Utilizing Explainability Techniques for Reinforcement Learning Model Assurance

Alexander Tapley[*] **Kyle Gatesman** **Luis Robaina** **Brett Bissey** **Joseph Weissman**
The MITRE Corporation - *AI Security and Perception*
`{atapley, kjgatesman, lrobaina, bbissey, weissmanj}@mitre.org`

Abstract

Explainable Reinforcement Learning (XRL) can provide transparency into the decision-making process of a Deep Reinforcement Learning (DRL) model and increase user trust and adoption in real-world use cases. By utilizing XRL techniques, researchers can identify potential vulnerabilities within a trained DRL model prior to deployment, therefore limiting the potential for mission failure or mistakes by the system. This paper introduces the ARLIN (Assured RL Model Interrogation) Toolkit, an open-source Python library that identifies potential vulnerabilities and critical points within trained DRL models through detailed, human-interpretable explainability outputs. To illustrate ARLIN's effectiveness, we provide explainability visualizations and vulnerability analysis for a publicly available DRL model. The open-source code repository is available for download at `https://github.com/mitre/arlin`.

1 Introduction

Over the last decade, reinforcement learning has increased in popularity due to its ability to achieve superhuman performance on a variety of classic board [1] and video game [2] environments. This gain in popularity has sparked an interest in using DRL for both decision support and autonomous operation within safety-critical scenarios such as air-to-air combat [3], nuclear power plant optimization [4], and ballistic missile guidance [5]. These use-cases are considered high-risk as even small mistakes can result in large losses of monetary value, equipment, and life. Before DRL models can safely be deployed within real-world safety critical environments, their associated vulnerabilities need to be identified and understood so effective training enhancements and verification guardrails can be implemented.

In this paper, we present the ARLIN Toolkit, an open-source research library written in Python that provides explainability outputs and vulnerability detection for DRL models, specifically designed to increase model assurance and identify potential points of failure within a trained model. To our knowledge, ARLIN is the first open-sourced Python toolkit focused on utilizing explainability techniques to assure RL models prior to deployment. ARLIN utilizes *matplotlib* [6] and *networkx* [7] to visualize a trained DRL model's decision making process and provide meaningful vulnerability identification and analysis to researchers. The modular library is structured to support custom architectures, algorithms, DRL frameworks, and analytics; and provides a well-documented and tested API for XRL research development and model assurance. The ARLIN repository is available for download at `https://github.com/mitre/arlin`.

[*] Corresponding author.

XAI in Action: Past, Present, and Future Applications @ NeurIPS 2023.

2 Background and Preliminaries

2.1 Reinforcement Learning

Reinforcement learning is an area of machine learning that focuses on teaching an intelligent agent how to interact within an environment in order to optimize a reward function and achieve a specified goal [8]. As the agent interacts with the environment, it receives scaled rewards to indicate good and bad actions. Through trial and error, the agent is able to identify the optimal policy in order to maximize the cumulative reward received and solve the given task.

In DRL, the environment is defined as a **Markov Decision Process**, MDP, $M = (S, A, P, \rho_0, R, \gamma, T)$, where S is the state space, A is the action space, $P : S \times A \times S \to [0, 1]$ is the state transition probability, $\rho_0 : S \times A \to [0, 1]$ is the initial state probability, $R : S \times A$ is the reward function, γ is the discount factor, and T is the maximum episode length. The policy $\pi_\theta : S \times A$ assigns a probability value to an action given a state.

During training, the agent observes the current state of the environment $s_t \in S$ and performs an action $a_t \in A$ according to its policy π_θ. The agent then receives a next state $s'_t \in S$ and reward r_t from R within the environment. The agent's goal is to find a policy that optimizes R. Due to the large state space S, neural networks are commonly used as function approximators in DRL tasks. While this helps the agent to generalize to continuous or large state spaces, it reduces transparency into the decision making process of the model.

2.2 Explainable Reinforcement Learning

The "black-box" nature of deep neural networks make verifying and understanding their underlying reasoning very difficult. A lot of work has been done in the field of Explainable AI (XAI) in recent years [9]. However, most of these works focus on supervised learning or unsupervised learning tasks that deal with non-sequential input data which do not directly transfer in the case of DRL due to the sequential nature of the task. The lack of transparency into the decision making process of an DRL model decreases user and public trust and introduces potentially catastrophic unknowns into the model performance, therefore increasing the potential for mission failure.

Explainable RL (XRL) is a field of RL that focuses on increasing DRL model transparency to give users insight into a model's decision making process. The information gained from XRL techniques can help researchers identify *why* agents are making certain decisions and increase user trust in the model. Milani [10] buckets current XRL works into 3 main categories: feature importance, learning process and MDP, and policy-level. These categories look into different aspects of the agent's decision making process including the importance of different features on the policy's chosen action, training examples that affect the policy outputs, and overall policy behavior analysis. This interpretability information can be labeled as *local* or *global*, where local explanations focus on interpreting the predictions of a single action at a point in time and global explanations give a holistic view of the policy's behavior overall [10]. Our work focuses on the *global* interpretability of an DRL model as we aim to analyze the overarching policy to identify potential critical points that may affect a policy's success.

2.3 Related Works

To our knowledge, ARLIN is the first open-sourced Python library focusing on global explainability and vulnerability detection through human-interpretable analysis visualizations. InterestingnessXRL [11] similarly provides explainability outputs for users, but focuses primarily on identifying interesting interactions between the agent and the environment called *highlights* and returns video-samples of the highlights along with analytics about the interaction itself. While vulnerabilities and critical points may be diagnosed as a highlight, this work does not explicitly focus on these areas. While other repositories linked to XRL are publicly available such as [12], these are providing XRL algorithms themselves as opposed to visualizations and analytics for trained DRL models.

3 Key Features

The ARLIN Toolkit provides three main explainability analysis components to users: latent space analysis, datapoint cluster analysis, and semi-aggregated Markov decision process (SAMDP) [13] analysis.

- **Latent space analysis** uses dimensionality reduction techniques to generate embeddings from user-specified datapoint metadata and plot them in 2-D space. Additional policy metadata can be overlaid onto the embeddings to visualize the relationship between the policy embeddings and the policy metadata.
- **Datapoint cluster analysis** uses unsupervised clustering methods to cluster datapoints based on user-defined policy metadata and provide analysis on each state cluster. Average metrics for each cluster can be plotted for comparison to identify potential outliers and gain information about what is happening in a specific area of the environment or point in time, such as failure states and critical points.
- **SAMDP analysis** transforms the identified state clusters into an SAMDP to provide a holistic overview of how the policy moves through the environment over an entire episode. The analysis uses graph theory to identify paths between nodes along with the actions needed to bring the policy from A to B. Paired with the cluster state analysis, users can identify the paths and actions required for a policy to reach an identified failure state or mistake-prone area.

4 Structure and Customizations

The following is a conceptual overview of the ARLIN library structure along with instructions for adding additional custom components. A practical example usage of the library's methods can be found in Appendix A 1.1.

4.1 Conceptual Structure

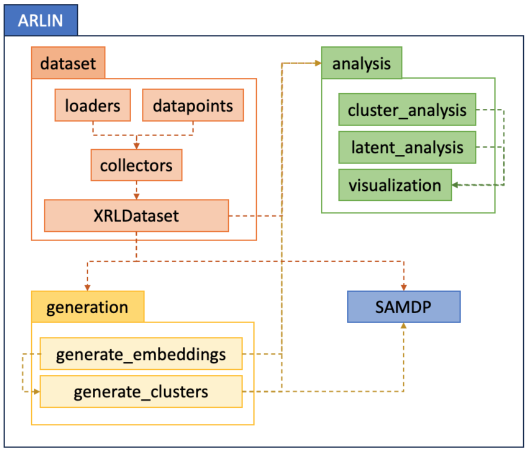

Figure 1: Conceptual structure diagram of the ARLIN library.

4.2 Code Structure

A conceptual diagram of the library structure and relationships between components is shown in Figure 1. At a high-level, ARLIN has 4 main components: `dataset`, `generation`, `analysis`, and `SAMDP`. The `dataset` component is used to create an XRL dataset, a collection of datapoints containing transition data and internal policy metadata collected at every episode step while running a policy within an environment. `generation` uses the XRL dataset to create embeddings and clusters, of which `analysis` provides meaningful analysis and visualizations. The cluster data and XRL dataset can also be provided to SAMDP to generate and visualize different SAMDP graphs of the agent's policy along with available paths between given clusters.

dataset The dataset directory contains all code necessary for creating an XRL dataset from a trained RL model. loaders handle the loading of a trained model while collectors are responsible for collecting the internal data from the RL model. datapoints outline the specific data that the dataset will be storing. The XRLDataset stores all traditional RL transition data (observation, action, reward, done, step) along with the model-specific metadata (Datapoint) gathered by the Collector. Custom loaders, collectors, and datapoints can be added to load custom models and work with custom architectures and algorithms for the collection of user-defined metadata, as outlined in section 4.3.

generation The generation.py file contains the code necessary for datapoint embedding and cluster generation. Metadata from the XRL dataset chosen by the user is reduced to two dimensions via t-SNE [14] to generate latent space embeddings. The datapoints within the XRL dataset are clustered based on user-specified metadata using MeanShift [15] and K-Means [16]. Each cluster represents an area of the policy's latent space where the policy's decision making is affected in similar ways, such as clusters with similar input features or similar output action results.

analysis The analysis directory contains methods for running analysis on both the embeddings as well as the clusters. This includes cluster state representation analysis which analyzes and visualizes the states within the cluster for insight into what states fall into each cluster. The visualization subdirectory contains methods for visualizing the generated analytics using matplotlib [6]. Example latent analysis and cluster analysis visualizations can be found in Appendices A and B, respectively.

samdp The samdp.py file includes the SAMDP class and associated methods. The SAMDP class is a semi-aggregated Markov decision process representation of the policy within its training environment. The SAMDP methods visualize the connections between clusters as well as available paths and actions required to travel to specific target clusters. Available SAMDP methods and visualizations are attached in Appendix C.

4.3 Custom Component Creation

The modular architecture of ARLIN provides support for user customization with no changes to the main library code. Requirements for creating custom components for common aspects of the library are detailed below:

Loaders: The addition of new loaders does not require any inheritance and can be created as a separate method specific to the model that is being loaded. A custom loader must return a trained model with which a user can run inference within the training environment.

Datapoints: To create a new datapoint, the user must inherit from BaseDatapoint and add any additional metadata that the XRLDataset will be storing for the user-specific use case. A datapoint holds information gathered at a single episode step and can store model-specific internal metadata gathered during the model's decision making process.

Collectors: Custom collectors must inherit from BaseDataCollector and implement the required methods. The collector is responsible for collecting the data needed to fill the datapoint, and therefore is specific to the model architecture as the collector needs to understand where to find the necessary metadata to store.

Latent and Cluster Analytics: To create new analysis visualizations, users can simply create a custom method that produces the wanted metric and return a GraphData object for input into the provided visualization methods.

5 Usage

ARLIN is designed to provide users with explainability outputs that can be analyzed to identify potential vulnerabilities and critical points within a trained policy. An example workflow for using ARLIN is shown in Figure 2.

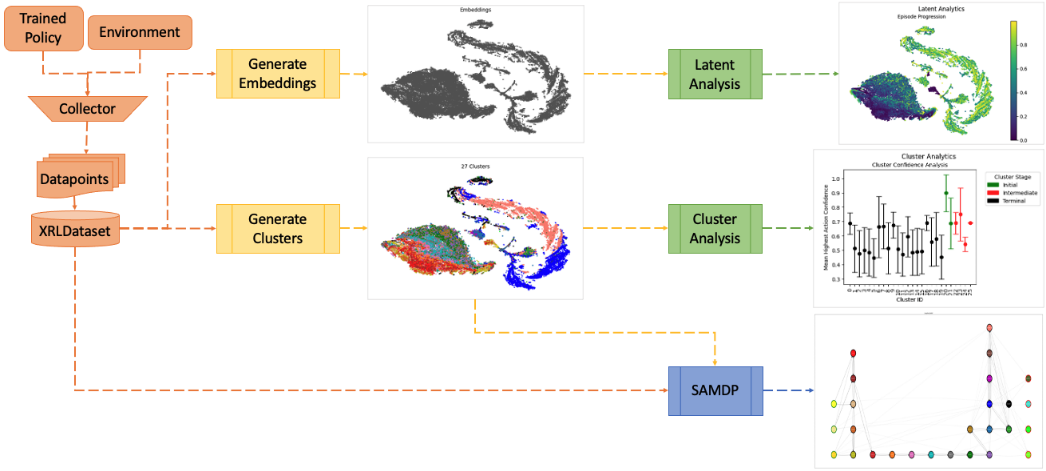

Figure 2: Example workflow to generate vulnerability analysis visualizations using the ARLIN Toolkit.

To illustrate ARLIN's effectiveness, we provide explainability outputs and corresponding vulnerability analysis for a publicly available DRL model - a model trained using Stable Baselines3 [17] with PPO [18] on OpenAI gym's Lunarlander-v2 environment [19], pulled from Huggingface.com - by following the steps outlined in Figure 2. The output visualizations and analysis can be found in Appendix A (latent analysis), Appendix B (cluster analysis), and Appendix C (SAMDP analysis).

6 Discussion and Future Work

We believe that ARLIN can accelerate research in the XRL field by providing a modular research library with an easy-to-use API for generating explainability visualizations for vulnerability and critical point identification and analysis. This work can be applied to practical use domains such as RL-assisted autonomous vehicle verification and validation and the field of adversarial RL. We hope that the library can expand to include additional analytics, metrics, and visualizations as well as add support for new algorithms and frameworks out of the box through continued author maintenance and community development.

Acknowledgments and Disclosure of Funding

The authors thank Walker Dimon and Guido Zarrella for helpful discussions throughout the development process. This work was funded by the 2023 MITRE Independent Research and Development Program's Early Career Research Program.

References

[1] D. Silver, A. Huang, C. Maddison *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, p. 484–489, 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[3] A. P. Pope, J. S. Ide, D. Micovic, H. Diaz, D. Rosenbluth, L. Ritholtz, J. C. Twedt, T. T. Walker, K. Alcedo, and D. Javorsek, "Hierarchical reinforcement learning for air-to-air combat," 2021.

[4] J. Degrave, F. Felici, J. Buchli *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, p. 414–419, 2022. [Online]. Available: https://doi.org/10.1038/s41586-021-04301-9

[5] B. Gaudet and R. Furfaro, "Terminal adaptive guidance for autonomous hypersonic strike weapons via reinforcement learning," 2021.

[6] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[7] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[9] S. R. Islam, W. Eberle, S. K. Ghafoor, and M. Ahmed, "Explainable artificial intelligence approaches: A survey," 2021.

[10] S. Milani, N. Topin, M. Veloso, and F. Fang, "A survey of explainable reinforcement learning," 2022.

[11] P. Sequeira and M. Gervasio, "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations," *Artificial Intelligence*, vol. 288, p. 103367, 2020.

[12] Y. Lan, X. Xu, Q. Fang, Y. Zeng, X. Liu, and X. Zhang, "Transfer reinforcement learning via meta-knowledge extraction using auto-pruned decision trees," *Knowledge-Based Systems*, vol. 242, p. 108221, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705122000624

[13] N. Baram, T. Zahavy, and S. Mannor, "Deep reinforcement learning discovers internal models," 2016.

[14] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[15] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[16] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. Society for Industrial and Applied Mathematics, 2007, p. 1027–1035.

[17] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[19] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

A Latent Analysis Examples

ARLIN's latent analysis methods make use of the embeddings generated by ARLIN's generation component by overlaying user-defined policy metadata over the generated embeddings to visualize how the metadata relates to location within the embedding space. This information can be helpful when working examining the latent space of a policy. Future work can make use of the latent space to identify similar datapoints or regions, or identify ways to traverse the latent space to reach specific outcomes determined by the metadata, such as actions to take.

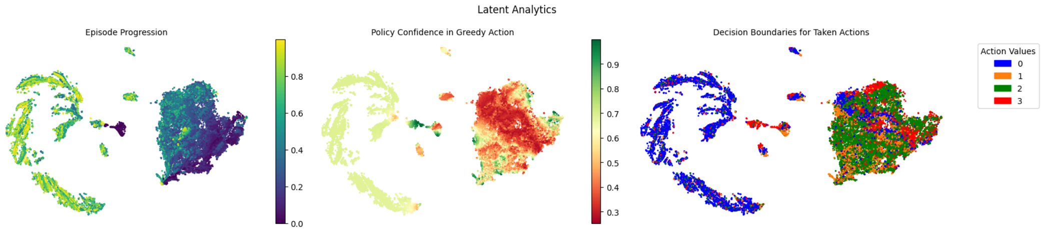

Figure 3: Example outputs from ARLIN's Latent Analysis methods. Graphics show policy metadata overlaid onto embeddings generated from ARLIN's generation component. Overlaid policy metadata overlaid onto embeddings from left to right: episode step that the datapoint was taken at, highest probability from the datapoint's action distribution, action taken at the given datapoint

B Cluster Analysis Examples

ARLIN's cluster analysis methods make use of the clusters generated by ARLIN's generation component by computing the average values of different policy metadata for each identified cluster. This information gives insight into vulnerable clusters and states within the environment that are reached by the policy.

The confidence analysis gives insight into how confident the policy is in the action that it is taking at a given point in time. Clusters with low confidence indicate areas of the environment where the policy is not confident in the action that it is taking due to limited training, particularly difficult areas of the environment, or areas where the policy action has no consequence. Clusters with high confidence are areas where the policy is sure of the action it is taking, which can be representative of an easy or very important cluster. A large variance typically represents a cluster where the policy is either very sure or very unsure of its actions, likely resulting in a higher likelihood for mistakes.

The expected return analysis gives insight into both the stage of the episode the cluster is in (early vs late) as well as insight into which states the policy thinks have a higher likelihood for mission success. When looking at initial clusters, a cluster that has a lower expected return is seen as a harder starting position for the policy. When looking at intermediate clusters, clusters with a higher expected return represent "early" stage clusters while "late" stage clusters have a lower expected return.

The reward analysis gives insight into how good the actions taken within the cluster are, represented by the amount of reward received. A higher average reward means the actions are considered better overall. When analyzing terminal clusters, mission failures can be typically be identified by clusters with a large negative reward.

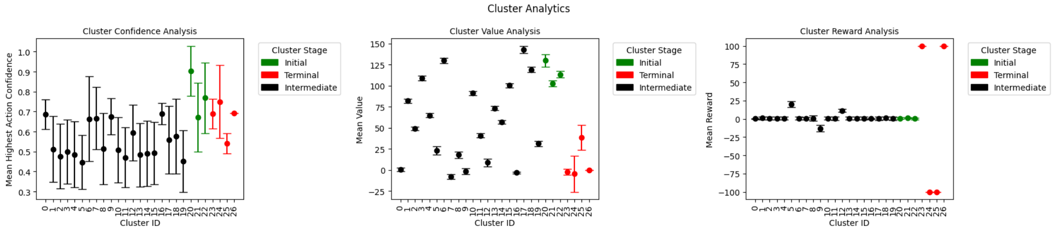

Figure 4: Example outputs from ARLIN's Cluster Analysis methods. Cluster analytics give average metrics for each cluster generated by ARLIN's generation component. From left to right: average greedy action confidence per cluster, average expected return per cluster, average reward per cluster.

In Figure 4, we can make a few assumptions about the clusters within our policy. When analyzing initial clusters, Cluster 21 has a low confidence and low expected return, indicating that the cluster is seen as a non-optimal starting position. The policy does not expect to get as much reward overall when starting in Cluster 21 than Cluster 20. When looking at intermediate cluster, Cluster 9 shows a low received reward but high confidence, indicating that it is likely a corrective maneuver that the policy feels is important to take. We can assume this is a late-stage maneuver as well given that the expected return is near 0. For terminal clusters, Cluster 23 has a low expected return and a low received reward, meaning this is likely an expected failure - the policy was expecting a low reward and got a low reward. Cluster 24, however, has a high expected reward but a low received reward, indicating an unexpected failure - the policy was expecting to get more than it received.

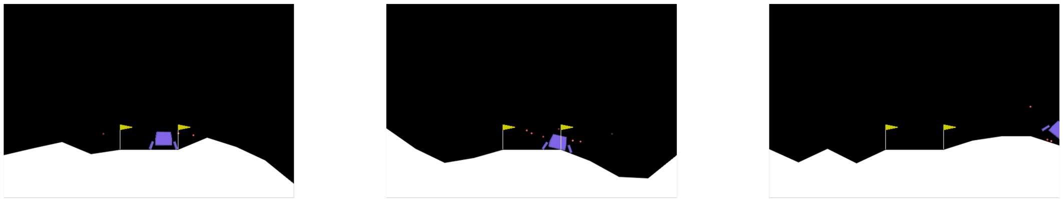

Figure 5: Cluster state analysis from ARLIN showing example images from different clusters. From left to right: Cluster 9 (intermediate), Cluster 23 (terminal), Cluster 24 (terminal).

As seen in Figure 5, our assumptions were correct. Cluster 9 is a late-stage corrective maneuver in which the agent is attempting to move further left to be inside the landing flags. Cluster 23 is an expected failure where the policy lands hard into the ground and crashes, and Cluster 24 is an unexpected failure in which the policy moves off screen, resulting in the end of the episode without a successful landing or a crash.

C SAMDP Examples

ARLIN's SAMDP component uses metadata from the XRLDataset along with the generated clusters to generate a semi-aggregated Markov decision process of the policy to show how the policy moves between clusters over the course of an episode. This information is useful in identifying paths between clusters. For vulnerability analysis, this is useful in identifying which actions lead an agent to mission failure, and which actions lead to mission success as well as identifying the critical points where the agent can go either way depending on the actions taken. ARLIN provides a variety of methods in the SAMDP package including holistic views of the entire SAMDP (Figure 6), paths between given clusters, or paths leading into a terminal state (Figure 8). All SAMDP methods can provide a full verbose view including the actions necessary for the movement (Figure 6), or a simplified view which only shows the connections and not the actions required (Figure 7). Some methods provide the option to only show the most probable connections as well, to avoid connections that are have been taken at least once, but are not likely to be taken by the policy in general.

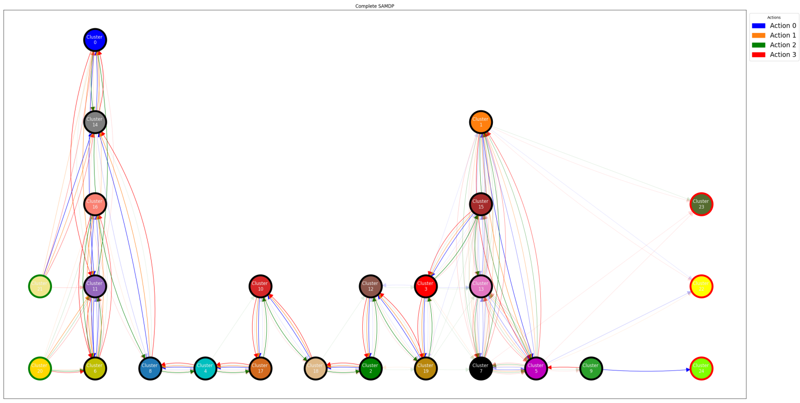

Figure 6: Fully verbose SAMDP showing connections between clusters and the action value that creates the movement.

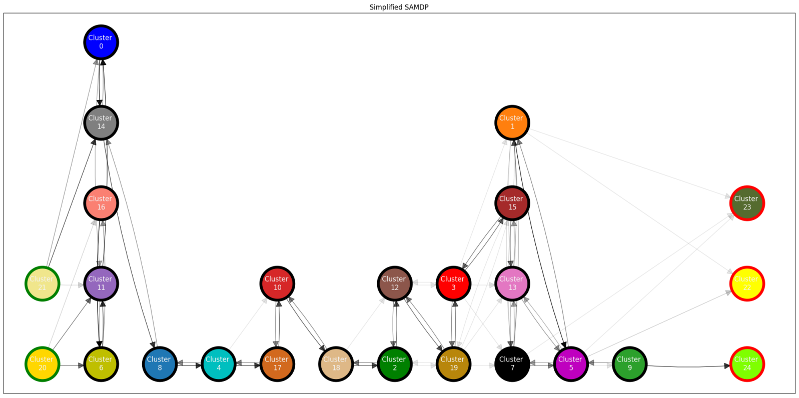

Figure 7: Simplified SAMDP view showing conenctions between clusters regardless of taken action.

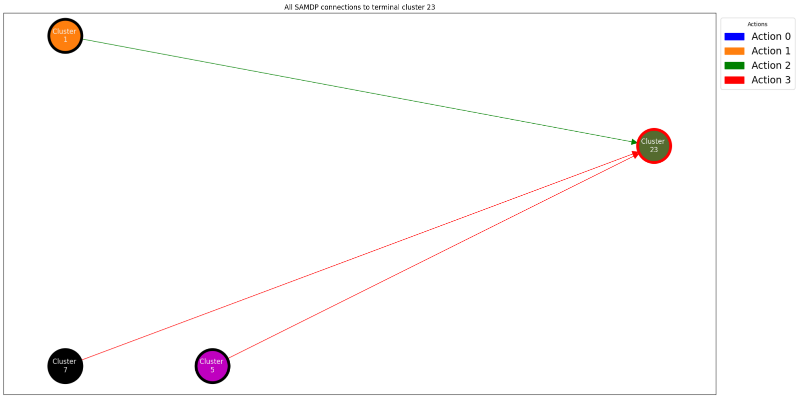

Figure 8: SAMDP view to show movements into all terminal clusters and the actions that bring the agent from a given cluster into the terminal clusters.