Exploring Agent Behaviors in Network Security through Trajectory Clustering

Ondrej Lukas[1,*,†], Sebastian Garcia[1,†]

[1]Faculty of Electrical Engineering, Czech Technical University in Prague, Czechia

Abstract

Reinforcement learning has been successfully used for training security agents, but there have not been explanations for the behavior of their policies. In this work, we study the behavior of reinforcement learning-based attacking agents in network security environments to understand how to improve them. The sequences of steps (trajectories) generated by the agent-environment interactions are used for (i) analyzing the change in behavior during the training process and (ii) analyzing the performance of the policy of the trained agent. Our proposed method uses a vector representation of the trajectory steps, which are clustered to find similarities in the trajectories based on actions taken, their effects on the state of the environment, and the rewards obtained by the agents. The trajectory cluster analysis is paired with additional visualizations to provide a better and deeper understanding of the policies. Preliminary results show that the proposed method can identify behavioral patterns in the agents' policies and subsequently help guide the agent's learning process.

Keywords

Explainable RL, Trajectory Analysis, Policy Evaluation, Network Security

1. Introduction

Reinforcement Learning (RL) has been successfully used in various complex problems, from theoretical games to robotics. Its application to the security domain has already been adopted by research in simulated security environments [1, 2]. Various model architectures were proposed for the RL-based agent playing the role of the attacker or penetration tester for traditional and Deep RL methods [3, 4, 5].

The evaluation of an agent and its learning progress often relies only on numerical metrics such as win rate or mean return. While informative, especially during the early stages of the agent training, such evaluation does not provide sufficient insight into the trained agent's behavior, development throughout the training process, and ability to generalize [6]. Therefore, a deeper insight into the trajectories generated by the agent policy plays an important role in the hyperparameter selection, training setup, and agent verification [7]. In this ongoing work, we are focusing on exploring two main research questions:

Late-breaking work, Demos and Doctoral Consortium, colocated with The 2nd World Conference on eXplainable Artificial Intelligence: July 17–19, 2024, Valletta, Malta

*Corresponding author.

†These authors contributed equally.

ondrej.lukas@aic.fel.cvut.czc (O. Lukas); sebastian.garcia@agents.fel.cvut.cz (S. Garcia)

https://cs.fel.cvut.cz/en/people/lukasond (O. Lukas); https://cs.fel.cvut.cz/en/people/garciseb (S. Garcia)

0000-0002-7922-8301 (O. Lukas); 0000-0001-6238-9910 (S. Garcia)

1. RQ1: How can a trajectory analysis provide insights for model-agnostic behavior evaluation and understanding?
2. RQ2: How do the trajectories exhibit changes during the training process?

Furthermore, we aim to explore the suitability of the trajectory analysis for finding similarities in the behavior of different model architectures. These can help identify the necessary steps in task solving and can be further used to validate the agents' behavior and explain it to humans.

The main contribution of this work is an evaluation of a variety of RL agents playing in the NetSecGame environment and comparing their behaviors. The comparison and evaluation are done following a method for processing the game-play trajectories of agent's policies.

2. Related Work

In recent years, there have been notable advancements in the RL for security both in the agents [8, 9, 3, 4, 5] and the environments [2, 1, 8]. Still, there is a lack of explainable methods that would allow verification and easier human-computer cooperation in the security domain.

The increased focus on model interpretability can also be seen in the reinforcement learning domain. There are three main approaches to explainable RL: *Model explanation* only focuses on the underlying model, *Policy Explanation* explains the behavior of the agent, and *Outcome explanation* focuses on the local explanation of a (sub)trajectory [10].

In the latter two, the trajectories are commonly used for decision attribution [11], visual explanations [12], directly for model improvement [13] or summarization of agent's behavior [14, 15] . However, none of these methods was evaluated in a security scenario.

3. Methodology

The NetSecGame[1] is an environment simulation of high-level network security tasks. The agent, playing the role of the attacker, interacts with the environment in an episodic setup, learning the dynamics of the environment in the process. The scenario used in this work simulates the Sensitive data exfiltration attack, in which the task of the attacker is to (i) understand the topology of the local networks, (ii) locate the sensitive data, (iii) take certain measures to access the data, and finally (iv) exfiltrate the data to an external location outside the local network.

In the NetSecGame, state representation and actions do not have a fixed size, which is different from most gym-like environments. States are represented by a collection of assets available to the agent consisting of a *set of known networks*, a *set of known hosts*, a *set of controlled hosts*, a *set of known services*, and a *set of known data*.

The actions in this environment consist of five *action types*, each of them having a different set of parameters that are selected from the state representation (e.g., IP addresses and services). All actions have a parameter that identifies from which host (position in the environment) it is executed. For example, given a state s in which the agent controls a single host **A** in a network **N**, the action of *ScanNetwork* can be played with parameters *source host=A*, *target network=N*.

[1] https://github.com/stratosphereips/NetSecGame

Such parametrization of actions allows for a modular and flexible environment that can model various scenarios and situations. Also, it makes the trained policies difficult to visualize, analyze, and evaluate. The environment changes after every agent's move, and a new state and an immediate reward is given to the agent. Each of these **steps** is represented by a tuple $(s_t, a_t, r_{t+1}, s_{t+1})$, where s_t is the current state of the game, a_t is the action performed in the state s_t, r_{t+1} is the immediate reward for playing action a_t, and s_{t+1} is the following state as the result of action a_t in state s_t.

A **trajectory** t is a sequence of steps starting from the initial state s_0 until the terminal state of the episode, which ends when the goal is reached, agent detected, or by timeout (reaching the maximum allowed episode length). To analyze the agent's behavior during and after the training, we capture the trajectories generated by each agent.

We evaluate three types of agents (models) in the current stage of this work: Two variants of Q-learning and an LLM-based agent. The first model is a vanilla Q-learning algorithm with decaying ϵ exploration. In contrast, the second model still uses Q-learning but is extended with concepts to generalize to networks without knowing details such as the IP addresses, helping merge some of the state-action pairs. This results in better generalization to unknown networks and less overfitting to the topology of the network used in the trained task. The last model evaluated is based on the OpenAI LLM GPT-3.5-turbo. The LLM-agent [8] uses the textual representation of the state, description of the goal, and the environment to select an action to be played in the state. The LLM model is **not** fine-tuned for playing the role of an attacker, apart from the prompt composition.

Trajectories from 500 evaluation episodes were collected for each model at multiple training checkpoints for comparison and analysis. In the case of the LLM agent, there was no training period; thus, only the evaluation trajectories were used. The first part of the policy evaluation focuses only on the sequence of actions. We study the action type distribution per step based on all the trajectories gathered for a policy. The distribution of the action types shows the agent's primary goal in each step of the interaction.

The optimal trajectory in the data exfiltration scenario consists of 5 steps, which allows computing the mean action type efficiency given the set of trajectories T as follows:

Let $T_{wins} = \{t \in T \mid t.\text{end} = \text{win}\}$ be a subset of trajectories in which the agent wins. Then, we compute the efficiency of the action type a^t as

$$efficiency_{a^t}(T_{wins}) = \frac{|T_{wins}|}{|\{s \in T_{wins} \mid s.\text{action} = a^t\}|}$$

This metric equals 1 for each action type for an optimal trajectory, as each should be played only once. Values less than 1 mean that the action of type t is repeatedly played - most likely with incorrect parameters.

While analyzing the action sequence brings insights into the agent's behavior, it does not fully use the information the trajectories provide. Therefore, we propose encoding each step s of a trajectory t using the following vector representation for further processing and analysis:

1. Size of each component of s.
2. Size of each component of s_{next}.
3. Amount of change caused by a ($|s_{next} - s|$).

4. Reward r.
5. Return when starting from the step s. (Sum of all rewards that the agent expects to receive from state s until the end of the episode)
6. One-hot encoded action a used in step s.

After the encoding, the trajectory steps are processed by UMAP [16] (Uniform Manifold Approximation and Projection). UMAP is a dimensionality reduction technique that efficiently maps high-dimensional data into a lower-dimensional space. It uses manifold learning techniques to model the underlying structure of the data, preserving both local and global structures. We propose using the projection to find similarities among the trajectory steps of different models.

4. Results

The results of comparing action types can be seen in Figure 1. It shows the distribution of actions in each step of the trajectory for Q-learning (Figure 1a), Q-Learning with general concepts(Figure 1b) and GPT-3.5 agent(Figure 1c). The bar plot in Figure 1d compares the Action efficiency of each model.

UMAP projection of the trajectory steps is shown in Figures 2 and 3. The step number, underlying model, action type, and the outcome of the trajectories are highlighted.

5. Discussion

The comparison in Figure 1 shows several differences in the policies, most notably in the lengths of trajectories and the action composition.

All three models show in the first steps an initial phase of *exploration* (mainly composed of *Scan Network* action and *Scan Services* action). Still, in the case of the Conceptual Q-learning, it is heavily focused on the *FindData* action. Since searching the data in a host requires control of the host, taking this action in the first step of the game is impractical as confirmed by the action efficiency of 15% shown in Figure 1d. The second major difference in the behavior of the Conceptual agent is the significant use of *Data Exfiltration* action in the later stages of the interaction. In comparison, the LLM and Q-learning agents are exfiltrating the data in very few cases, suggesting that it only happens for the correct data point required to win the game.

The high action efficiency of the Q-learning agent may indicate that the model could be overfitted to the particular task and network topology. This hypothesis is further supported by a low amount of *Find Data* actions, likely caused by a lack of exploration. In contrast, the LLM agent (which has no additional training for this particular task) shows more exploration (usage of *Scan Network*, *Find Services*, and *Find Data*). Such behavior, which shows less efficiency in this particular task and topology, can lead to better generalization capabilities of the policy.

The UMAP projection in Figure 2 supports the hypothesis of unnecessary use of *Exfiltrate Data* action of the conceptual agent as those steps should be taken later in the interaction and lead to either a *timeout* ending or a *detection* ending, which is visible in the largest central cluster.

Model attribution in the second subplot of Figure 2 indicates higher similarity in the Q-learning and LLM trajectory steps despite significant model differences.

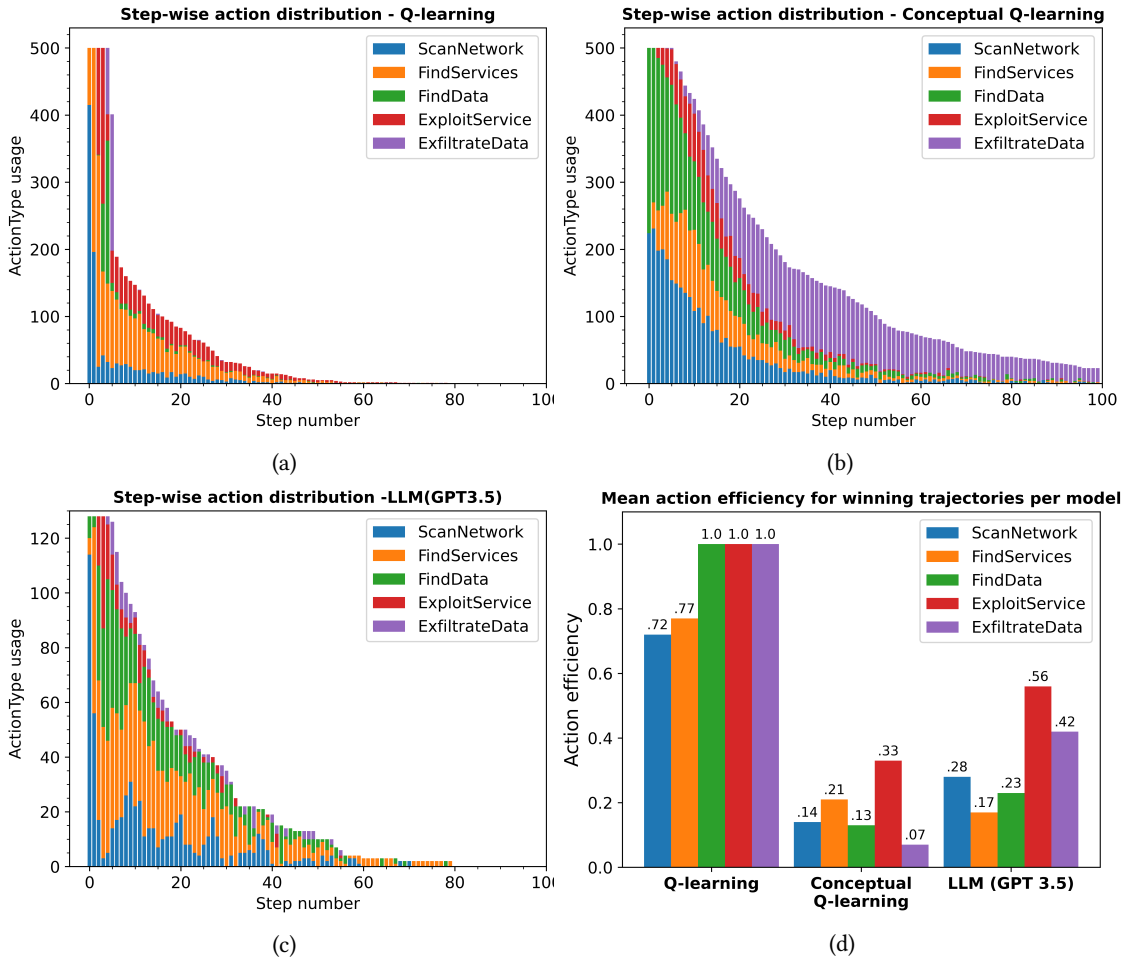

Figure 1: Figures (a), (b), and (c) show action type distribution per step for Q-learning (a), Q-learning with concepts (b), and LLM-based model (c) in all evaluation episodes. The height of the bar represents the number of evaluation episodes in which the corresponding step was reached. The decreasing height of the bars shows a lower occurrence of long episodes. Figure (d) plot shows action efficiency per model for **winning** episodes only (higher is better).

Figure 3 shows the comparison of the trajectories of the Q-learning model at five distinct points of training. Since the figure depicts only one model type, it shows a lower separation of the clusters. However, the *Action type* subplot shows smaller clusters around them, which show higher purity and correspond to the winning trajectories. These clusters are attributed to the policies in the later stage of the training, having steps that occurred in the first twenty steps of the trajectories. A possible explanation is that as the model adapts to the environment, the produced trajectories have less exploration and higher similarity. In the projection, this results in the smaller peripheral clusters.

A notable exception are the two clusters of steps with action *ScanNetwork* and *FindServices* in the lower left part of the plot. The end reason subplot shows that they consist of both winning

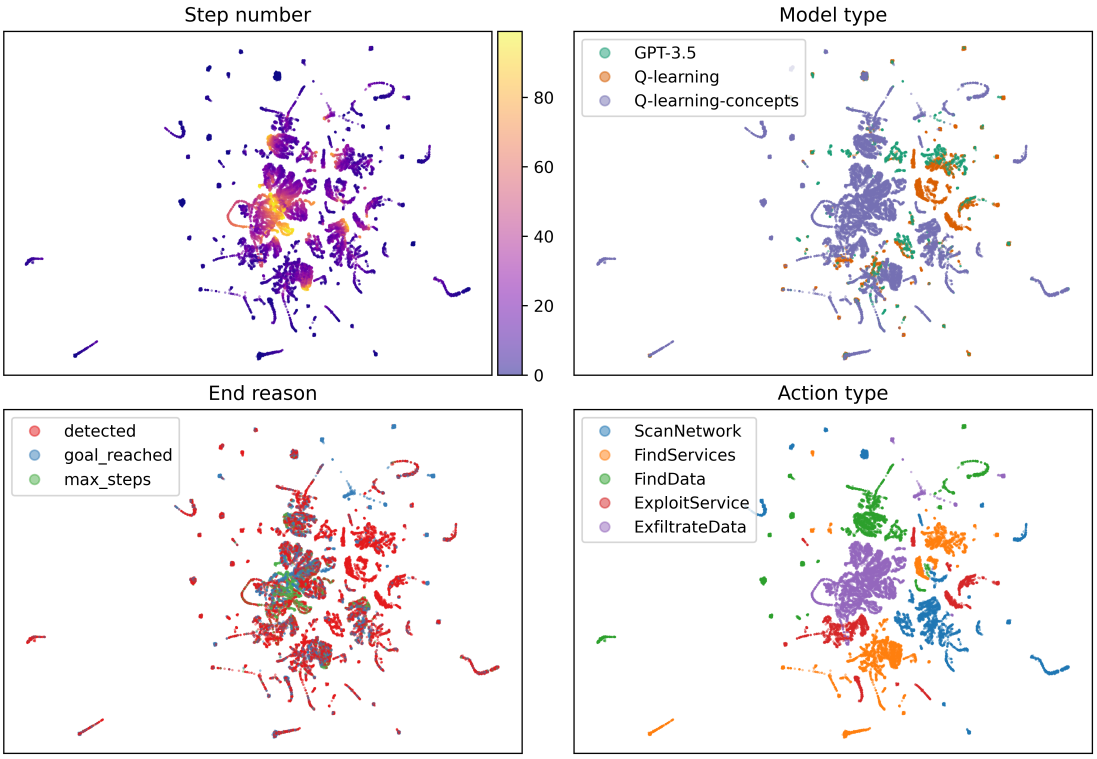

Figure 2: UMAP projection of the vector representation of trajectory steps. *Step number* is the sequence number of the step in each trajectory. The knowledge of the *End reason* of the trajectory is assigned to all the steps in that trajectory.

and losing trajectory steps. We can see that those steps occur at the beginning of the trajectories and for most of the models. The most likely reason is that these clusters consist of the agents' initial recon steps. Since the starting state, while being randomized, is very similar in each of the trajectories, this part of the Q-table is learned very early in the training process.

6. Conclusion and Future Work

In this early-stage work, we introduce the policy evaluation for a security scenario using the trajectory step analysis. We propose the vector representation of the trajectories generated by the RL agent and demonstrate its application in visual explanations of trained policies. We show that the trajectory steps and proposed vector representation can be used to find similarities in the policies of different model types. We evaluate the method to explain the policies during the training process.

Currently, no DRL models are included in the evaluation. Additionally, comparison with other security environment is needed.

In the project's current phase, the analysis focuses only on the steps of the trajectories, but such an approach might not capture all the complexities of the policy. Future steps should

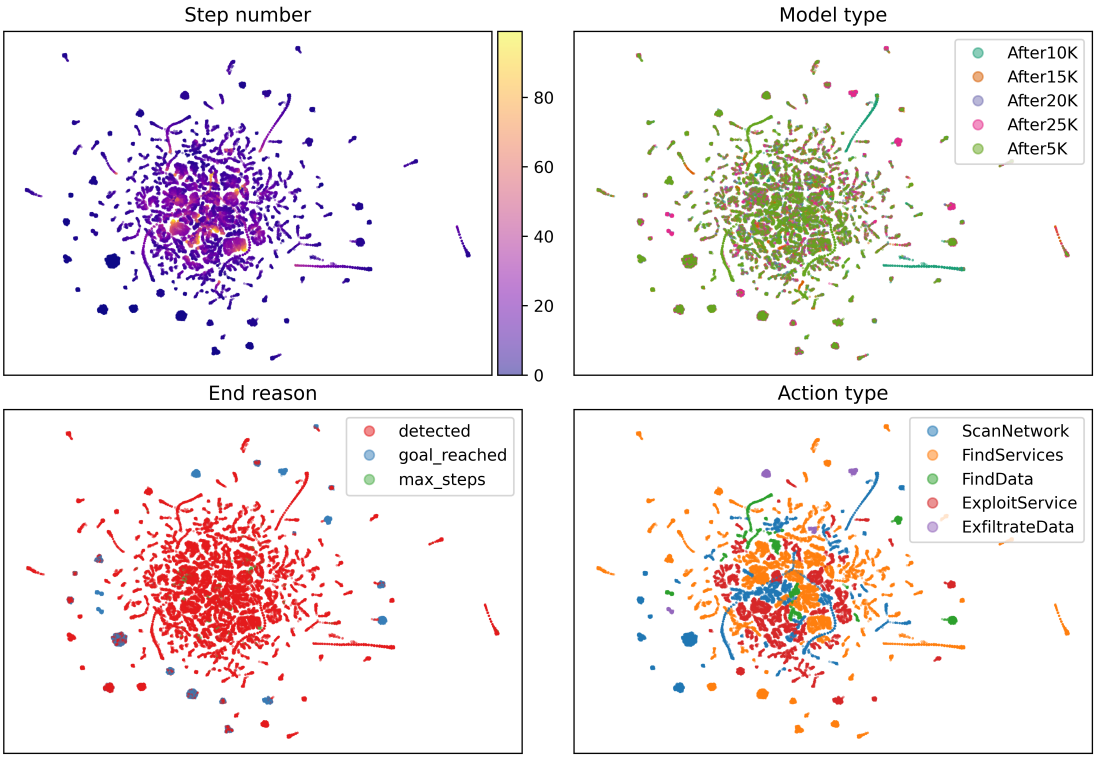

Figure 3: UMAP projection of trajectories obtained from the same Q-learning agent after 5000, 10 000, 15 000, 20 000, and 25 000 training episodes.

focus on extending this work to the sequence of steps and potentially whole trajectories. The primary motivation for such an extension is to better understand and interpret the changes in the agent's behavior during training. Secondly, the better clustering of trajectory steps can allow the detection of agents' intrinsic sub-goals in the trajectories, their comparison across the model types, and their mapping to the existing knowledge base of attacking techniques.

References

[1] M. D. R. Team., Cyberbattlesim, https://github.com/microsoft/cyberbattlesim, 2021. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.

[2] J. Janisch, T. Pevnỳ, V. Lisỳ, Nasimemu: Network attack simulator & emulator for training agents generalizing to novel scenarios, in: European Symposium on Research in Computer Security, Springer, 2023, pp. 589–608.

[3] S. Chaudhary, A. O'Brien, S. Xu, Automated post-breach penetration testing through reinforcement learning, in: 2020 IEEE Conference on Communications and Network Security (CNS), 2020, pp. 1–2. doi:10.1109/CNS48642.2020.9162301.

[4] K. Tran, A. Akella, M. Standen, J. Kim, D. Bowman, T. Richer, C.-T. Lin, Deep hierarchical reinforcement agents for automated penetration testing, 2021. arXiv:2109.06449.

[5] Z. Hu, R. Beuran, Y. Tan, Automated penetration testing using deep reinforcement learning, in: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2020, pp. 2–10. doi:10.1109/EuroSPW51379.2020.00010.

[6] K. Cobbe, O. Klimov, C. Hesse, T. Kim, J. Schulman, Quantifying generalization in reinforcement learning, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, PMLR, 2019, pp. 1282–1289. URL: https://proceedings.mlr.press/v97/cobbe19a.html.

[7] S. Milani, N. Topin, M. Veloso, F. Fang, Explainable reinforcement learning: A survey and comparative review, ACM Comput. Surv. 56 (2024). URL: https://doi.org/10.1145/3616864. doi:10.1145/3616864.

[8] M. Rigaki., O. Lukáš., C. Catania., S. Garcia., Out of the cage: How stochastic parrots win in cyber security environments, in: Proceedings of the 16th International Conference on Agents and Artificial Intelligence - Volume 3: ICAART, INSTICC, SciTePress, 2024, pp. 774–781. doi:10.5220/0012391800003636.

[9] T. T. Nguyen, V. J. Reddi, Deep reinforcement learning for cyber security, IEEE Transactions on Neural Networks and Learning Systems 34 (2023) 3779–3795. doi:10.1109/TNNLS.2021.3121870.

[10] G. A. Vouros, Explainable deep reinforcement learning: State of the art and challenges, ACM Comput. Surv. 55 (2022). URL: https://doi.org/10.1145/3527448. doi:10.1145/3527448.

[11] S. V. Deshmukh, A. Dasgupta, B. Krishnamurthy, N. Jiang, C. Agarwal, G. Theocharous, J. Subramanian, Explaining RL Decisions with Trajectories, 2024. URL: http://arxiv.org/abs/2305.04073. doi:10.48550/arXiv.2305.04073, arXiv:2305.04073 [cs].

[12] Y. Takagi, R. Tabalba, N. Kirshenbaum, J. Leigh, Abstracted trajectory visualization for explainability in reinforcement learning, 2024. arXiv:2402.07928.

[13] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, S. Levine, Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 1009–1018. URL: https://proceedings.mlr.press/v80/co-reyes18a.html.

[14] D. Amir, O. Amir, Highlights: Summarizing agent behavior to people, in: Proceedings of the 17th international conference on autonomous agents and multiagent systems, 2018, pp. 1168–1176.

[15] N. Topin, M. Veloso, Generation of policy-level explanations for reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 2019, pp. 2514–2521.

[16] L. McInnes, J. Healy, J. Melville, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, ArXiv e-prints (2018). arXiv:1802.03426.