May 2022

Testing Reinforcement Learning Explainability Methods in a Multi-agent Cooperative Environment

Marc DOMÈNECH I VILA [a] and Dmitry GNATYSHAK [b] and Adrián TORMOS [b] and Sergio ALVAREZ-NAPAGAO [b]

[a] *Universitat Politècnica de Catalunya, Barcelona, Spain*
[b] *Barcelona Supercomputing Center, Barcelona, Spain*

Abstract. The adoption of algorithms based on Artificial Intelligence (AI) has been rapidly increasing during the last years. However, some aspects of AI techniques are under heavy scrutiny. For instance, in many cases, it is not clear whether the decisions of an algorithm are well-informed and reliable. Having an answer to these concerns is crucial in many domains, such as those in were humans and intelligent agents must cooperate in a shared environment. In this paper, we introduce an application of an explainability method based on the creation of a Policy Graph (PG) based on discrete predicates that represent and explain a trained agent's behaviour in a multi-agent cooperative environment. We also present a method to measure the similarity between the explanations obtained and the agent's behaviour, by building an agent with a policy based on the PG and comparing the behaviour of the two agents.

Keywords. Explainable AI, Reinforcement Learning, Policy Graphs, Multi-agent Reinforcement Learning, Cooperative Environments

1. Introduction and Motivation

Over the last decade, methods based on machine learning have achieved remarkable performance in many seemingly complex tasks such as image processing and generation, speech recognition or natural language processing. It is reasonable to assume that the range of potential applications will keep growing bigger in future years. However, there are still many concerns about the transparency, understandability and trustworthiness of systems built using these methods, esp. when they are based on so-called *blackbox* models. For example, there is still a need for proper explanations of the behaviour of autonomous vehicles and this could be a risk for their real-world applicability and regulation [1].

Since AI has an increasing impact on people's everyday lives, it becomes urgent to keep progressing on the field of Explainable Artificial Intelligence (XAI) [2]. In fact, there are already regulations that require AI model creators to enable mechanisms that can produce explanations for them, such as the European Union's General Data Protection Regulation (GDPR) that went into effect on

May 25, 2018 [3]. This law creates a "Right to Explanation" whereby a user can ask for the explanation of an algorithmic decision that was made about them. Therefore, XAI is not only desirable but also a frequent requirement. This is also the case for virtual or physical agents trained via Reinforcement Learning (RL), and explainability in RL (XRL) is starting to gain momentum as a different field of XAI.

This paper aims to continue the line of research opened in [4; 5], which consists in producing explanations from predicate-based Policy Graphs (PG) generated from the observation of RL-trained agents in the Cartpole environment. In this paper, we present an application of the same methodology in order to generate explanations for agents trained in a cooperative environment using Multi-Agent Reinforcement Learning (MARL) methods. In the physical world, cooperation between humans and AIs will gradually become more common [6], and thus we believe that it is crucial to be able to explain the behavior of cooperative agents so that their actions are understandable and can be trusted by humans.

Currently, there are several approaches to explain RL agents. In this work, we briefly overview some of them in **Section 2**, we choose one that builds a graph that represents the agent's behaviour and we apply it to a MARL environment in **Section 4**, also giving insight in how to generate explanations. Once we apply the method, we build a new agent using the graph as a policy in order to compare both agents in **Section 5** and finally, we end with a summary of the main conclusions and contributions from the work done in **Section 6**.

2. Related work

The area of explainability in reinforcement learning is still relatively new, especially when dealing with policies as blackbox models. In this section, we will provide a brief overview of some state-of-the-art XRL methods as well as discuss in more depth the method chosen in this work. A more detailed study of the explainability methods in RL can be found in [7].

According to [7], XRL methods can be classified by their scope of explanation (global/local), timing of explanation (post-hoc/intrinsic), time horizon of explanation (reactive/proactive), type of the environment (deterministic/stochastic), type of policy (deterministic/stochastic) and their agent cardinality (single-agent/multi-agent).

Reactive explanations are those that are focused on the immediate moment. A family of reactive methods is policy simplification, which finds solutions based on tree structures. In these, the agent answers the questions from the root to the bottom of the tree in order to decide which action to take. For instance, Coppens et al. [8] use Soft Decision Trees (SFT), structures that work similarly to binary trees but where each decision node works as a single perceptron that returns, for a given input x, the probability of going right or left. This allows the model to learn a hierarchy of filters in its decision nodes. Another family is reward decomposition, which tries to decompose the reward into meaningful components. In [9], Juozapaitis et al. decompose the Q-function into reward types to try to explain why an action is preferred over another. With this, they can know whether

the agent is taking an action to be closer to the objective or to avoid penalties. Another approach is feature contribution and visual methods, like LIME [10] or SHAP [11], which try to find which of the model features are the most relevant in order to make decisions. On the other hand, Greydanus et al. differentiate between gradient-based and perturbation-based saliency methods [12]. The first ones try to answer the question "Which features are the most relevant to decide the output?" while the latter are based on the idea of perturbing the input of the model in order to analyse how its predictions changes.

Proactive models are those that focus on longer-term consequences. One possible approach is to analyse the relationships between variables. This family of techniques give explanations that are very close to humans because we see the world through a causal lens [13]. According to [14], the causal model tries to describe the world using random variables. Each of these variables has a causal influence on the others. This influence is modelled through a set of structural equations. Madumal et al. generate explanations of behaviour based on a counterfactual analysis of the structural causal model that is learned during RL [15]. Another approach tries to break down one task into multiple subtasks in order to represent different abstraction levels [16]. Therefore, each task can only be carried out if its predecessor tasks have been finished.

According to [17], in order to achieve interoperability, it is important that the tasks had been described by humans. For instance, [16] defines two different policies in hierarchical RL, local and global policies. The first one uses atomic actions in order to achieve the sub-objectives while the second one uses the local policies in order to achieve the final goal.

In addition, there is another approach that combines relational learning or inductive logic programming with RL. The idea behind these methods [18] is to represent states, actions and policies using first order (or relational) language. Thanks to this, it is easier to generalize over goals, states and actions, exploiting knowledge learnt during an earlier learning phase. Finally, another approach consists in building a Markov Decision Process and follow the graph from the input state to the main reward state [15]. This allows us to ask simple questions about the chosen actions. As an optional step, we can simplify the state representation (discretizing it if needed). This step becomes crucial when we are talking about more complex environments [4]. The latter approach is the one we will use in our work. It consists of building a policy graph by mapping the original state to a set of predicates (discretization step) and then repeatedly running the agent policy, recording its interactions with the environment. This graph of states and actions can then be used for answering simple questions about the agent's execution which is shown at the end of **Section 4**.

3. Training agents in a cooperative environment: Overcooked-AI

In this paper, we have used the PantheonRL [19] package for training and testing an agent in Overcooked-AI[20]. Overcooked-AI is a benchmark environment for fully cooperative human-AI task performance, based on the popular video game Overcooked. The goal of the game is to deliver soups as fast as possible. Each soup

requires placing up to 3 ingredients in a pot, waiting for the soup to cook, and then having an agent pick up the soup and delivering it. The agents should split up tasks on the fly and coordinate effectively in order to achieve high rewards.

The environment has the following reward function: 3 points if the agent places an onion in a pot or if takes a dish, and 5 points if it takes a soup. Here in this work, we have worked with five different layouts: *simple*, *unident_s*, *random0*, *random1* and *random3* (**Figure 1**).

Figure 1. Overcooked layouts (*simple, unident_s, random1, random0, random3)*

At each timestep, the environment returns a list with the objects not owned by the agent present in the layout and, for each player, the position, orientation, and object that it is holding and its information. We can also get the location of the basic objects (dispensers, etc.) at the start of the game.

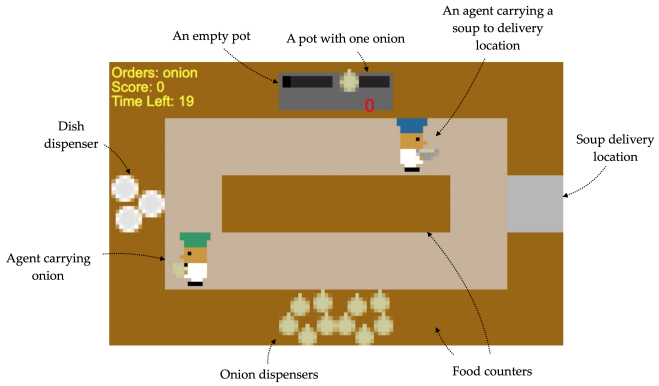

Figure 2. Overcooked-AI game dynamics.

For example, the agent would receive the following data from **Figure 2**:

- **Player 1**: Position (5, 1) - Facing (1, 0) - Holding Soup
- **Player 2**: Position (1, 3) - Facing (-1, 0) - Holding Onion
- **Not owned objects**: Soup at (4, 0) - with 1 onion and 0 cooking time.

The aim of our work is not to solve the Overcooked game but rather to analyze the potential of explainability in this cooperative setting. Therefore, we do not really care about what method is used to train our agent. However, it is important that the agent performs reasonably well in order to verify that we are explaining an agent with a reasonable policy. Therefore, we have used the Proximal Policy Optimization (PPO) algorithm because it is one of the methods that has achieved the best results in [20]. Indeed, if we get good results with PPO, we should also get

good results with other methods since this explainability method is independent from the RL method used. In our case, we have trained five different agents (one for each layout) for 1M total timesteps and with an episode length of 400 steps. The results are the following:

- *simple*: Mean Episode Reward = 387.87 and Standard Deviation = 25.33
- *unident_s*: Mean Episode Reward = 757.71 and Standard Deviation = 53.03
- *random0*: Mean Episode Reward = 395.01 and Standard Deviation = 54.43
- *random1*: Mean Episode Reward = 266.01 and Standard Deviation = 48.11
- *random3*: Mean Episode Reward = 62.5 and Standard Deviation = 5.00

4. Building a policy graph for the trained agent

As we mentioned in **Section 2**, we have chosen a method based on the creation of a policy graph. This method consists of building a directed graph where each node represents a state, and each edge represents the transition going from one node to another taking a specific action.

In order to build the policy graph, we need to discretize the state representation. This step is crucial since we need to map each state to a node in our PG. We have created a total of 10 predicates to represent each state.

The first two are *held* and *held_partner*, which have 5 possible values depending on which object the agent and its partner, respectively, are holding (e.g., "O" for "Onion" object or "*" if is not holding anything). The third is *pot_state*, which has 4 possible values depending on the state of each pot:

- "Of", when [pot.onions = 0].
- "Fi" , when [pot.onions = 3 $\wedge$ pot.timer = 20].
- "Co" , when [pot.onions = 3 $\wedge$ pot.timer < 20].
- "Wa" , when [pot.onions < 3].

To relate the actions of the agent with the relative position of the objects, we introduce another 6 predicates: *onion_pos(X)*, *tomato_pos(X)*, *dish_pos(X)*, *pot_pos(X)*, *service_pos(X)* and *soup_pos(X)*. All of them with the same 6 possible values depending on the next action to perform to reach the object quickly (e.g., "T" for "Top" action). The last one is *partner_zone*, intended to help the agents cooperate along with *held_partner*. It has 8 possible values depending on which cardinal point the partner is located (e.g., "NE" for "North East").

As we mentioned in **Section 2**, the aim of the PG algorithm is to apply the same method as in **Figure 3**: record all the interactions of the original trained agent by executing it in a large set of random environments and build a graph relating predicate-based states and actions. Our work has followed two approaches for building this graph:

- **Partial Policy Graph**: This algorithm builds a directed graph. For each state, it takes the most probable action. Therefore, we do not add all the agent interactions to our graph, only those that belong to the most used action by the agent. This means that for each node, we only have one possible action (the most probable in this case). We think this could be an interesting approach since we are building a deterministic agent.

- **Complete Policy Graph**: This algorithm builds a multi-directed graph. For each state, it takes all the agent interactions, adding them to our graph. This means that for each node, we have multiple possible actions with an associated probability. We think this could be an interesting approach since we are maintaining stochasticity.

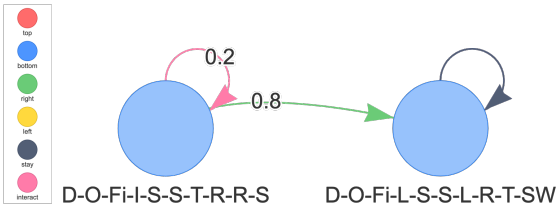

Figure 3. Extract of two states from a Complete PG generated from Overcooked.

Once we have built our PG, we have access to the generation of natural language explanations as in [4], where the authors validate the PG by comparing the sentences generated by the algorithm against sentences written by human experts. While this is valid as a qualitative approach for validation, it becomes obvious that this method heavily depends on experts and on the nature of the specific domain. However, the authors propose three questions to help explain the agent's behaviour. These questions are:

1. **What will you do when you are in state X?**: Look for the possible actions in the policy graph from the input state that the user wants to check.
2. **When do you perform action X?**: Look for these states where the action X is the most probable action.
3. **Why did you not perform action X in state Y?**: Look at which state it would reach if it had chosen action X in state Y. Once we have those nearby (similar) states, then we compute the difference between both states.

5. Validating the policy graph

In this section, we study how to validate our Policy Graph. To do it, we build another agent that follows a policy based on the PG we obtained as explained in **Section 4**: at each step, the agent receives the current state and decides its next action by querying the PG for the most probable action from the most similar state to the current one. In order to test this new agent, we run it multiple times in random environments. If we discretize the space as explained in **Section 4**, we have 10 predicates and 37,324,800 potential states. This means it is very likely that the new agent will find states never seen before, so it is very important to set up a strategy to deal with these situations. We introduce a state similarity metric to deal with unknown states: we consider that two states (S_1, S_2) are similar when $\mathit{diff}(S_1, S_2) \leq 1$, where $\mathit{diff}(S_1, S_2)$ computes the number of different predicate values one each other. For example, if we have the states S_1 = O-Co-S and S_2 = O-Co-N, then $\mathit{diff}(S_1, S_2) = 1$ so they are considered similar.

As we saw in **Section 4**, we are testing 2 different algorithms. Therefore, we have built 2 new agents for each layout. Now, we will see how each of these algorithms make decisions. Assuming we are in state S, we can distinguish 3 cases:

1. $S \in PG$: Picks an action using weights from the probability distribution in the PG.
2. $S \notin PG$, but a similar state is found: Same as case 1 but using the similar state.
3. $S \notin PG$ and a similar state is not found: Pick a random action.

All the agents have been trained using batches of 25 seeds. Altogether, the training consisted in 500 seeds and 3 episodes per seed. In order to validate the new agent, we have generated 3 metrics: Transferred Learning (TL), the ratio between the RL agent average episode reward and the new agent's average episode reward; Standard Deviation (STD), the ratio between the RL agent average standard deviation of reward and the new agent's standard deviation of reward; and New States (NS) visited, the proportion between previously unknown and total visited states. In order to do comparisons, we have tested both PG algorithms using multiple discretizers (D11, D12, D13 and D14). D11 has the predicates *held*, *pot_state* and *predicate_pos*, D12 has D11 predicates plus *held_partner*, D13 has D11 predicates plus *partner_zone* and D14 has all predicates.

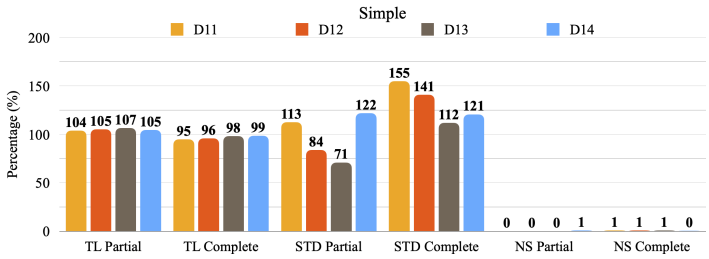

Figure 4. Results from layout *simple*

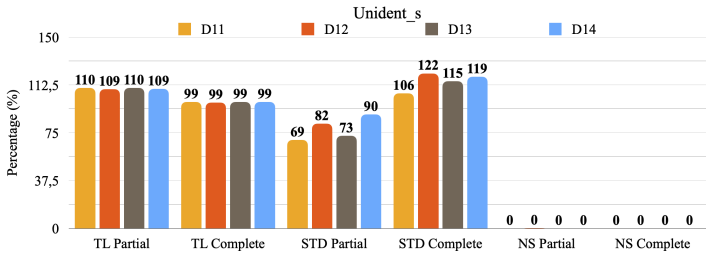

Figure 5. Results from layout *unident_s*

Figures [**4-8**] show the results we have obtained from the different agents. We can see that in the *simple* and *unident_s* scenarios, the Partial agents manage to outperform the original ones while the Complete agents do not. On the other hand, it seems that in these cases it is not necessary to introduce cooperative predicates to explain their behaviour, unlike in the *random0* and *random3* scenarios. For

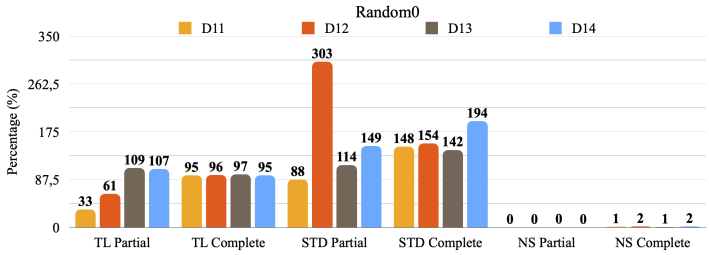

Figure 6. Results from layout *random0*

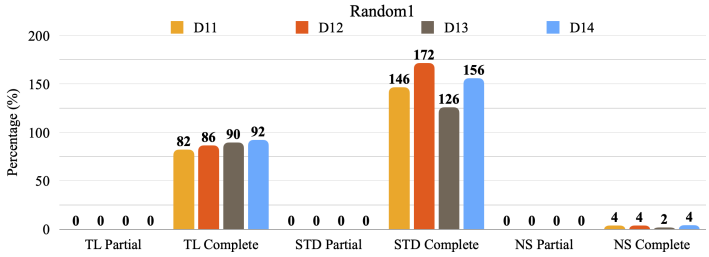

Figure 7. Results from layout *random1*

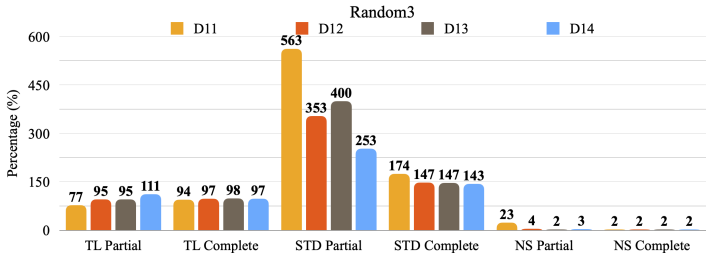

Figure 8. Results from layout *random3*

instance, in *random0* the Partial agent needs the predicate *partner_zone* and in *random3* the predicate *held_partner* to get good results. We can also see how in *random1*, the Partial agent is not even able to score even though the Complete algorithm scores quite well. Regarding the standard deviation of these agents, it is usually higher than the original agent's, likely because the PG we have built is based on the simplification of states and actions, so the policy also ends up being a simplification. Finally, we can see how the vast majority of agents achieve a really low NS percentage, which means that the agent knows exactly what action to take most of the time.

To summarise, according to **Figures [4-8]**, the Complete algorithm is more stable in our 3 metrics than the Partial one, as the latter can score zero points or perform better than the original agent depending on the layout, probably due to its deterministic nature. Therefore, the Complete algorithm is the more reliable from an XRL point of view and should be the one used for producing explanations.

On the other hand, we have also seen that although there are scenarios where it is not necessary to introduce cooperative predicates to explain the agent's

behaviour, there are others where this information is crucial, which makes sense due to the fact that the layout influences the need for cooperation.

6. Conclusions

XAI is a research area that is growing by leaps and bounds in recent years, due to the need to understand and justify the decisions made by AIs, especially in the field of RL. All the research in this area can be key not only to study the quality of an agent's decision but also to help people rely on AI, especially in situations where humans and machines have to cooperate, and it is becoming necessary to be able to give explanations about their decisions. There are already some proposals in the literature to provide them, and it is important to test their effectiveness in practice.

In this work, we have used an explainability method based on the construction of a PG by discretizing the state representation into predicates for later applying it to a cooperative MARL environment (Overcooked). We have proposed two different algorithms to generate the PG and we have managed to give some explanations that according to [4] should be validated by human experts with domain-specific knowledge. In order to validate the PG, we have applied a method already tested in [5] to generate automatically policies based on these explanations to build agents that represent them. Finally, we have seen that this technique is capable of giving explanations in a MARL cooperative environment like Overcooked.

References

[1] Omeiza D, Webb H, Jirotka M, Kunze L. Explanations in Autonomous Driving: A Survey. IEEE Transactions on Intelligent Transportation Systems. 2021:1-21. ArXiv:2103.05154 [cs]. Available from: `http://arxiv.org/abs/2103.05154`.

[2] Longo L, Goebel R, Lecue F, Kieseberg P, Holzinger A. Explainable artificial intelligence: Concepts, applications, research challenges and visions. In: International Cross-Domain Conference for Machine Learning and Knowledge Extraction. Springer; 2020. p. 1-16.

[3] Goodman B, Flaxman S. European Union regulations on algorithmic decision-making and a "right to explanation". AI magazine. 2017;38(3):50-7.

[4] Hayes B, Shah JA. Improving robot controller transparency through autonomous policy explanation. In: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI. IEEE; 2017. p. 303-12.

[5] Climent A, Gnatyshak D, Alvarez-Napagao S. Applying and Verifying an Explainability Method Based on Policy Graphs in the Context of Reinforcement Learning. In: Artificial Intelligence Research and Development. IOS Press; 2021. p. 455-64.

[6] Dafoe A, Hughes E, Bachrach Y, Collins T, McKee KR, Leibo JZ, et al. Open Problems in Cooperative AI. arXiv:201208630 [cs]. 2020 Dec. ArXiv: 2012.08630. Available from: `http://arxiv.org/abs/2012.08630`.

[7] Krajna A, Brcic M, Lipic T, Doncevic J. Explainability in reinforcement learning: perspective and position. arXiv preprint arXiv:220311547. 2022.

[8] Coppens Y, Efthymiadis K, Lenaerts T, Nowé A, Miller T, Weber R, et al. Distilling deep reinforcement learning policies in soft decision trees. In: Proceedings of the IJCAI 2019 workshop on explainable artificial intelligence; 2019. p. 1-6.

[9] Juozapaitis Z, Koul A, Fern A, Erwig M, Doshi-Velez F. Explainable reinforcement learning via reward decomposition. In: IJCAI/ECAI Workshop on explainable artificial intelligence; 2019. .

[10] Ribeiro MT, Singh S, Guestrin C. "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, San Francisco, CA, USA, August 13-17, 2016; 2016. p. 1135-44.

[11] Lundberg SM, Lee SI. A Unified Approach to Interpreting Model Predictions. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, et al., editors. Advances in Neural Information Processing Systems 30. Curran Associates, Inc.; 2017. p. 4765-74. Available from: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[12] Greydanus S, Koul A, Dodge J, Fern A. Visualizing and understanding atari agents. In: International conference on machine learning. PMLR; 2018. p. 1792-801.

[13] Sloman S. Causal models: How people think about the world and its alternatives. Oxford University Press; 2005.

[14] Halpern JY, Pearl J. Causes and Explanations: A Structural-Model Approach — Part 1: Causes. 2013 jan. Available from: https://arxiv.org/abs/1301.2275v1.

[15] Madumal P, Miller T, Sonenberg L, Vetere F. Explainable reinforcement learning through a causal lens. In: Proceedings of the AAAI conference on artificial intelligence. vol. 34; 2020. p. 2493-500. Issue: 03.

[16] Kulkarni TD, Narasimhan K, Saeedi A, Tenenbaum J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. Advances in neural information processing systems. 2016;29.

[17] Shu T, Xiong C, Socher R. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. arXiv preprint arXiv:171207294. 2017.

[18] Zambaldi V, Raposo D, Santoro A, Bapst V, Li Y, Babuschkin I, et al.. Relational Deep Reinforcement Learning. arXiv; 2018. Number: arXiv:1806.01830 arXiv:1806.01830 [cs, stat]. Available from: http://arxiv.org/abs/1806.01830.

[19] Sarkar B, Talati A, Shih A, Sadigh D. PantheonRL: A MARL Library for Dynamic Training Interactions. arXiv; 2021. Number: arXiv:2112.07013 arXiv:2112.07013 [cs]. Available from: http://arxiv.org/abs/2112.07013.

[20] Carroll M, Shah R, Ho MK, Griffiths T, Seshia S, Abbeel P, et al. On the utility of learning about humans for human-ai coordination. Advances in neural information processing systems. 2019;32.