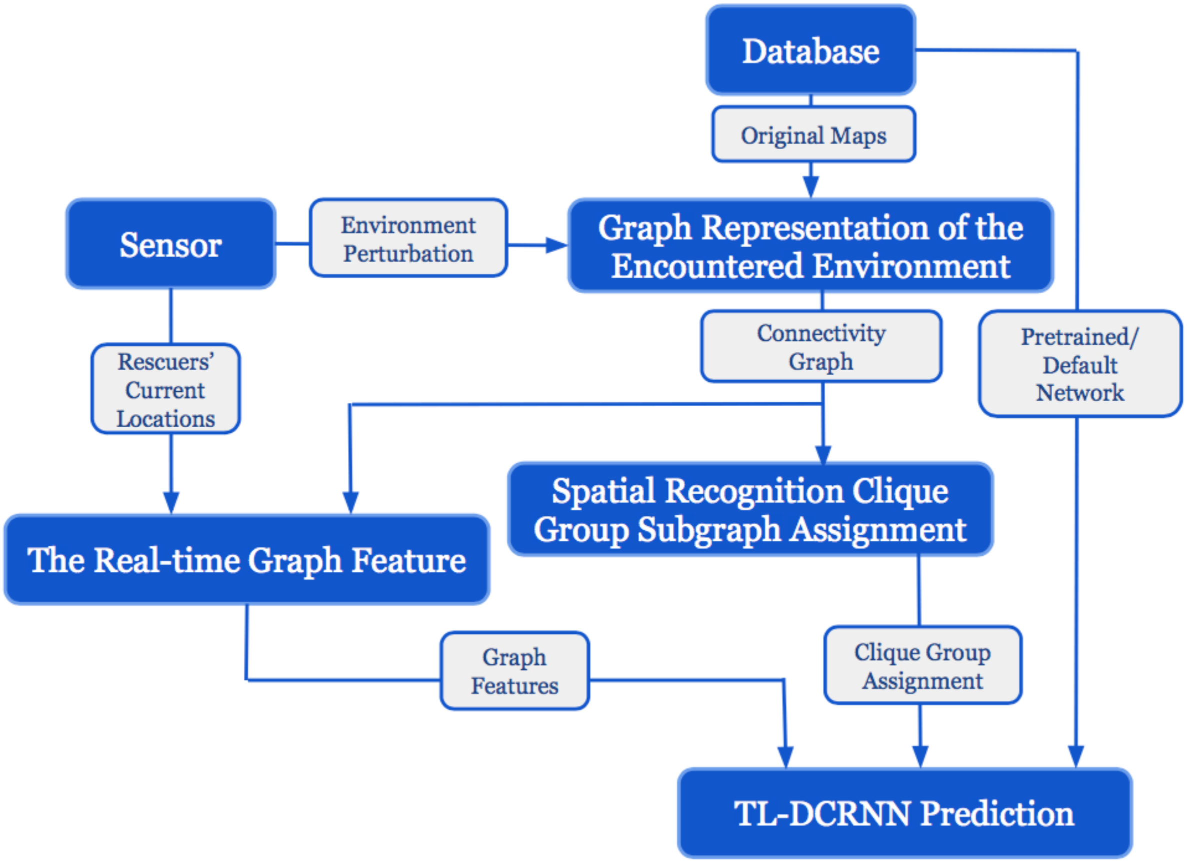

Database
Original Maps
Sensor
Environment Perturbation
Graph Representation of the Encountered Environment
Connectivity Graph
Pretrained/ Default Network
Rescuers' Current Locations
Spatial Recognition Clique Group Subgraph Assignment
The Real-time Graph Feature
Graph Features
Clique Group Assignment
TL-DCRNN Prediction

Figure 8.4: The Architecture of the Navigation Prediction Process

8.2.3 Navigation Prediction

The navigation prediction task involves the agent predicting the next room a test (unknown) human will enter next, predicted at particular time intervals. The humans are players familiar with the Minecraft games, and are recruited to play the simulated search and rescue mission in Minecraft. Figure 8.4 illustrates the main components / processes of our architecture (blue blocks) and data flow between the components (grey blocks). In addition to the agent's observation stream listed above, the agent is provided with a map of the environment and victim and environmental perturbations (e.g., debris blockages and openings in walls), from which it generates a graph-based representation of the environment. The agent updates the representation based on victims and environment perturbations observed by the rescuer in the agent's data stream. Features are extracted from the graph-based representation, and a TL-DCRNN model is used to forecast future features, from which room visitation predictions can be made.

Graph Representation

The original map of the environment is converted into an abstract connectivity graph, $G = \{V, E\}$, where each $v \in V$ represents a predefined, semantically meaningful region (room, a segment of hallway, or intersections), and $e(v_1, v_2) \in E$ indicates connectivity between the regions v_1 and v_2 in the following way. Each (x, y, z) location in the original map is manually assigned to a region; environment perturbations result in edges either being added (in the case of a wall opening) or removed (in the case of rubble blockage).

Spatial Recognition Clique Group Subgraph Assignment

Given the connectivity graph $G = \{V, E\}$, the sub-graph list $\{G_c\}$ is referred to as "clique group". A clique is defined as a set of rooms with a door adjacent to a common hallway segment. Formally, for a hallway segment, $v_h \in V$, the clique associated with the segment is defined as $G_c^h \triangleq \{v_i | (v_i, v_h) \in E; v_i \text{ not a hallway segment}\}$. Hallway segment boundaries are defined by walls or perturbations blocking passage, as well as intersections with other perpendicular hallway segments (i.e., T-intersections and corners). Clique groups are the hallway clique unions that share elements, defined as $G_c \triangleq \{G_c^{h_1} \cup G_c^{h_2} | G_c^{h_1} \cap G_c^{h_2} \neq \}$. We hypothesize that rescuers are likely to explore all rooms within a clique, and navigate to an adjacent, unexplored clique once the current clique has been fully explored.

Real-time Graph Feature Generation

Graph features are computed whenever the rescuer enters a new vertex in the graph representation of the map (i.e., enters a new room or hallway segment). Graph features are calculated using a history of the previous T vertices the rescuer occupied. A single graph feature at time t is a vector denoted as $f_t \in \mathbb{R}^k$, where k is the number of vertices in the connectivity graph. Each element in the feature vector is defined as $f_t^{(i)} = \gamma^{(t-t_i)}$, where t_i is the most recent time step when the rescuer was located in v_i. In essence, the feature vector captures the region the rescuer currently occupies (indicated with a value of 1 at the index of the region), and the recent history of the rescuer's trajectory (indicated by values less than 1 in the feature vector, where elements with larger values correspond to more recently visited regions).

TL-DCRNN Navigation Prediction

The future sequence of rooms visited by the rescuer is predicted using a TL-DCRNN model. The model takes as input as sequence of graph features, $F_t = \{f_{t-T+1}, f_{t-T+2} \cdots f_t\}$, and the sub-graph list $\{G_c\}$, and predicts the future sequence of graph features, $F_t^{output} = \{f_{t+1}, f_{t+2} \cdots f_{t+T}\}$. The TL-DCRNN model learns as set of filter weights, $\mathbf{W}_O$ and $\mathbf{W}_I$, which intuitively represent the likelihood of the next room visited by the rescuer (outflow), and the likelihood of the room previously visited (inflow). After partitioning the graph, the sub-graph features are passed to a Recurrent Neural Network (RNN), which outputs the series of predicted graph features. Graph features are calculated using a diffusion process defined by the equation $\mathbf{W}_G F = \sum_{d=0}^{K-1} (\mathbf{W}_O (\mathbf{D}_O^{-1} \mathbf{A})^d + \mathbf{W}_I (\mathbf{D}_I^{-1} \mathbf{A})^d) F$ [101]. Here, K refers to the maximum number of steps taken for diffusion, $\mathbf{A}$ is the adjacency matrix of the graph, $\mathbf{D}_O$ and $\mathbf{D}_I$ are in-degree and out-degree diagonal matrices of graph nodes, thus $\mathbf{D}_O^{-1}\mathbf{A}$ and $\mathbf{D}_I^{-1}\mathbf{A}$ are transition matrices which indicate the flow, or transition frequency between regions. The predicted sequence of rooms visited by the rescuer can be extracted from the index of the maximum value of the predicted feature vectors at each time step.

8.2.4 Triage Strategy Prediction

In the triage task, the agent predicts over time the next victim type the human rescuer will triage. We hypothesize that a rescuer's triage strategy is agnostic to the location of the victim, and only depends on the reward that a rescuer gets in triaging a victim of a particular severity class of injuries.

Triage Strategy

We formulate the triage strategy prediction as a classification problem, using a sequence of Field of View observations (specifically, the location and severity of victims within the rescuer's Field of View) and observation timestamp as input.

Based on the set of victim severity levels, we identify four categories of strategies a rescuer can employ for triaging victims:

1. **Strict:** Rescuer exclusively triages *critical* victims during the first 5 minutes, *medium* victims from 5–7 minutes and *regular* victims from 7–10 minutes. In the *Falcon-2victim* domain, the rescuer following this strategy will triage *regular* victims from 5–10 minutes.
2. **Slack:** Rescuer triages *critical* and *medium* victims as they are discovered during the first 7 minutes, then triage *regular* from 7–10 minutes.
3. **Preemptive:** Rescuer triages victims as discovered regardless of victim severity.
4. **Probabilistic:** Rescuer triages a discovered victim only if the expected number of victims that can be triaged in the remaining time is less than the number of victims remaining in the environment. The rescuer is made aware of the total number of victims from each severity class, and can therefore calculate both the expected and remaining number of victims.

Attention based LSTM

We use a sinusoidal embedding for the position of the victims and the timestamp, following the works of [150]. At every timestep, we collect the list of victims, and use a common feedforward network E_ψ to convert the information into embeddings. To consider the case of no victims, we also include a "dummy" victim embedding at each timestep. Let there be N_t victims at time t. Each victim is depicted by a tuple of severity level, r, and (x, y) locations i.e. $v_i^t = (r_i^t, x_i^t, y_i^t), \quad i \in \{1 \ldots N_t\}$. This representation is fed into the feedforward network to get the victim embedding $e_i = E_\psi(v_i)$. The embeddings are given by $\{e_i\}\Big|_{i=1}^{N_t} \cup \{e_0\}$ where e_0 is the learned dummy victim embedding. This set of embeddings is passed into a self-attention network A_θ where θ are the learnable parameters, giving us modified embeddings $\{f_i\}_{i=0}^{N_t} = A_\theta\Big(\{e_i\}_{i=0}^{N_t}\Big)$. Next, we take the average of these embeddings as the "summary" vector that goes into the LSTM, $s_t = \frac{\sum_{i=0}^{N_t} f_i}{N_t+1}$. This architecture allows us to account for variable number of victims at each timestep without making the architecture task specific. This vector is used as input to the LSTM. The final triage strategy prediction y_t is given by the LSTM equation $y_t = \text{softmax}(g_\phi(h_t)), \quad h_t, c_t = \text{LSTM}(s_t, h_{t-1}, c_{t-1})$, where h_t, c_t are the hidden and context state of the LSTM, and g_ϕ is a feedforward network.

8.3 EXPERIMENTS

We performed a series of experiments to evaluate the agent's ability transfer prediction models trained on a source domain to a target domain in the USAR domains described in Section 8.2.1. To train and evaluate the networks, we used a previously collected set of trajectories generated by human participants on the Falcon map variants [67], and collected trajectories generated by human participants on the Sparky map. In each experimental run, a single human participant navigated a given map and triaged victims, accumulating score points for each triaged victim. Score point were allocated in ascending order on seriousness of injury. Each participant was assigned to only one map or map variant, in the experiment. In other words no participant repeated the experiment in two or more maps.

For navigation strategy prediction, there were 8 trajectories of map Sparky for training, 138 trajectories of map Falcon for training, and 33 trajectories of map Falcon for test. Trajectories of different map Falcon perturbations were considered as different trials. For triage strategy prediction, we used a rule-based agent conditioned on rescue strategy to generate trajectories in the Falcon map. For domains for triage prediction (*Falcon-2victim*, *Falcon-3vicim*), we generated 100 trajectories for each triage strategy, resulting in a total of 300 trajectories for the *Falcon-2victim* domain and 400 trajectories for the *Falcon-3victim* domain. To introduce variation in the trajectories, we introduced perturbations in victim locations by making the victims do a "random walk" around their starting locations. Victim severity levels were also randomized.

From each rescue strategy, we used up to 70 trajectories for training, 10 for validation, and 20 for testing. Evaluating the transferability of the prediction models to new domains involves pre-training a model on trajectories from a *source* domain (e.g., *Falcon-easy*), and evaluating the performance of the model on predicting trajectories from a *target* domain (e.g., *Falcon-hard*) after performing additional training on trajectories from the target domain. Note that the trajectories in different domains other than the domain where network is trained with are considered "new" data by the network.

8.3.1 Evaluation Metrics

For evaluating navigation prediction, we used Mean Average Error (MAE) [89] and [101],

$$MAE(\mathbf{f}, \hat{\mathbf{f}}) = \frac{1}{|\Omega|} \sum_{i \in \Omega} |f_i - \hat{f}_i|$$

where the ground truth is represented by $\mathbf{f} = f_1, f_2 ... f_T$, the predictions are represented by $\hat{\mathbf{f}} = \hat{f}_1, \hat{f}_2 ... \hat{f}_T$, and Ω referred to the observed samples. For triage strategy prediction, we used the mean accuracy over time (MAT) as our final evaluation metric. In the initial stages, the prediction can be random due to lack of rescuer's behavior data around the victims. Therefore, the average is taken only after one minute. Given a sequence of observations $\mathbf{O}$ and the ground truth rescue strategy $\mathbf{c}$, the metric is given by:

$$\mathbf{MAT}(\mathbf{O}, \mathbf{c}) = \frac{1}{|\mathbf{O}|} \sum_{t} \mathbb{I}(\mathbf{y_t} = \mathbf{c})$$

where $\mathbf{y_t}$ is the predicted strategy at timestep t. $\mathbf{c}$ is one of the four classes mentioned in Section 8.2.4.

8.3.2 Navigation Prediction with Transfer Learning

With the transfer learning experiments, we aim to answer the following questions regarding the navigation prediction model: (1) How effective/accurate is a model in the test domain that has learned to predict with a certain accuracy in the source domain? Does it achieve zero-shot learning? (2) How does the performance of the model in the source domain compared with a model that has been trained from scratch in the target domain? (3) How does the time required to train a source model and refine in the target domain with few-shot learning compare with the time to train a model in the target domain from scratch? (4) How does the number of input trajectories affect performance? (5) How do maps with different room configurations, vs the same map with different perturbations affect the transfer learning process? (6) Does the abstraction level of graph affect performance?

Baselines

We compare our mechanism with a Rule-based Navigation Prediction and a traditional time-series prediction method Autoregressive Integrated Moving Average (ARIMA) Model.
Rule-Based Model. The rule-based agent could make two levels of prediction on the rescuer's navigation behavior: (1) the next clique group, and (2) the next room(s). The rule-based prediction was dependent on the knowledge condition of rescuers, i.e. if they knew about what beeping meant, how long triaging victims would take, and how much reward they would get after finishing triaging the victims. Generally the prediction accuracy was higher for rescuers who had the knowledge of the beep. The rule-based method considers the following to predict next room(s):

- The current location of the rescuer
- The rooms not visited in the current clique group
- Whether the rescuer knows about beep
- Whether seriously injured victims are dead
- The rooms connected to the current location with a hole/internal door

The factors to predict next clique group:

- The current clique group:
- Whether seriously injured victims are dead
- Whether rooms in the current clique group have victims
- Whether there is a new clique group nearby

ARIMA Model. Autoregressive integrated moving average (ARIMA) was widely used [149] to perform time series prediction. One of its important variation models is Seasonal Autoregressive Integrated Moving Average with eXogenous variables (SARIMAX) [91], which includes a seasonal pattern in the prediction for short-term forecasting. By indexing the active regions and concatenating trajectories of different rescuers, we are able to observe a "seasonal" pattern, where different rescuers might visit same locations in a similar order. This enabled us to implement SARIMAX as a baseline. However, this extended ARIMA model does not take a graph structure of the map into consideration, and cannot be trained with data from other maps or perturbations. It only took in the training trajectories of the target map and perturbations.

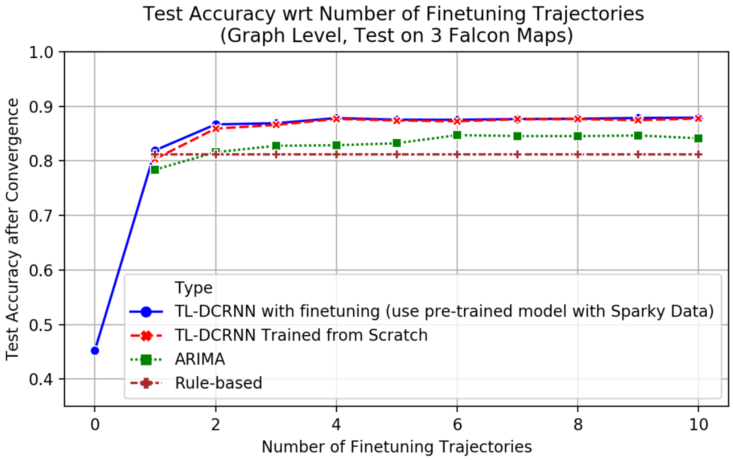


Figure 8.5: Transfer from Map Sparky to Map Falcon

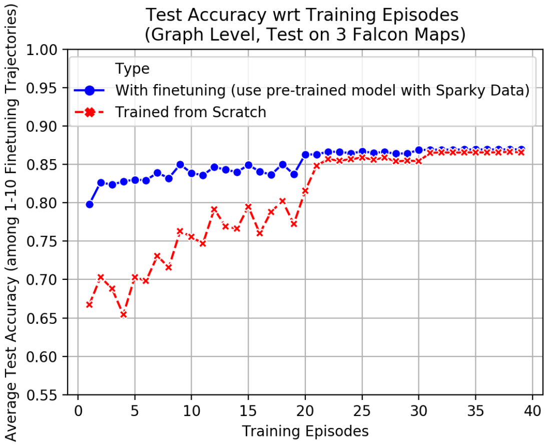


Figure 8.6: Transfer from Map Sparky to Map Falcon

From *Sparky* to *Falcon*

We compared the accuracy of the pre-trained model with fine-tuning *Sparky* trajectories (in blue), i.e. adding *Falcon* trajectories to the *Sparky* pre-trained model and the accuracy of the model trained on *Falcon* from scratch (in red) in Fig. (8.5) and (8.6). We gradually added a few training and validation trajectories of the *Falcon* domains (*Falcon-easy*, *Falcon-medium*, *Falcon-hard*) to the Sparky model (ranging from 1 to 10 incrementally, distinguished by different map perturbations respectively and took the average). We had the following observations on the converged accuracy and the training process: (a) TL-DCRNN outperformed ARIMA (in green) and the rule-based (in brown) baselines. (b) With increasing number of train/val trajectories, the accuracy after convergence of all models slightly improved with respect to varying the number of input *Falcon* trajectories. (c) TL-DCRNN with fine-tuning and TL-DCRNN trained from scratch have similar converged accuracy, but the one with fine-tuning converged faster.

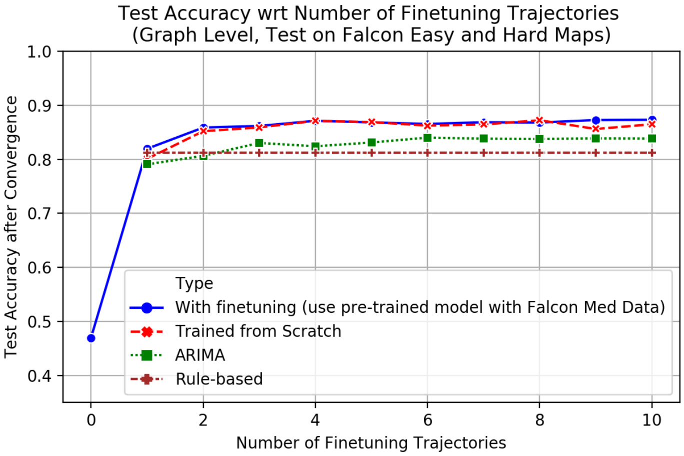


Figure 8.7: Transfer from Map Falcon Med to Map Falcon Hard and Easy

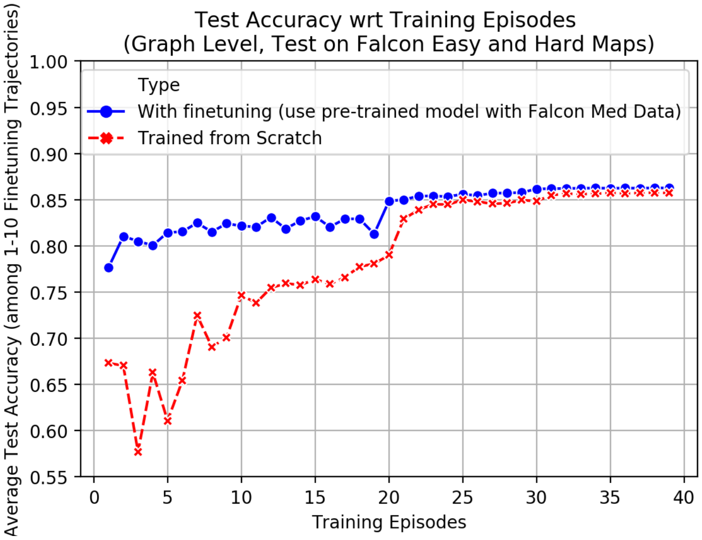


Figure 8.8: Transfer from Map Falcon Med to Map Falcon Hard and Easy

From *Falcon-med* to *Falcon-hard* and *Falcon-easy*

When using the trajectories of *Falcon* domains with different perturbation and victim densities instead of using the ones of *Sparky* domain, we obtained the results in Fig. (8.7) and (8.8). All the conclusions reported previously still held, and there was no significant difference in terms of accuracy after convergence or training process. This indicated that the data from the same map but with different perturbations was not necessarily more helpful than data from a different map. With perturbations, the navigation behavior can be as different as from another map.

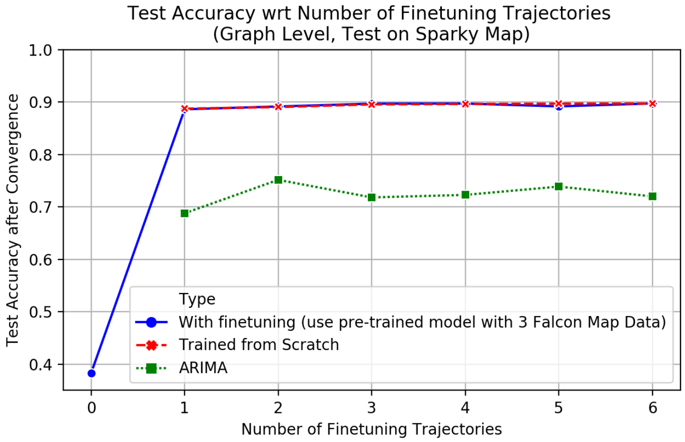


Figure 8.9: Transfer from Map Falcon to Map Sparky

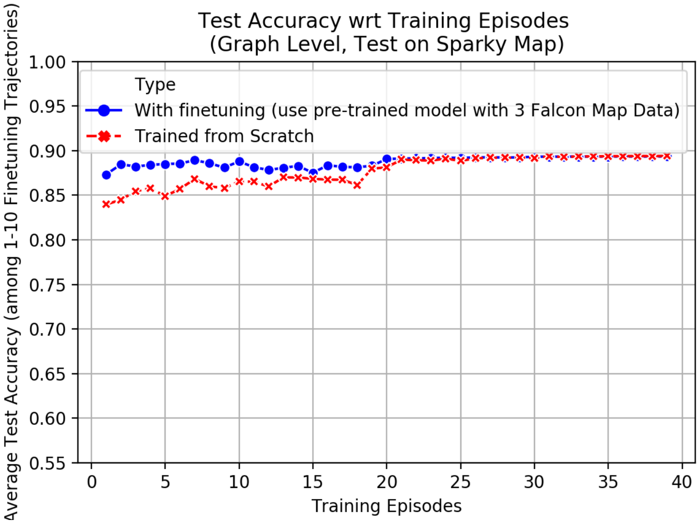


Figure 8.10: Transfer from Map Falcon to Map Sparky

From *Falcon* to *Sparky*

We reversed the process to transfer from the *Sparky* domain to the *Falcon* domain in Fig. (8.9) and (8.10). The *Sparky* map was relatively easier for TL-DCRNN to learn from and resulted in a higher accuracy than the previous comparison, while ARIMA was not able to improve. The TL-DCRNN also converged faster.

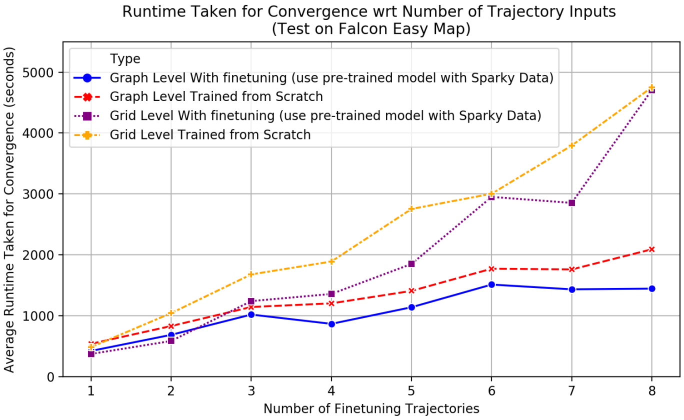


Figure 8.11: Compare with Lower Level abstraction

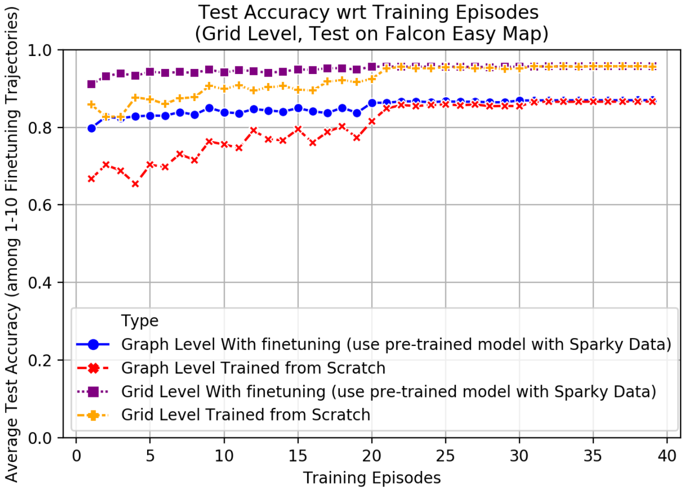


Figure 8.12: Compare with Lower Level abstraction

Comparison with Grid-Based Representation

We compared the graph-based representation prediction with predictions made using a grid representation of the map in Fig. (8.11) and (8.12). In the grid representation, each 3 cell by 3 cell region was considered as a graph node; edges connected adjacent node. The state space was 12.5 times larger. The denser graph led to a higher accuracy after convergence when transferring from *Sparky* domain to *Falcon-easy* domain. However, the run-time to convergence increased.

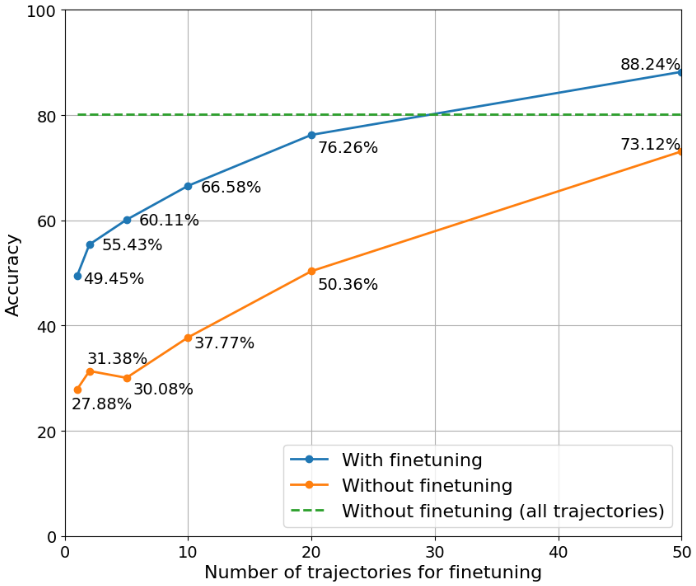


Figure 8.13: Transfer learning on triage strategy prediction

8.3.3 Triage Strategy Prediction with Transfer Learning

In this section, we compare the performance of the model trained on the harder *Falcon-3victim* domain from scratch, and transferring a model pre-trained on the easier *Falcon-2victim* source domain, and finetuning on few trajectories from the *Falcon-3victim* target domain, reducing the required training data and time. We therefore perform the following experiments:

1. Models trained on the *Falcon-3victim* domain from scratch, i.e. without finetuning, with a varying number of training trajectories.
2. Models pre-trained on the *Falcon-2victim* domain as a source domain with maximum number of training trajectories, and transferring the model to the *Falcon-3victim* as the target domain by finetuning with varying number of target-domain trajectories.

The results are shown in Figure 8.13. The dotted plot shows the test accuracy with 70 training trajectories in the *Falcon-2victim* domain. We train with 1, 2, 5, 10, 20 and 50 training trajectories from each strategy. Finetuning improves generalization performance even given a small number, such as 2, of finetuning trajectories from target *Falcon-3victim*. In our case, the source task does not contain information about the target task at all (for example, no *Falcon-2victim* trajectory contains information about *medium* severity victims), so the finetuning indeed improves generalization. Moreover, using even as few as 30 trajectories for finetuning the *Falcon-2victim* source model reaches the performance of the *Falcon-3victim* model trained from scratch with *all* trajectories (green constant line in the figure) and for more than 30 finetuning trajectories, it outperforms the *Falcon-3victim* model.

8.4 Conclusion

We have built an agent that makes predictions on the navigation and rescue strategies of a human rescuers in a simulated urban search and rescue mission. We showed experimentally that: (a) using an abstract representation, i.e graphs enables efficient navigation strategy transfer from source to target domains from smaller to large maps with different victim and perturbation configurations; (b) for triage strategy, training a source model with smaller number of victim classes and adding a few finetuning trajectories form the target domain with larger number of triage victim classes, is not only high performing and efficient, but surprisingly outperforms a model trained from scratch in the target domain, while also converging faster. This is an interesting finding and we will explore it further in additional domains in future work. We also plan to study transfer with a team of rescuers instead of a single one. This is a very challenging task, not only because of the increase in the number of humans but also because of the inter-dependency of rescuer policies since they coordinate as a team. In this work, humans are not advised or received explanations for the advice, however, we have built a teacher that is strong in predicting the next steps of humans. This paved the way for our major work on action advising policy transfer. Throughout experiencing transfer learning with few-short learning, we also developed interest in a more robust approach to transfer policy.

Chapter 9

An Automatic Parameter Selection Method for Network-based Transfer

Transitioning from the previous chapter, where we explored the direct transfer of model parameters in urban search and rescue scenarios, we now turn our attention to a domain where the implications of transfer learning are further nuanced by the need for selective parameter adaptation. While the direct parameter transfer offers a straightforward method of applying learned knowledge to new contexts, it often lacks the granularity necessary to understand and control the transfer effects fully. This becomes particularly critical in fields where the direct application of learned models can result in unpredictable, and sometimes non-explainable, outcomes due to the complex interplay of domain-specific variables.

In the forthcoming chapter, we address these challenges within the context of self-driving technology. Here, the necessity for domain-specific training and the high stakes associated with accurate prediction models underscore the importance of a meticulous approach to transfer learning. The chapter introduces a sophisticated method of transfer learning in self-driving vehicles, highlighting the development of an "Automatic Transfer Selector via Reinforcement Learning." This technique represents a significant advancement in the field, focusing on the selective transfer of parameters that are most relevant and beneficial to the task at hand, thereby optimizing the efficiency and efficacy of the transfer process. Unlike the broader strokes of direct parameter transfer, this method provides a more targeted and thoughtful approach to adapting pre-existing knowledge to new scenarios, reducing the reliance on extensive manual testing and adaptation. By leveraging reinforcement learning, this approach not only enhances the precision of parameter selection but also showcases the versatility and potential of reinforcement learning in crafting nuanced, application-specific transfer learning strategies [55].

9.1 Background

The prediction module in autonomous vehicles (AVs) plays a pivotal role, as it enables these AVs to anticipate the intentions of surrounding vehicles. Data gathered from perception devices is processed by this module, which typically uses a neural network to generate a label or trajectory that represents these intentions. The accuracy of this prediction is undoubtedly critical to the AV's ability to understand the environment, and serves as a prerequisite of making autonomous and correct decisions. One of the key challenges in the prediction module is the heavy reliance of prediction models on domain-specific data sets. Self-driving companies operate across various markets, each characterized by unique data domains influenced by differing traffic regulations, driving cultures, and geographical characteristics.

However, the opaque nature of neural network models makes it difficult to identify which factors specifically influence the models. Moreover, slight variations in data can significantly alter the parameters of these models. Training domain-specific models is economically burdensome but necessary for self-driving companies to ensure accuracy across different markets. Additionally, in some markets, data collection may be constrained by legal, financial, and resource-related factors, leading to data inefficiency and increasing the difficulty to develop a model specifically for this domain. Thus, a crucial focus in improving the prediction module is to optimize the utilization of existing prediction models. This means if a model can effectively generalize its predictions, successful knowledge transfer could significantly reduce the development costs associated with prediction models. However, this transfer process is complex, as models developed for specific markets are not easily transferrable due to domain specifications.

The rapid advancement of transfer learning may provide solutions to the challenges of domain-specific models. Traditional transfer learning techniques, which reuse models trained on sufficient data from a specific source domain, can reduce biases in domain features and statistics, even when the model is applied to another target domain with only a smaller and potentially less representative dataset available. At first glance, these advancements seem to solve the aforementioned problems. However, a significant roadblock arises in domain adaptation tasks: these methods typically require access to both source and target data domains, and the quantity of this data must be enough to ensure representability. This is because traditional transfer learning studies domain adaptation and domain generation, with the assumption that a classifier working well in the source dataset may also function in the target dataset. This is achieved by examining the domain differences between the two datasets or generalizing from the source dataset. Work in deep transfer learning specifically focuses on classifiers that are deep neural networks, but these, too, usually require that both source and target datasets be accessible. Generating and accessing both datasets can be expensive, and legal regulations can complicate the transfer of data across different markets. For instance, data protected in one country might not be usable in another, or due to strict regulations, no data could be exported at all. Consequently, these factors demonstrate that traditional transfer learning may not fully address the challenge of domain specificity.

A promising subfield of transfer learning, referred to as network-based transfer, provides a solution to the problem of inaccessible datasets. The basis of this approach lies in the inherent structure of neural networks, which extract different levels of features. For instance, generic features can be transferred between domains by replicating high-level parameters of certain networks. Moreover, given a small amount of data in the target domain, these networks can be fine-tuned to exceed the performance achieved by training on the target dataset from scratch. Network-based transfer eliminates the need for direct access to the source dataset or for comparing datasets, as the essential knowledge is already encoded within the network after training, and can be transferred to the new domain.

However, network-based transfer does have two major limitations. (1) Pretraining with specific features from the source domain can sometimes be misleading, resulting in performance that is even worse than training from scratch. The success of transfer largely depends on the similarity between the source and target datasets, and there's no practical way to ensure this similarity, nor do technicians know how they can identify potentially harmful features. (2) The issue of parameter selection emerges when only a few parameters are meaningful for transfer, especially in the context of deep networks. Manually searching for the optimal combination of parameters to transfer can be extremely time-consuming. Thus, developing an automatic parameter selector is crucial to optimize network-based transfer, in order to improve efficiency and facilitate a positive transfer.

In this chapter, an innovative approach is proposed to enhance network-based transfer, specifically for prediction in autonomous vehicles across international markets. The Automatic Transfer Selector via Reinforcement Learning (ATSRL) is introduced, utilizing a reinforcement learning (RL) agent to automatically select the portions of the network that should be transferred. This technique significantly enhances the efficiency of parameter selection for transfer prediction, providing a solution to the labor-intensive task of manually testing each parameter combination or enduring lengthy training periods to evaluate the impact of prediction transfer. To be more specific, a teacher neural network trained on inaccessible source data is used as the starting point, and a set of student networks that pre-load various parameter units from the teacher network is provided. The RL agent treats its current student selection as its state, the next student selection as its action, and a statistical score of a small batch of data on which the student network is trained as its reward. By treating the selection process as an RL problem, the transfer procedure is accelerated, since the most promising students are prioritized. The experimental setup involves the application of ATSRL to a temporal convolutional neural network, with the teacher's training performed on sufficient China data and transferred to the student's training on one month's US data, with the focus on specific driving scenarios. The results demonstrate significant improvement in the prediction transfer process.

In summary, this chapter makes the following key contributions:

1. Reinforcement learning (RL) is leveraged to establish an efficient and safe transfer mechanism for domain-specific prediction models in self-driving vehicles. The method is model-agnostic, implying it can be applied to any specific prediction model.
2. The proposed method is not hindered by the need to collect data from various domains; instead, it requires only the parameters of the prediction model trained with the source data. This approach enables model training even with a limited amount of self-driving data in a new market, and most importantly, the transfer procedure is both fast and safe.
3. Experiments have been conducted using a temporal convolutional neural network, an advanced model widely used in the industry. The effectiveness of the proposed method is empirically demonstrated.

A highly regarded survey paper suggests four categories of modern transfer learning: instance-based, mapping-based, network-based and adversarial-based [140]. Instance-based methods posit that partial instances from the source domain can be leveraged in the target domain by weight adjustment. For example, some researchers utilize AdaBoost-based technology to adjust weights of source domain instances, filtering out ones dissimilar to the target domain for classification [161]. This line of work has seen improvements in extensions to regression problems [58], faster algorithms [161], and more advanced metric frameworks [159]. However, it assumes access to source instances, which may not be possible when legal regulations within a market-protected self-driving data.

Similar issues arise with mapping-based methods, which rely on establishing a mapping relationship from source to target domains to a new data space with better similarity. Traditional works have applied transfer component analysis [115, 166], which has been extended to deep neural networks to develop an adaptation module [95]. However, difficulties arise when the prerequisite of statistically measuring both source and target datasets is hampered by restricted access to the source dataset. The unclear legal status of sharing source data statistics can place burden on technicians collaborating across markets, both in terms of cost and manpower.

Contrastingly, neither network-based nor adversarial-based approaches are concerned with the inaccessibility to the source dataset. The aim of both is to identify transferable components within networks. Adversarial-based approaches rely on an adversarial layer, inspired by generative adversarial nets (GAN) [49] to identify transfer components, an advancement from basic network-based approaches. However, adversarial-based approaches [44, 148], which incorporate an adversarial layer into the prediction network, substantially modify the structure of the prediction networks. Despite increasing interest, most of the experiments have been conducted using basic CNN networks [147].

This chapter proposes a method that falls into the network-based transfer category. A key challenge in this category is identifying which components are transferable. For network-based deep transfer learning, certain pre-trained network parameters from the source domain may be reused in the target domain to improve the performance. It is important to note that while some network modules are transferable, others may obstruct transferability [163]. This seminal work has motivated numerous researchers to explore the idea of transferring parts of the network, as opposed to traditional data-centric approaches. This strategy has been proven effective across various areas and is widely employed. This demonstrates neural networks are excellent feature extractors for clustering algorithms, even though the data domains may be irrelevant. Different layers extract different data representations, and researchers have used front layers to compute intermediate image representation across various datasets [112], presenting the success of finding generic features of the images. Certainly, not all network-based approaches eliminate the need for direct source data access. In some works, adaptive classifiers and transferable features are jointly learnt for both source and target datasets [94], or domain adaptation commonly employed in traditional transfer learning is utilized [173]. Despite the diverse methods offered by network-based approaches given the advancements in modern neural networks, and their ability to greatly reduce the need for source data access, these approaches come with their own limitations. Primarily, there is no systematic method to determine which network parts are beneficial for transfer. Using the basic method outlined in [163], it is extremely time-consuming to find transferable network parameters. Moreover, preventing a non-transferable part from hindering the prediction is not straightforward. A great amount of research work focuses on avoiding "negative transfer", which involves the transfer of inapplicable modules [168]. The proposed method in this chapter builds an automatic selector to ensure the efficient and positive transfer of the network.

A few works have considered automating the process of transfer. For example, there is an overlap between a subfield of meta-learning and network-based transfer learning, wherein the parameters of the network in the source domain may be used to improve the initialization of the one in the target domain. Works in meta-learning aim to learn optimal parameters for networks, with a subset considering reusing parameters from a source domain [68]. Few-shot learning is motivated by the idea of training using only a handful of samples from the target domain, but this approach is model-specific [122]. Model-Agnostic Meta-Learning (MAML) shares the closest motivation to the approach proposed in this chapter within the framework of meta learning, where researchers aim to identify parameters that are most sensitive to task changes and can largely improve the loss [40]. However, MAML makes the assumption that loading all parameters directly as an initialization is beneficial for the target task, and that the most promising parameters should be primarily improved. This approach doesn't fully address the potential for negative transfer that can occur when all parameters are copied and used as initial points. Moreover, more complex and dissimilar datasets may present greater challenges for transfer, as discussed in [23]. The experiments detailed in the renowned paper are simplistic, either focusing on object detection or Mojoco robotics, and may be impractical for real-world scenarios such as self-driving prediction data across different markets. In addition to MAML, a pairwise comparison inspired by meta-learning has been designed to determine what and where to transfer [72]. This involves using a meta-network to learn mappings and weights accordingly for transferable layers. This approach, more robust than MAML, no longer assumes that all parameters are beneficially initialized but instead attempts to select the good ones. However, this method still requires learning the results of all combinations of parameters, leading to an inefficient procedure. In contrast, the method proposed in this chapter doesn't alter the loss or experiment with all combinations of parameters to test transferrability. Instead, it judges which parts of the network would be beneficial for transfer by assigning a small score, and then proceeds with the promising candidates. This approach is safer and more efficient for transfer for real-world dataset for self-driving prediction.

Other distinct approaches to transfer learning exist. To name a few, unsupervised learning and fine-tuning of transferable base knowledge are enabled through multi-scale convolutional sparse coding [25]. Furthermore, RL algorithms have been utilized in the design of neural networks [9], where a learning agent explores architectures to enhance performance. However, the method proposed in this chapter, while echoing the idea of RL in architecture design, distinguishes itself as the first to employ RL in selecting transferable parameters specifically for complex, real-world datasets, such as self-driving data, in a safe and efficient manner.

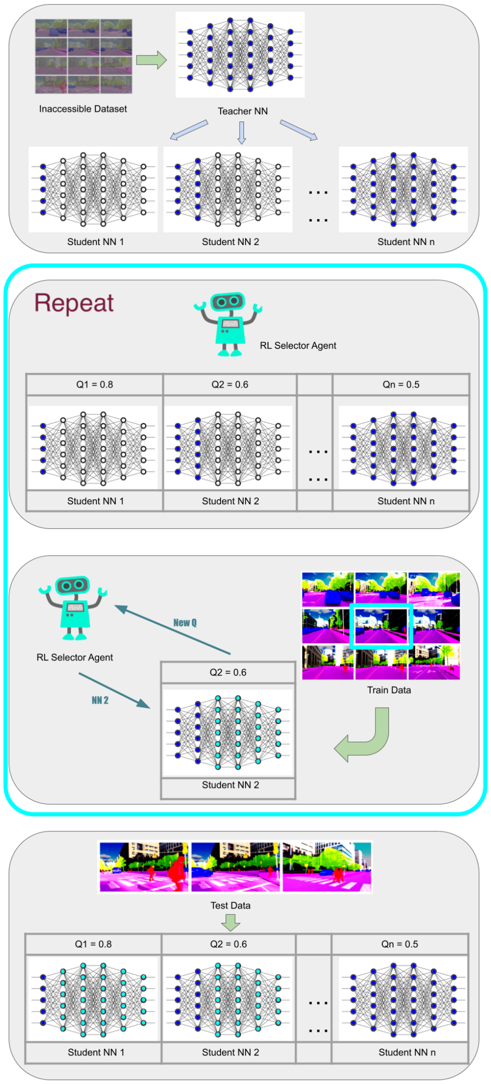


Figure 9.1: A visualization of the procedure.

9.2 Methods

A high-level overview of our proposed method, Automatic Transfer Selector via Reinforcement Learning (ATSRL) is shown in Figure 9.1. The first block provides a visualization of the initialization process, the recurring second and third blocks represent the core procedure for selecting which models to train, and the final block concludes the procedure by inputting the test data into the various trained model candidates.

Initialization We assume there is a Teacher Neural Network $\mathcal{N}_T$ trained from a dataset that is no longer accessible in the current environment due to legal constraints or technical limitations. However, the features and the network structures are known. Thus we initialize n student neural networks identical to the teacher's, where n is the number of parameter units that we apply freezing or pretrain-finetuning to. These parameter units can be a single neuron, a layer, or multiple layers of parameters appropriate for the specific structure of the network. For each student $\mathcal{N}_{Si}$, we copy the parameter units from 0 to i from the teacher $\mathcal{N}_T$.

Select a Student We model the student selection process as an RL problem. We use the following MDP definitions to represent the RL selector agent:

1. $\mathcal{S}$: The current student $\mathcal{N}_{Si}$ being trained
2. $\mathcal{A}$: $\{i+1$ or $i-1\}$
3. $\mathcal{R}$: f_1 score of $\mathcal{N}_{Si}$ given a batch of Train Data D_t
4. $\mathcal{T}$: Prob$(s'|s = \mathcal{N}_{Si}, a = i \pm 1)$
5. γ: 1.0

Our RL selector agent treats the selection process as an RL mission. Each state $s \in \mathcal{S}$ represents the current student being selected for training. A value (V or Q, depending on the implemented RL algorithm) provides the estimated cumulative reward the state can achieve. Given the updated value table, the selector agent chooses an action $a \in A$ that decides what should be the next state, i.e. the next student to be trained. The next student is either the proceeding or the succeeding student with respect to the current student, depending on ± 1. The reward is the f_1 score the student receives from training a batch of data from the train dataset D(the tth batch). The transition function is set to be deterministic, and the decay rate is 1.0, i.e. a decay over longer episodes is not considered because it does not affect the outcome.

Update a Student After the selector agent enters the current state, i.e. selects which student $\mathcal{N}_{Si}$, the student NN is updated. For example, in the third block of Figure 9.1, student 2 is selected. It is given a small batch of training and validation data from the entire training dataset. The network of student 2 could either freeze the first 2 parameter units or apply pretrain-finetuning to them. Regardless, the remaining parameter units are updated, and thus, with the f_1 score (or other meaningful statistics), the selector agent also updates the corresponding value of student 2. Then the selector agent repeats the process to find the next student to train.

Test all Students After the RL selector agent has completed the training for each of the student networks, all the students are tested with the testing data. The best model among all the candidates is then selected as the final output student model.

Complexity Analysis The time complexity of the ATSRL method can be mainly attributed to the number of student networks and the training process for each of these networks. In the worst-case scenario, if we consider each student network to be trained independently, the time complexity is proportional to $O(n \times T)$, where n is the number of student networks, and T is the time required to train each network. However, the actual runtime could be considerably less, as the process of training student networks may be optimized by transfer learning from the teacher network and from each other, and only the most promising student is trained fully with partial training data. The space complexity, on the other hand, depends on the number of student networks and the size of each network, and can be approximated as $O(n \times M)$, where M is the memory requirement for each network.

f_1 **Score Briefing** The f_1 score is a crucial statistic in binary classifications, representing a balance between precision (the ratio of true positives to all predicted positives) and recall (the ratio of true positives to all actual positives). Calculated as the harmonic mean of precision and recall, the f_1 score ranges from 0 to 1, with 1 indicating perfect precision and recall, and 0 showing a total absence of either. Though other statistics can be considered as needed, depending on the characteristics of the data, the f_1 score is most widely utilized in real-world datasets.

Algorithm 9 Automatic Transfer Selector via Reinforcement Learning (ATSRL)

```
Input: Train Data $D = \{D_1, D_2 ... D_m\}$ where $m$ is the number of fold $D$ is split, Teacher NN $\mathcal{N}_T$
Initialize $V(s_i)$ or $Q(s_i, a)$, $s_{t0}$, $f_1\_best$, Student NN $\mathcal{N}_{Si}, i \in \{1, 2 ... n\}$
                                        ▷ with $1, 2 ... i$ parameter units copied from $\mathcal{N}_T$ to $\mathcal{N}_{Si}$
(optional) lock $1, 2 ... i$ parameter units of $\mathcal{N}_{Si}$
while $D_t \neq \phi$ do
    $D_t \leftarrow D_j \in D$
    $f_1 \leftarrow$ Train($\mathcal{N}_{St}, D_t$)
    Update $(Q(s_t), f_1)$
    if $f_1 > f_1\_best$ then
        save $\mathcal{N}_{St}$
    end if
    $a_t \leftarrow \max_a Q(s_t, a)$
    $s_{t+1} \leftarrow \mathcal{T}(s_t, a_t)$
end while
```

The pesudocode is presented more formally in Algorithm 9. The algorithm's inputs include the training data $D = \{D_1, D_2 ... D_m\}$ and the teacher network $\mathcal{N}_T$. The training data D is divided into smaller batches, each used for fine-tuning in each step. The current state s_i represents the current student network $\mathcal{N}_{Si}$ being trained, requiring respective initializations of their values. s_{t0} denotes the initial student selected, and f_1_best is a list of the best observed f_1 scores or other relevant statistics for each student. Besides, parameter units are transferred from $\mathcal{N}_T$ to $\mathcal{N}_{Si}$.

At each time step, a batch from the training data is selected. Repeated traversals over the data are permissible as student models may encounter the same batch of data multiple times. The selected student is trained with this data batch, and the resulting f_1 score is recorded. This f_1 score enables the update of the current state's value. If the score outperforms previously observed scores for this state, this model $\mathcal{N}_{St}$ is saved. Finally, the action and the next state are selected based on the current value and transition.

9.3 Experiments

The data utilized for the experiment is a month's of truck driving data from the United States, specifically related to the cutting-in scenario. This data is divided into training + validation, and testing datasets at a ratio of $0.8 : 0.2$. The input data consists of a sequence of 21 features of length 10, processed for the cutting-in scenario. The output data is a binary integer, indicating whether or not cutting-in occurs.

The temporal convolutional neural network (TCN) [8] served as the prediction model. TCN creates a hierarchical structure that integrates temporal features into a convolutional neural network. Each internal node of the tree-like structure is a residual block, composed of multiple layers of dilated convolution, activation, and normalization. Due to this structure, the initial consideration in the experiment was to freeze or pretrain residual blocks instead of individual layers. The TCN used in this case has a depth of 4, signifying the presence of 4 residual blocks plus extraneous parameters that could be selected as the transfer target.

A model was trained using the training + validation data and was denoted as ScratchUS. Additionally, a prediction model trained in China was obtained. Although the features were identical, the data from China was inaccessible for the experiment. This model was referred to as TeacherCN. The goal was to leverage both TeacherCN and the existing US training + validation data to create a new transferred model that would surpass ScratchUS in performance on the testing data. The chosen metrics were the 5 most important statistics in self-driving prediction:

1. Lane Change False positive Rate (FPR): FPR of accurately predicting the lane change
2. Lane Change Precision: precision of lane change
3. Lane Change Recall: recall of lane change
4. 05 Frame Early Rate: prediction correctness using the 5 latest frames
5. 10 Frame Early Rate: prediction correctness using the 10 latest frames

False Positive Rate (FPR) is a critical statistic in the context of lane change predictions in autonomous driving. It measures the percentage of instances in which a lane change is incorrectly predicted, that is, when the prediction system inaccurately indicates that a lane change will occur when, in fact, it does not. Minimizing FPR is crucial because an excessive number of false alarms can lead to unnecessary or inappropriate responses from the vehicle, potentially compromising safety and operational efficiency.

All subsequent experiments were conducted on the same testing data, utilizing the above statistics. A total of 16382 data points were used for testing. For convenience in visualization, 1 - FPR was displayed instead of FPR. These statistics are empirically informative for the other teams in the general planning-control-prediction-fuel-economy group at PlusAI, and are thus more meaningful than other indicators such as validation loss.

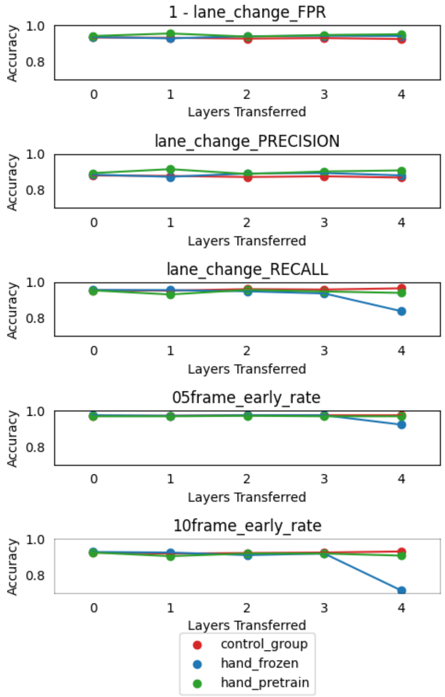


Figure 9.2: Compare freezing and pretraining residual blocks by hand.

9.3.1 Traditional Network Transfer By Hand

The manual freezing and pretraining of residual blocks was first examined. The result is shown in Figure 9.2. The control group (red dots) utilized the model directly trained by the training set of data (ScratchUS) to investigate potential co-adaptive features. The results revealed that as residual block 0, residual block 01, residual block 012, residual block 0123, and all parameters were gradually frozen, the red dots remained nearly unchanged. This suggested limited co-adaptive features that might impact the result.

Subsequently, TeacherCN was used as the teacher model, and the same manual freezing procedure was repeated (blue dots). Almost every freezing procedure remains unchanged except when all parameters were frozen (with x index at 4). In this case, it was inferred that all parameters of residual blocks are transferable except for the remaining parameters. This further suggested that TeacherCN was not directly applicable to the data collected in the US as the lane change recall and 10 frame early rate were extremely poor.

Manual pretraining with TeacherCN (green dots) served as a sanity check. As expected, it performed slightly better than the control group and manual freezing of residual blocks. Notably, pretraining and finetuning with all parameters again proved to be an exception (with x index at 4), which could also be inferred from the blue dots. For other transfer options, performance was slightly better than the control group. The rest of the experiment aimed to enhance this advantage using the proposed methods.

Each point on the plot represented an independent training procedure, with the training + validation data divided into 3 folds, each taken by each training. 50 episodes were assigned for each fold to ensure convergence. All training procedures used the same 3 folds.

9.3.2 RL Variations Trained Over Time

Figure 9.3 illustrated how 5 important statistics changed over time, from iterating 60 folds of data to 150 folds. The entire training + validation data was divided into 30 folds, representing the performance for 2 and 5 iterations of traversing the entire data. As the number of folds increased, fewer data entries were contained in each fold, and convergence time was accordingly shortened to 30 episodes for each fold. Moreover, in contrast to providing the same 3 folds to all the training procedures, only the model represented by the current state was given one of the 30 folds of data, and trained to convergence at a time.

Three RL algorithm variations were experimented with: TD(0) (in orange), SARSA (in green), and DQN (in blue). Similar to Figure 9.2, but reversed, the y-axis displayed the transferred residual blocks, and the x-axis presented the accuracy. Darker colors represented 60 folds while the lighter colors represented 150 folds.

For the lane change FPR, regardless of what residual blocks were transferred, DQN clearly performed better than the other two. Increasing the number of folds did not necessarily improve the FPR. For TD and SARSA, the difference was more pronounced. This was also similar for lane change precision where DQN outperformed the other two and showed smaller improvement over time. This inferred that DQN trained the fastest and best and no additional folds were necessary. For the remaining three statistics of lane change recall, 05 frame early rate, and 10 frame early rate, DQN behaved in a general manner. Small variances were observed among the specific residual blocks selected for transfer according to those statistics.

Interestingly, though SARSA selected actions based on the estimation of Q values, it did not demonstrate more advantage than TD(0) where the selection process was largely random. It could be concluded that DQN performed better than the other two variations, but SARSA did not seem to be more advantageous as a general observation. However, the best model trained by SARSA outperformed TD(0), as will be shown in the next sub-section.

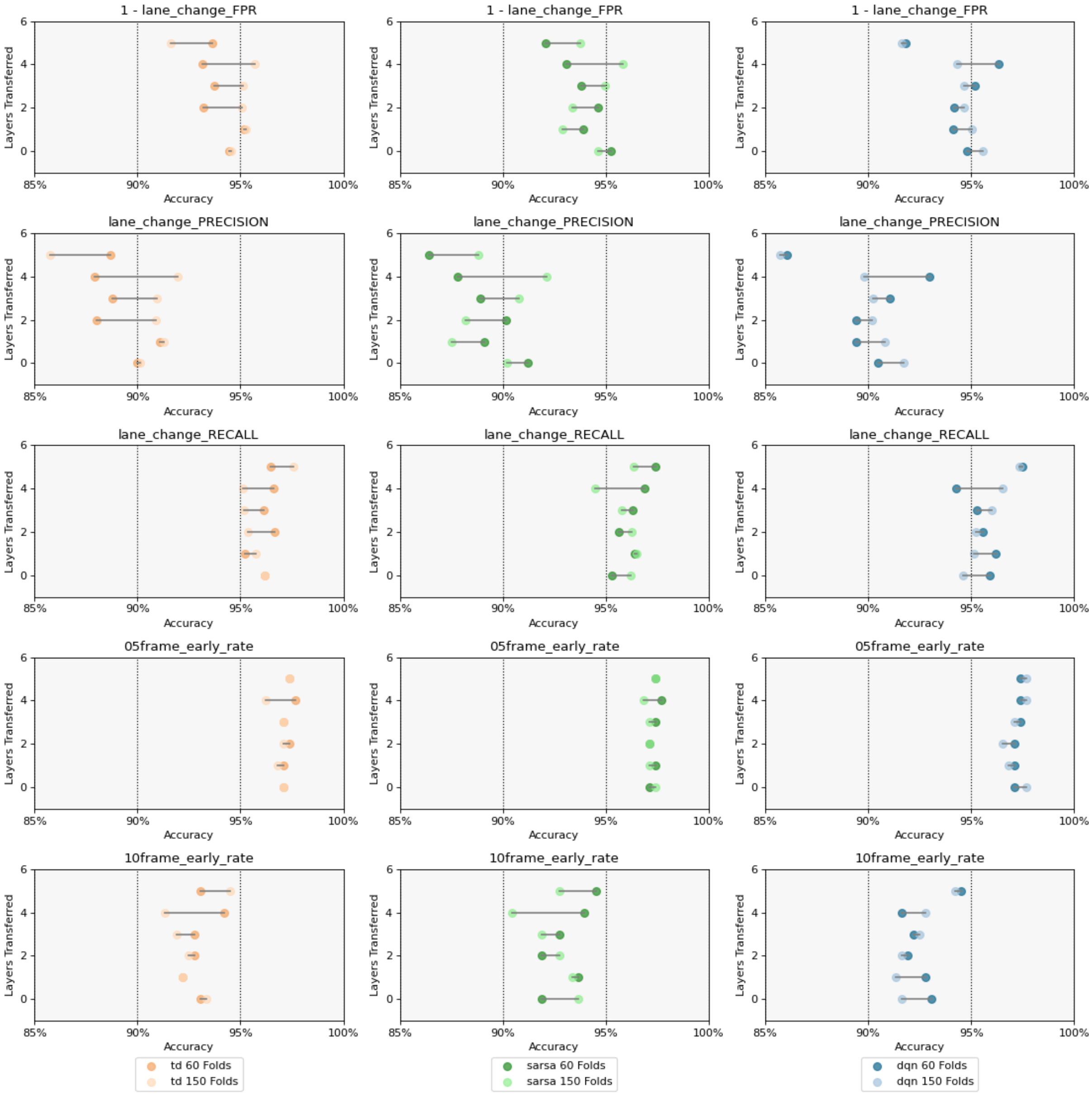


Figure 9.3: Compare RL variations with 60 folds and 150 folds of data to train.

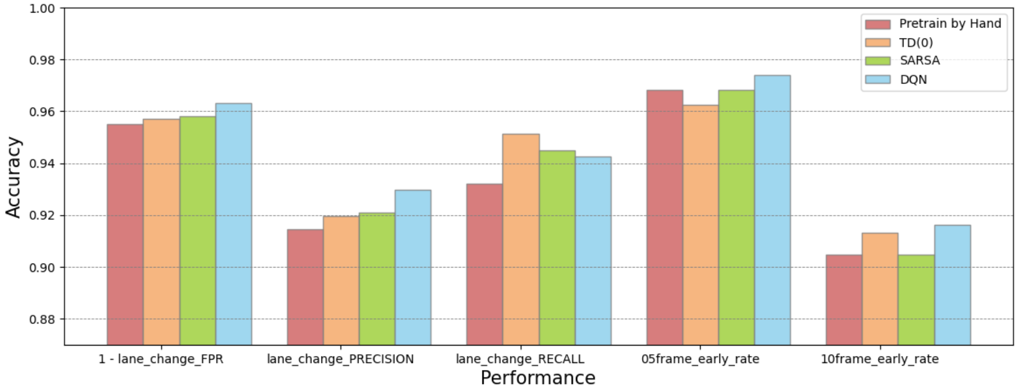


Figure 9.4: Compare best RL variations.

9.3.3 Best Models of RL Variations

From Figure 9.2 and Figure 9.3, it might be difficult to define what constitutes the best model, since there were five statistics of interest. A high early rate might also correlate with a high FPR rate, making it challenging to decide which one should be sacrificed for the trade-off.

However, from Figure 9.3, it was also observed that pretraining and finetuning with residual blocks of 0123 resulted in the lowest FPR, highest precision, and 05 frame early rate, while at the same time the other 2 statistics were within the top 3. This observation was consistent with Figure 9.2, though the advantage was small. It was inferred that the algorithm designed with RL agents was capable of pointing out the best candidate as well.

In this case, transferring residual blocks of 0123 was treated as the best, and its RL variations and manual pretrain were compared in Figure 9.4. Notably, DQN (in blue) was found to be optimal as it achieved the highest number among four in all statistics except for lane change recall. TD(0) and SARSA showed alternate advantages over each other, generally doing a better job than pretraining manually. One possible explanation was that multiple small batches allowed for better finetuning. However, the primary focus was not on surpassing the manual pretraining, but on improving the auto-selection process.

In summary, the DQN transfer model with residual blocks of 0123 was retained as the best, and further compared with a layer-level selection model in the next sub-section.

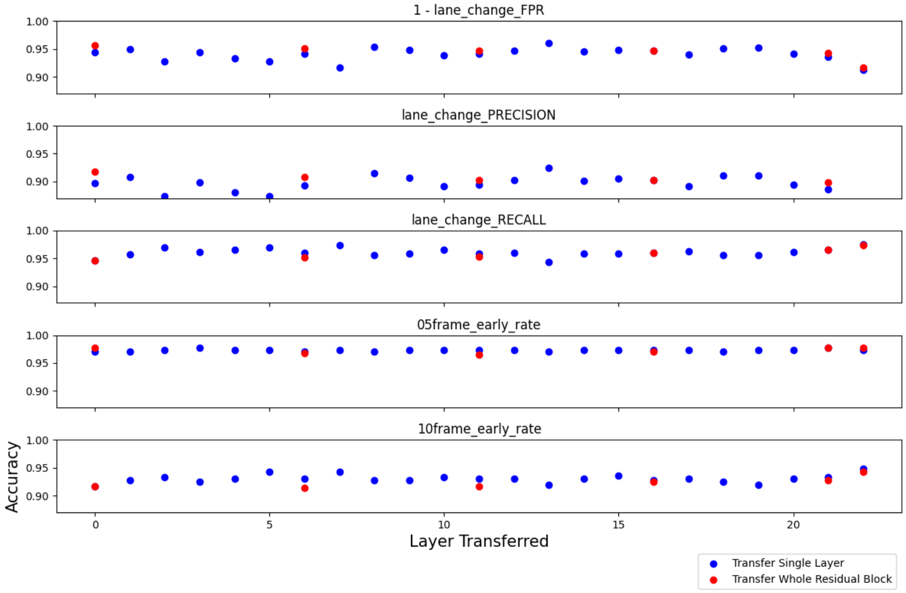


Figure 9.5: Layer level Transfer

9.3.4 Layer-level Transfer

Up until now, pre-training and finetuning was performed with the parameters of entire residual blocks - one reason was due to the structure of TCN, and the second was that traditional manual pretraining and finetuning made it too laborious to approach optimal transfer by hand. The results in the previous sub sections validated the methodology, leading to exploration if pre-training and finetuning with parameters of single layers could yield even better results. Additionally, the number of folds was increased to 50, while the episode of each fold was decreased to 20 to achieve convergence, and similarly each time only the model with specific layers transferred was being trained.

In line with transferring by residual blocks, layer-level transfer referred to transferring parameters of the 1st, 1st + 2nd, 1st + 2nd + 3rd, ... until all the parameters. Therefore, with some layers transferred, they were exactly the same as transferring the residual blocks as previous. In the results shown in Figure 9.5, the x axis referred to layers transferred, and the y axis showed the accuracy. The blue dots and red dots denoted transferring the single layer or whole residual blocks respectively, where they had the same x axis, it meant the corresponding residual blocks composed of the layers.

Most red dots were close to the blue dots representing the same layers to transfer. To clarify, there was no interest in boosting the corresponding blue dots to perform significantly better than the red ones. It was anticipated that the slight difference was due to training variance. The goal of this experiment was to find out if there were candidate layers that could be transferred and outperform the existing performance, i.e., could any blue dots be found that were better among all candidates of blue and red dots. The answer was affirmative, as readers could easily observe that red dots were never the optimal among the 5 statistics. Therefore, with the proposed algorithm, it was claimed that the selection became more efficient for this transfer model.

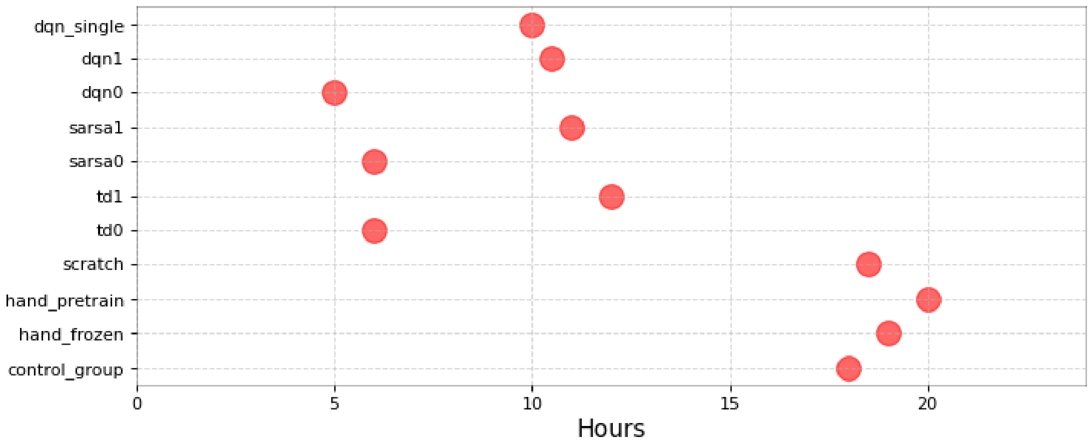


Figure 9.6: Runtime of the Experiments

9.3.5 Runtime Comparison

A comparison of the runtime of the experiments was also performed, as shown in Figure 9.6. One significant challenge with manual pretraining and finetuning is its labor-intensive nature. Selecting with RL agents can save time because the advantage of neural networks is that finetuning a small amount of data can significantly enhance performance. With less data, training converges faster and the use of various batches of data prevents overfitting.

It's worth noting that the result was empirical, with statistics provided in the previous sections. In this comparison, "dqn0" and "dqn1" referred to the DQN selector with 60 and 150 folds, respectively, and the same applied to "sarsa" and "td". The result showed that with the same amount of data used for training (5 iterations traversing the entire training + validation data, with various batch size), the DQN selector with the single layer transferred was the most efficient.

9.4 Conclusion

This chapter introduced innovative methods of Automatic Transfer Selector via Reinforcement Learning applied to the self-driving prediction model domain. There was a successful use of a network trained in China, despite no access to the China data, to assist the learning of a prediction network for US data. This process proved more efficient and effective than manual training, and it also surpassed pre-training and fine-tuning conducted solely using US data.

Future work could expand the current scenario to include not only cut-in but also cut-out and merge. The claim was also made that the algorithm is not limited to the temporal convoluntional network. While it is is the most recent module in the company's infrastructure and hence served as the test-bed, future plans include extending to newer models as they become available, and making further comparisons. Furthermore, plans to perform experiments with a teacher model either from the US or China are in place to assist students with a small amount of Germany data. This will likely prove to be a more challenging task given the differences in traffic rules and culture between Germany and both China and the US.

The focus of this work was largely practical, aimed at improving the existing architecture. Thus, extensive comparison with a broad range of network-based transfer learning methods was not performed. Additionally, the most advanced RL algorithms were not considered in the study, as the aim was to validate the practicability of the current approach. These represent the limitations and two key areas for further exploration in future work.

Chapter 10

Conclusion

In this thesis, we have demonstrated that the knowledge transferring process between reinforcement learning agents in the form of teacher-student action advising can significantly expedite the learning process and enhance overall performance. Our work has discussed the limitations of conventional reinforcment learning techniques in dealing with the inherent complexity and sample inefficiency of these tasks, while highlighting the potential of teacher-student action advising strategies in a few directions.

In addition to identifying the benefits of action advising, we have also proposed innovative enhancements to the teacher-student framework. These include allowing the teacher to explain the rationale behind its advice, equipping students with the ability to determine reliability of the given advice, developing a feedback system to help teachers identify and filter out poor advice, and additionally enabling teachers to offer alternative advice that is more catered to students' needs. Our methods have been thoroughly tested and documented in individual chapters, and the positive outcomes substantiate the validity and effectiveness of our approach.

In our future work, we aim to extend the application of our findings in more complex scenarios involving heterogeneous multi-agent teams. We plan to let the teacher guide these diverse student agents towards achieving collective goals. Furthermore, we aim to investigate dynamic peer-to-peer advising, enabling agents to switch roles between teacher and student based on their situational knowledge and experience.

The studies conducted and the future directions outlined herein have the potential to significantly enhance the existing frameworks for teacher-student action advising. They pave the way for more efficient learning processes, improved performance in high-dimensional tasks, and a greater degree of collaboration and knowledge exchange in the context of reinforcement learning. By continuing to investigate and enhance these techniques, we can unlock a vast number of possibilities for reinforcement learning and multi-agent systems.

Bibliography

[1] Knowledge-grounded self-rationalization via extractive and natural language explanations. 2022. 7.1

[2] Gpt-4, Mar 2023. URL https://openai.com/research/gpt-4. 7.3

[3] Jose M Alonso, Alejandro Ramos-Soto, Ehud Reiter, and Kees van Deemter. An exploratory study on the benefits of using natural language for explaining fuzzy rule-based systems. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2017. 7.1

[4] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara Grosz. Interactive teaching strategies for agent training. 2016. 1.2

[5] Ofra Amir, Finale Doshi-Velez, and David Sarne. Summarizing agent strategies. *Autonomous Agents and Multi-Agent Systems*, 33:628–644, 2019. 7.1

[6] Daksh Anand, Vaibhav Gupta, Praveen Paruchuri, and Balaraman Ravindran. An enhanced advising model in teacher-student framework using state categorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6653–6660, 2021. 1.2

[7] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27, 2014. 7.2.1

[8] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. 9.3

[9] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2016. 9.1

[10] Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. Unsupervised domain adaptation by domain invariant projection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 769–776, 2013. 8.1

[11] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016. 8.1

[12] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference*

on Machine Learning, pages 501–510. PMLR, 2018. 8.1

[13] André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020. 8.1

[14] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via model extraction. *arXiv preprint arXiv:1706.09773*, 2017. 7.2.1

[15] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *arXiv preprint arXiv:1805.08328*, 2018. 2.1

[16] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018. 7.2.1

[17] Sarthak Bhagat, Simon Stepputtis, Joseph Campbell, and Katia Sycara. Sample-efficient learning of novel visual concepts. In *Conference on Lifelong Learning Agents*. PMLR, 2023. 7.1

[18] Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models. *URL https://openaipublic. blob. core. windows. net/neuron-explainer/paper/index. html.(Date accessed: 14.05. 2023)*, 2023. 7.1

[19] Azzedine Boukerche and Jiahao Wang. Machine learning-based traffic prediction models for intelligent transportation systems. *Computer Networks*, 181:107530, 2020. 7.1

[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 7.1

[21] Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. *Advances in Neural Information Processing Systems*, 31, 2018. 7.1

[22] Joseph Campbell and Katsu Yamane. Learning whole-body human-robot haptic interaction in social contexts. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10177–10183. IEEE, 2020. 7.1

[23] Joseph Campbell, Yue Guo, Fiona Xie, Simon Stepputtis, and Katia Sycara. Introspective action advising for interpretable transfer learning. In *Conference on Lifelong Learning Agents*, 2023. 1.2, 1.3, 1.4, 4, 5, 5.1, 5.3.2, 7.1, 9.1

[24] Adrijana Car, George Taylor, and Chris Brunsdon. An analysis of the performance of a hierarchical wayfinding computational model using synthetic graphs. *Computers, environment and urban systems*, 25(1):69–88, 2001. 8.1

[25] Hang Chang, Ju Han, Cheng Zhong, Antoine M Snijders, and Jian-Hua Mao. Unsupervised transfer learning via multi-scale convolutional sparse coding for biomedical applications. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1182–1194, 2017. 9.1

[26] Michelene TH Chi, Nicholas De Leeuw, Mei-Hung Chiu, and Christian LaVancher. Elicit-

ing self-explanations improves understanding. *Cognitive science*, 18(3):439–477, 1994. 2

[27] Elizabeth R Chrastil and William H Warren. From cognitive maps to cognitive graphs. *PloS one*, 9(11):e112544, 2014. 8.1

[28] Christian Arzate Cruz and Takeo Igarashi. Interactive explanations: Diagnosis and repair of reinforcement learning based agent behaviors. In *2021 IEEE Conference on Games (CoG)*, pages 01–08. IEEE, 2021. 7.1

[29] Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *The 22nd international conference on artificial intelligence and statistics*, pages 1331–1340. PMLR, 2019. 4.2.1, 4.2.2, 5.3.2

[30] Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64: 645–703, 2019. 1.2

[31] Felipe Leno Da Silva, Ruben Glatt, and Anna Helena Reali Costa. Simultaneously learning and advising in multiagent reinforcement learning. In *Proceedings of the 16th conference on autonomous agents and multiagent systems*, pages 1100–1108, 2017. 1.2, 2.3.1, 5.1

[32] Felipe Leno Da Silva, Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Uncertainty-aware action advising for deep reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5792–5799, 2020. 1.1, 1.2

[33] Felipe Leno Da Silva, Garrett Warnell, Anna Helena Reali Costa, and Peter Stone. Agents teaching agents: a survey on inter-agent transfer learning. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–17, 2020. 1.1

[34] Quanyu Dai, Xiao Shen, Xiao-Ming Wu, and Dan Wang. Network transfer learning via adversarial domain adaptation with graph convolution. *arXiv preprint arXiv:1909.01541*, 2019. 8.1

[35] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2019. 7.1

[36] Sean C Duncan. Minecraft, beyond construction and survival. 2011. 8.1

[37] Upol Ehsan, Pradyumna Tambwekar, Larry Chan, Brent Harrison, and Mark O Riedl. Automated rationale generation: a technique for explainable ai and its effects on human perceptions. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 263–274, 2019. 7.1

[38] Anestis Fachantidis, Matthew E Taylor, and Ioannis Vlahavas. Learning to teach reinforcement learning agents. *Machine Learning and Knowledge Extraction*, 1(1):21–42, 2019. 1.2

[39] Meherwar Fatima and Maruf Pasha. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications*, 9(01):1–16, 2017. 7.1

[40] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017. 9.1

[41] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 1.3, 2.3.1

[42] Jared T Freeman, Lixiao Huang, Matt Woods, and Stephen J Cauffman. Evaluating artificial social intelligence in an urban search and rescue task environment. In *AAAI 2021 Fall Symposium, 4-6 Nov 2021*. AAAI, 2021. 2.3.1, 5.3.1, 5.5.3

[43] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017. 7.2.1

[44] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016. 9.1

[45] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018. 1.1

[46] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018. 2.1

[47] Dimitra Gkatzia, Oliver Lemon, and Verena Rieser. Natural language generation enhances human decision-making with uncertain information. *arXiv preprint arXiv:1606.03254*, 2016. 7.1

[48] Yongxi Gong, Yu Liu, Jian Yang, and Guicai Li. Structural hierarchy of spatial knowledge based on landmarks and its application in locality descriptions. In *2010 18th International Conference on Geoinformatics*, pages 1–5. IEEE, 2010. 8.1

[49] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 9.1

[50] Jaskaran Grover, Yiwei Lyu, Wenhao Luo, Changliu Liu, John Dolan, and Katia Sycara. Semantically-aware pedestrian intent prediction with barrier functions and mixed-integer quadratic programming. *IFAC-PapersOnLine*, 55(41):167–174, 2022. 1.1

[51] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. Xai—explainable artificial intelligence. *Science robotics*, 4(37):eaay7120, 2019. 7.1

[52] Wenbo Guo, Xian Wu, Usmann Khan, and Xinyu Xing. Edge: Explaining deep reinforcement learning policies. *Advances in Neural Information Processing Systems*, 34: 12222–12236, 2021. 7.1

[53] Yue Guo, Rohit Jena, Dana Hughes, Michael Lewis, and Katia Sycara. Transfer learning for human navigation and triage strategies prediction in a simulated urban search and

rescue task. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 784–791. IEEE, 2021. 2.3.1, 5.5.3, 8

[54] Yue Guo, Joseph Campbell, Simon Stepputtis, Ruiyu Li, Dana Hughes, Fei Fang, and Katia Sycara. Explainable action advising for multi-agent reinforcement learning. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023. 1.2, 1.4, 2, 3, 4.1.1, 5, 5.1, 5.5.3

[55] Yue Guo, Yu Wang, I-Hsuan Yang, and Katia Sycara. Reinforcement learning methods for network-based transfer parameter selection. *Intelligence & Robotics*, 3(3):402–419, 2023. 9

[56] Yue Guo, Xijia Zhang, Simon Stepputtis, Joseph Campbell, and Katia Sycara. Adaptive action advising with different rewards. In *Conference on Lifelong Learning Agents*, 2024. 5

[57] Nikunj Gupta, G Srinivasaraghavan, Swarup Kumar Mohalik, and Matthew E Taylor. Hammer: Multi-level coordination of reinforcement learning agents via learned messaging. *arXiv preprint arXiv:2102.00824*, 2021. 1.2

[58] Shrey Gupta, Jianzhao Bi, Yang Liu, and Avani Wildani. Boosting for regression transfer via importance sampling. *International Journal of Data Science and Analytics*, pages 1–12, 2023. 9.1

[59] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. 5.5.1

[60] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*. 5.5.1

[61] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, volume 3. sn, 2016. 2.3.1

[62] Bradley Hayes and Julie A Shah. Improving robot controller transparency through autonomous policy explanation. In *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*, pages 303–312, 2017. 7.1

[63] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 7.2.1

[64] Stephen C Hirtle and John Jonides. Evidence of hierarchies in cognitive maps. *Memory & cognition*, 13(3):208–217, 1985. 8.1

[65] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016. 1.1

[66] Shengran Hu and Jeff Clune. Thought cloning: Learning to think while acting by imitating human thinking. *arXiv preprint arXiv:2306.00323*, 2023. 7.1

[67] L Huang, J Freeman, N Cooke, M Cohen, X Yin, J Clark, M Wood, V Buchanan, C Carrol, F Scholcover, A Mudigonda, L Thomas, A Teo, M Freiman, J Colonna-Romano, L Lapu-

jade, and K Tatapudi. Using humans' theory of mind to study artificial social intelligence in minecraft search and rescue. In *(to be submitted to the) Journal of Cognitive Science*, 2021. (document), 8.1, 8.3, 8.3

[68] Mike Huisman, Jan N Van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541, 2021. 9.1

[69] Ercüment Ilhan, Jeremy Gow, and Diego Perez-Liebana. Teaching on a budget in multi-agent deep reinforcement learning. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019. 1.2

[70] Ercument Ilhan, Jeremy Gow, and Diego Perez-Liebana. Action advising with advice imitation in deep reinforcement learning. *arXiv preprint arXiv:2104.08441*, 2021. 1.2

[71] Vidhi Jain, Rohit Jena, Huao Li, Tejus Gupta, Dana Hughes, Michael Lewis, and Katia Sycara. Predicting human strategies in simulated search and rescue task. In *Artificial Intelligence for Humanitarian Assistance and Disaster Response Workshop, NeurIPS*, 2020. 8.1

[72] Yunhun Jang, Hankook Lee, Sung Ju Hwang, and Jinwoo Shin. Learning what and where to transfer. In *International conference on machine learning*, pages 3030–3039. PMLR, 2019. 9.1

[73] John DeNero, Dan Klein, Ketrina Yim, and Pieter Abbeel. UC Berkeley CS188 intro to AI. University of California, Berkeley. 2.3.1

[74] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247. Citeseer, 2016. 8.1

[75] W Lewis Johnson. Agents that learn to explain themselves. In *AAAI*, pages 1257–1263. Palo Alto, CA, 1994. 7.1

[76] K-Team. Khepera iv robot. https://www.wevolver.com/specs/khepera.iv, 2021. 5.5.3

[77] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In *Robot soccer world cup*, pages 72–85. Springer, 2006. 2.3.1

[78] Daniel Kasenberg, Antonio Roque, Ravenna Thielstrom, Meia Chita-Tegmark, and Matthias Scheutz. Generating justifications for norm-related agent decisions. *arXiv preprint arXiv:1911.00226*, 2019. 7.1

[79] Robert Kass, Tim Finin, et al. The need for user models in generating expert system explanations. *International Journal of Expert Systems*, 1(4), 1988. 7.1

[80] Dong-Ki Kim, Miao Liu, Shayegan Omidshafiei, Sebastian Lopez-Cot, Matthew Riemer, Golnaz Habibi, Gerald Tesauro, Sami Mourad, Murray Campbell, and Jonathan P How. Learning hierarchical teaching policies for cooperative agents. *arXiv preprint arXiv:1903.03216*, 2019. 1.2

[81] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021. 1.1

[82] Gregory Kuhlmann and Peter Stone. Graph-based domain mapping for transfer learning in general games. In *European Conference on Machine Learning*, pages 188–200. Springer, 2007. 8.1

[83] Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *The AAAI-2004 workshop on supervisory control of learning and adaptive systems*. San Jose, CA, 2004. 1.2, 5.1

[84] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 1.1

[85] Jaekoo Lee, Hyunjae Kim, Jongsun Lee, and Sungroh Yoon. Transfer learning for deep learning on graph-structured data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 8.1

[86] Michael Lewis, Katia Sycara, and Illah Nourbakhsh. Developing a testbed for studying human-robot interaction in urban search and rescue. In *Human-Centered Computing*, pages 270–274. CRC Press, 2019. 2.3.1, 5.3.1, 5.5.3

[87] Shiyang Li, Jianshu Chen, Yelong Shen, Zhiyu Chen, Xinlu Zhang, Zekun Li, Hong Wang, Jing Qian, Baolin Peng, Yi Mao, et al. Explanations from large language models make small reasoners better. *arXiv preprint arXiv:2210.06726*, 2022. 7.1

[88] Shuai Li, Azarakhsh Keipour, Kevin Jamieson, Nicolas Hudson, Charles Swan, and Kostas Bekris. Large-scale package manipulation via learned metrics of pick success. *arXiv preprint arXiv:2305.10272*, 2023. 7.1

[89] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017. 8.1, 8.3.1

[90] Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*, pages 414–429. Springer, 2019. 7.1

[91] Nengbao Liu, Vahan Babushkin, and Afshin Afshari. Short-term forecasting of temperature driven electricity load using time series and neural network model. *Journal of Clean Energy Technologies*, 2(4):327–331, 2014. 8.3.2

[92] Tongtong Liu, Joe McCalmon, Thai Le, Md Asifur Rahman, Dongwon Lee, and Sarra Alqahtani. A novel policy-graph approach with natural language and counterfactual abstractions for explaining reinforcement learning agents. *Autonomous Agents and Multi-Agent Systems*, 37(2):34, 2023. 7.1

[93] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013. 8.1

[94] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain

adaptation with residual transfer networks. *Advances in neural information processing systems*, 29, 2016. 9.1

[95] Mingsheng Long, Yue Cao, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Transferable representation learning with deep adaptation networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3071–3085, 2018. 9.1

[96] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562, 2018. 4.2, 5, 5.3, 5.3.1

[97] Richard Maclin and Jude W Shavlik. *Incorporating advice into agents that learn from reinforcements*. University of Wisconsin-Madison. Computer Sciences Department, 1994. 1.2, 5.1

[98] Richard Maclin and Jude W Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3):251–281, 1996. 1.2, 5.1

[99] Tamas Madl, Stan Franklin, Ke Chen, Robert Trappl, and Daniela Montaldi. Exploring the structure of spatial representations. *PloS one*, 11(6):e0157343, 2016. 8.1

[100] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. *Transportation Research Record*, 2674(9):473–488, 2020. 8.1

[101] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. Transfer learning with graph neural networks for short-term highway traffic forecasting. *arXiv preprint arXiv:2004.08038*, 2020. 8.1, 8.2.3, 8.3.1

[102] Ana Marasović, Chandra Bhagavatula, Jae sung Park, Ronan Le Bras, Noah A. Smith, and Yejin Choi. Natural language rationales with full-stack visual reasoning: From pixels to semantic frames to commonsense graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2810–2829. Association for Computational Linguistics, November 2020. 7.1

[103] Ana Marasović, Iz Beltagy, Doug Downey, and Matthew E Peters. Few-shot self-rationalization with natural language prompts. *arXiv preprint arXiv:2111.08284*, 2021. 7.1, 7.1

[104] Ettore Mariotti, Jose M Alonso, and Albert Gatt. Towards harnessing natural language generation to explain black-box models. In *2nd Workshop on Interactive Natural Language Technology for Explainable Artificial Intelligence*, pages 22–27, 2020. 7.1

[105] Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Javad Hosseini, Mark Johnson, and Mark Steedman. Sources of hallucination by large language models on inference tasks. *arXiv preprint arXiv:2305.14552*, 2023. 7.1

[106] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019. 7.1

[107] Piotr Mirowski, Matthew Koichi Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Koray Kavukcuoglu, Andrew Zis-

serman, and Raia Hadsell. Learning to navigate in cities without a map. *arXiv preprint arXiv:1804.00168*, 2018. 8.1

[108] Aditi Mishra, Utkarsh Soni, Jinbin Huang, and Chris Bryan. Why? why not? when? visual explanations of agent behaviour in reinforcement learning. In *2022 IEEE 15th Pacific Visualization Symposium (PacificVis)*, pages 111–120. IEEE, 2022. 7.1

[109] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 5.5.3

[110] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P How. Learning to teach in cooperative multiagent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6128–6136, 2019. 1.1, 1.2, 2.3.2

[111] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P How. Learning to teach in cooperative multiagent reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6128–6136, 2019. 1.2, 5.1

[112] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014. 9.1

[113] Błażej Osiński, Adam Jakubowski, Paweł Ziecina, Piotr Miłoś, Christopher Galias, Silviu Homoceanu, and Henryk Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 6411–6418. IEEE, 2020. 1.1

[114] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009. 8.1

[115] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010. 9.1

[116] Grusha Prasad, Yixin Nie, Mohit Bansal, Robin Jia, Douwe Kiela, and Adina Williams. To what extent do human explanations of model behavior align with actual model behavior? In *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 1–14, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.blackboxnlp-1.1. 7.1

[117] Erika Puiutta and Eric MSP Veith. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, pages 77–95. Springer, 2020. 7.1

[118] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4932–4942,

Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1487. 7.1

[119] Haidi Rao, Xianzhang Shi, Ahoussou Kouassi Rodrigue, Juanjuan Feng, Yingchun Xia, Mohamed Elhoseny, Xiaohui Yuan, and Lichuan Gu. Feature selection based on artificial bee colony and gradient boosting decision tree. *Applied Soft Computing*, 74:634–642, 2019. 3.1

[120] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018. 1.3

[121] Emilio Remolina, Juan A Fernandez, Benjamin Kuipers, and Javier Gonzalez. Formalizing regions in the spatial semantic hierarchy: An ah-graphs implementation approach. In *International Conference on Spatial Information Theory*, pages 109–124. Springer, 1999. 8.1

[122] Danilo Rezende, Ivo Danihelka, Karol Gregor, Daan Wierstra, et al. One-shot generalization in deep generative models. In *International conference on machine learning*, pages 1521–1529. PMLR, 2016. 9.1

[123] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016. 7.1

[124] Coppelia Robotics. Coppeliasim. http://www.coppeliarobotics.com, 2021. Version 4.2.0. 5.5.3

[125] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 1.2, 2.1, 5.1

[126] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 4.2, 4.2.1

[127] Christian Scheller, Yanick Schraner, and Manfred Vogel. Sample efficient reinforcement learning through learning from demonstrations in minecraft. In *NeurIPS 2019 Competition and Demonstration Track*, pages 67–76. PMLR, 2020. 1.1

[128] Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018. 4.2.1, 5.3.2

[129] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 5.5.3

[130] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2.3.1, 4.1.3, 4.2.1

[131] Farzaneh Shoeleh and Masoud Asadpour. Skill based transfer learning with domain adaptation for continuous reinforcement learning domains. *Applied Intelligence*, 50(2): 502–518, 2020. 8.1

[132] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017. 7.1

[133] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 1.1

[134] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 1.1

[135] Simon Stepputtis, Maryam Bandari, Stefan Schaal, and Heni Ben Amor. A system for imitation learning of contact-rich bimanual manipulation policies. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, October 2022. doi: 10.1109/iros47612.2022.9981802. URL https://doi.org/10.1109/iros47612.2022.9981802. 7.1

[136] Sriram Ganapathi Subramanian, Matthew E Taylor, Kate Larson, and Mark Crowley. Multi-agent advisor Q-learning. *Journal of Artificial Intelligence Research*, 74:1–74, 2022. 1.2

[137] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 7.1

[138] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017. 1.3

[139] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 1.3, 2.3.1, 4.1

[140] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018. 8.1, 9.1

[141] Adriana Tapus, Shrihari Vasudevan, and Roland Siegwart. Towards a multilevel cognitive probabilistic representation of space. In *Human Vision and Electronic Imaging X*, volume 5666, pages 39–48. International Society for Optics and Photonics, 2005. 8.1

[142] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009. 8.1

[143] Andrea L Thomaz and Cynthia Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 172(6-7):716–737, 2008. 1.2, 5.1

[144] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060, 2013. 1.1, 1.2, 2.2.1, 2.3.2, 2.3.3, 5.1

[145] Lisa Torrey, Trevor Walker, Jude Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*, pages 412–424. Springer, 2005. 1.2, 5.1

[146] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. Skill acquisition via transfer learning and advice taking. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pages 425–436. Springer, 2006. 1.2, 5.1

[147] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE international conference on computer vision*, pages 4068–4076, 2015. 9.1

[148] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017. 9.1

[149] Noel D Uri. Forecasting peak system load using a combined time series and econometric model. *Applied Energy*, 4(3):219–227, 1978. 8.3.2

[150] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 8.2.4

[151] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018. 7.1

[152] Horatiu Voicu. Hierarchical cognitive maps. *Neural Networks*, 16(5-6):569–576, 2003. 8.1

[153] Samir Wadhwania, Dong-Ki Kim, Shayegan Omidshafiei, and Jonathan P How. Policy distillation and value matching in multiagent reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8193–8200. IEEE, 2019. 1.2

[154] Xinzhi Wang, Shengcheng Yuan, Hui Zhang, Michael Lewis, and Katia Sycara. Verbal explanations for deep reinforcement learning neural networks with attention on extracted features. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–7. IEEE, 2019. 7.1

[155] William H Warren. Non-euclidean navigation. *Journal of Experimental Biology*, 222(Suppl

1), 2019. 8.1

[156] Lindsay Wells and Tomasz Bednarz. Explainable AI and reinforcement learning—a systematic review of current approaches and trends. 4. ISSN 2624-8212. 7.1

[157] Sarah Wiegreffe, Jack Hessel, Swabha Swayamdipta, Mark Riedl, and Yejin Choi. Reframing human-ai collaboration for generating free-text explanations. *arXiv preprint arXiv:2112.08674*, 2021. 7.1

[158] Jingda Wu, Zhiyu Huang, Zhongxu Hu, and Chen Lv. Toward human-in-the-loop ai: Enhancing deep reinforcement learning via real-time human guidance for autonomous driving. *Engineering*, 21:75–91, 2023. 1.1

[159] Yonghui Xu, Sinno Jialin Pan, Hui Xiong, Qingyao Wu, Ronghua Luo, Huaqing Min, and Hengjie Song. A unified framework for metric transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 29(6):1158–1171, 2017. 9.1

[160] Tianpei Yang, Weixun Wang, Hongyao Tang, Jianye Hao, Zhaopeng Meng, Hangyu Mao, Dong Li, Wulong Liu, Yingfeng Chen, Yujing Hu, et al. An efficient transfer learning framework for multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:17037–17048, 2021. 1.2

[161] Yi Yao and Gianfranco Doretto. Boosting for transfer learning with multiple sources. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 1855–1862. IEEE, 2010. 9.1

[162] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681, 2021. 1.1

[163] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014. 9.1

[164] Renos Zabounidis, Joseph Campbell, Simon Stepputtis, Dana Hughes, and Katia P Sycara. Concept learning for interpretable multi-agent reinforcement learning. In *Conference on Robot Learning*, pages 1828–1837. PMLR, 2023. 7.1

[165] Yusen Zhan, Haitham Bou Ammar, and Matthew E Taylor. Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer. 1.2, 5.1

[166] Jing Zhang, Wanqing Li, and Philip Ogunbona. Joint geometrical and statistical alignment for visual domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1859–1867, 2017. 9.1

[167] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE, 2017. 8.1

[168] Wen Zhang, Lingfei Deng, Lei Zhang, and Dongrui Wu. A survey on negative transfer.

IEEE/CAA Journal of Automatica Sinica, 10(2):305–329, 2022. 9.1

[169] X. Zhang, Y. Guo, S. Stepputtis, K. Sycara, and J. Campbell. Explaining agent behavior through natural language interactions. In *Proceedings of the Human Multi-Robot Interaction (HmRI) Workshop at the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023. 7

[170] Yudong Zhang, Shuihua Wang, Preetha Phillips, and Genlin Ji. Binary pso with mutation operator for feature selection using decision tree applied to spam detection. *Knowledge-Based Systems*, 64:22–31, 2014. 3.1

[171] Yifan Zhou, Shubham Sonawani, Mariano Phielipp, Heni Ben Amor, and Simon Stepputtis. Learning modular language-conditioned robot policies through attention. *Autonomous Robots*, August 2023. doi: 10.1007/s10514-023-10129-1. URL https://doi.org/10.1007/s10514-023-10129-1. 7.1

[172] Changxi Zhu, Yi Cai, Ho-fung Leung, and Shuyue Hu. Learning by reusing previous advice in teacher-student paradigm. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1674–1682, 2020. 1.1, 1.2, 2.3.2, 5.1

[173] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 30, 2016. 9.1

[174] Qi Zhu, Yidan Xu, Haonan Wang, Chao Zhang, Jiawei Han, and Carl Yang. Transfer learning of graph neural networks with ego-graph information maximization. *arXiv preprint arXiv:2009.05204*, 2020. 8.1

[175] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020. 8.1

[176] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020. 8.1

[177] Matthieu Zimmer, Paolo Viappiani, and Paul Weng. Teacher-student framework: a reinforcement learning approach. In *AAMAS Workshop Autonomous Robots and Multirobot Systems*, 2014. 1.2, 2.3.2, 5.1