Transparency of Task Dependencies of Reinforcement Learning in Unmanned Systems

Yiwei Shi, Weiru Liu
School of Engineering Mathematics and Technology, University of Bristol, Bristol, U.K.
{yiwei.shi, weiru.liu}@bristol.ac.uk

***Abstract*—Reinforcement learning has been applied widely, however there are yet no easy ways to explain both an agent's behaviour or the complex unmanned systems that an agent is in. In this paper, we design a framework and its corresponding algorithm to discover the hidden rules or relationships in the unmanned systems supported by policies obtained through Reinforcement Learning training. It is demonstrated experimentally that our approach is effective in building an explicit representation for revealing the hidden rules or constraints when an agent needs to accomplish some tasks in that unmanned systems.**

***Index Terms*—Explainable Artificial Intelligence, Explainable Reinforcement Learning, Task Dependencies, Unmanned Systems**

I. Introduction

In the wake of rapid advancements in artificial intelligence technology, the decision-making prowess of unmanned systems [1]–[6] has soared, fueled by sophisticated algorithms. These systems, now equipped with strategies to execute complex tasks, play a critical role in key domains like unmanned logistics [7], [8], remote sensing detection [9], [10], and automated control [11], [12]. Central to this advancement is Reinforcement Learning (RL), an AI paradigm renowned for optimizing decision processes. However, RL's inherent lack of transparency in decision-making presents a significant challenge in unmanned systems, especially in applications where trust and verifiability are crucial.

Consequently, the handover of custom AI modules as inscrutable, pre-trained neural models to users further compounds this issue in unmanned systems. While these systems are proficient in performing assigned tasks, their operational logic often remains obscure, particularly to users without extensive AI expertise. Hence, developing methodologies to provide clear, cogent explanations of these decision-making processes is essential for building trust in unmanned systems.

As performance in RL-based unmanned systems has advanced, the demand for explainability has surged, aiming to enhance system transparency and user trust. Current RL-based unmanned systems embed the relationships and constraints among various components within the neural network's parameters. Yet, directly presenting these intricate parametric representations as explanations can be bewildering to both users and developers of unmanned systems.

Explainable Artificial Intelligence (XAI) technologies [13]–[15] in RL shed light on the behaviors of agents within unmanned systems. An RL policy can be interpreted as the agent's behavioral pattern in response to its surroundings in these systems. In simpler task settings within unmanned systems, agents exhibit uniform behaviors, making explanations more straightforward. However, in environments with multiple tasks and interdependencies, agent behaviors diverge considerably, given the presence of various optimal policies in unmanned systems. This complexity makes generating explanations for a single policy less convincing, as it might lack generalizability across different scenarios in unmanned systems.

In the realm of explainable reinforcement learning in unmanned systems, efforts like those in [16] and [17] have modeled the unmanned systems with surrogate models to elucidate agent behaviors. These models, however, often suffer from sampling limitations, leading to incomplete representations and less persuasive explanations. Counterfactual explanations, as explored in [18], attempt to rationalize an agent's specific actions within unmanned systems but are similarly constrained by sampling issues and verification challenges. Approaches like [19] and [20], which utilize environmental knowledge or decision trees, provide intuitive policy representations but struggle with generalization or overfitting, especially in complex dependency scenarios within unmanned systems. Most current RL explanations cater to single-task environments, leaving a gap in multi-task settings in unmanned systems.

Addressing this, our study focuses on multi-task RL environments within unmanned systems, where different optimal policies might emerge under varying circumstances. Merely illustrating these policies falls short of revealing the deeper constraints and relationships in unmanned systems. Our research aims to uncover these hidden constraints and relationships through an agent's trial-and-error in partially known unmanned systems. We investigate whether these constraints and relationships can be explicitly mapped if an agent undergoes sufficient attempts to accomplish diverse tasks within unmanned systems.

Our contributions in this paper are twofold: 1) we propose a post-hoc explanation framework for unmanned systems, and 2) we develop an algorithm to mine inter-task dependencies in these systems. This framework gathers data from the RL training phase in unmanned systems—where policies are sufficiently trained—and processes this data to create a dependency graph. This graph represents the constraints or rules within unmanned systems, adhered to by all learned policies. Our experimental design demonstrates these rules or

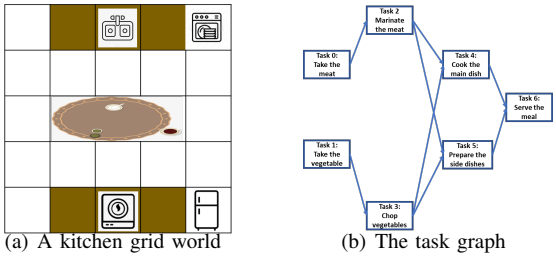
constraints as inferred from policy trajectories in unmanned systems.

II. Preliminaries and Notation

Markov decision process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ where $\mathcal{S}$ indicates a set of states, $\mathcal{A}$ indicates a set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ indicates a transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ indicates a reward function and $\pi : \mathcal{S} \to \mathcal{A}$ indicates a policy. Markov process can be denoted as a tuple ($\mathcal{S}$, $\mathcal{T}$).

Definition II.1. *A network $(\mathcal{V}, \mathcal{E})$ is a directed acyclic graph (DAG) where $\mathcal{V} = \{v_1, v_2, ..., v_m\} \subset \mathcal{S}$ is a finite set of nodes and $\mathcal{E} = \{(v_i, v_j) | i, j = 1, 2, ..., m\}$ is a set of directed edges over $\mathcal{V}$.*

In our study, we model tasks within autonomous systems as nodes, forming what we term a task network or task graph. Take, for instance, the scenario depicted in Figure 1, which demonstrates the interdependencies among tasks. Here, an autonomous agent—or robot—is programmed to execute seven distinct tasks including "taking the meat out" (as shown in Figure 1(b)), "cooking the main dish," and others. To accomplish these tasks, the agent must navigate to specific locations such as the position of the fridge (illustrated in Figure 1(a)) or the sink (for washing vegetables).



(a) A kitchen grid world (b) The task graph

Fig. 1. An illustration scenario for tasks in RL

In a task network, represented as (V, E), there may be multiple initial task nodes. These are tasks that are independent and do not rely on the completion of any other tasks—for instance, "taking the meat out" or "taking the vegetables." A task can commence only after all its preceding tasks have been fulfilled. For instance, within the task graph shown in Figure 1, task 0 and task 1 are both initial nodes and can be started without any prerequisites. These independent tasks are termed root nodes, like task 0 and task 1, while those without any subsequent tasks are designated as leaf nodes, such as task 6.

As depicted in Figure 1, the environment features five objects: the sink, dishwasher, dining table, cooker, and fridge. The agent can perform seven actions: move forward (↓), move backward (↑), move left (←), move right (→), carry out operation 1 (op 1), perform operation 2 (op 2), and no operation (no-op).

The agent maneuvers to the location of specific objects to perform tasks associated with that object. For instance, it approaches the fridge and engages in operation 1 to accomplish task 0 or operation 2 for task 1. At the sink, the agent executes task 2 with operation 1 and task 3 with operation 2. Near the cooker, it carries out operation 1 for task 4 and operation 2 for task 5. Finally, by reaching any of the three spots around the dining table, the agent can perform operation 1 to complete task 6.

Definition II.2. *Let $h = (s_1, a_1, \ldots, s_t, a_t) \subset \mathcal{H}$ be a trajectory of a policy. Let $H = [h_1, h_2, \cdots, h_n]^T$ be an $1 \times n$ metrix, containing n trajectories, where $[\cdot]^T$ indicates matrix transposition.*

Definition II.3. *Let O be a task in $\mathcal{V}$ and h be a policy. O is said to be complete in h for timesteps $[i, \cdots, t]$ if either O has no predecessor tasks or every predecessor task of O has been complete before timestep i.*

Definition II.4. *Let $(s_{t_i}, a_{t_i}, s_{t_i+1})$ (resp. (s_{t_i}, s_{t_i+1})) be the completion marker for task i in a trajectory of MDP (resp. MP). If $(s_{t_i}, a_{t_i}, s_{t_i+1})$ exists in trajectory h (resp. h'), then this indicates that task i has completed at time t_t in h (resp. h'), where the only difference between h and h' is that h' does not explicitly specific action a_t.*

Note that the marker for task i completion may not be unique.

Definition II.5. *Let M_o^a be a after-relationship vector for task O where the elements in M_o^a denote all the tasks that can be executed after O is complete. Let M^a be the matrix for all the m tasks' after-relationship vectors, then M^a is represented as:*

$$M^a = [M_1^a, M_2^a, \cdots, M_m^a]^T = \begin{bmatrix} M_{11}^a & M_{12}^a & \cdots & M_{1m}^a \\ M_{21}^a & M_{22}^a & \cdots & M_{2m}^a \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1}^a & M_{m2}^a & \cdots & M_{mm}^a \end{bmatrix}$$

As an example, the after-relationship vector for each task in Figure 1 is shown below. In a after-relationship vector such as for task O_0, value 1 in the vector indicates that that corresponding task shall be executed after task O_0. Note, we have given each task an order number, as shown in Figure 1, such as task 0 is for 0, task 1 is for 1, task 3 is for 3 $\cdots$, and all the after-relationship has the same ordering.

Definition II.6. *Let M_o^b be a before-relationship vector for task O where the elements in M_o^b denote all the tasks that can be executed before O is complete. Let M^b be the matrix for all the m tasks' before-relationship vectors, then M^b is represented as:*

$$M^b = [M_1^b, M_2^b, \cdots, M_m^b]^T = \begin{bmatrix} M_{11}^b & M_{12}^b & \cdots & M_{1m}^b \\ M_{21}^b & M_{22}^b & \cdots & M_{2m}^b \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1}^b & M_{m2}^b & \cdots & M_{mm}^b \end{bmatrix}$$

As an example, the before-relationship vector for each task in Figure 1 is shown below. In a before-relationship vector such as for task O_0, value 1 in the vector indicates that that corresponding task shall be executed before task O_0. Here, we

follow the same order numbers for tasks as before, such as task 0 is for $0 \cdots$.

Definition II.7. *Let M_o^u be a free-relationship vector for task O where the elements in M_o^u denote all the tasks that can be executed either after or before O is complete. Let M^u be the matrix for all the m tasks' free-relationship vectors, then M^u is represented as:*

$$M^u = [M_1^u, M_2^u, \cdots, M_m^u]^T = \begin{bmatrix} M_{11}^u & M_{12}^u & \cdots & M_{1m}^u \\ M_{21}^u & M_{22}^u & \cdots & M_{2m}^u \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1}^u & M_{m2}^u & \cdots & M_{mm}^u \end{bmatrix}$$

As an example, the free-relationship vector for each task in Figure 1 is shown below. In a free-relationship vector such as for task O_0, value 1 in the vector indicates that that corresponding task shall be executed after and before task O_0.

Note: The free vector of task i M_i^u is the form of the intersection of before set X_i^b and after set X_i^a of task i.

Definition II.8. *Let $\overline{M_o^b}$ be a predecessors-relationship vector for task O where the elements in $\overline{M_o^b}$ denote all the tasks that can **only** be executed before O is complete. Let $\overline{M^b}$ be the matrix for all the m tasks' predecessors-relationship vectors.*

As an example, the predecessors-relationship vector for each task in Figure 1 is shown below. In a predecessors-relationship vector such as for task O_0, value 1 in the vector indicates that that corresponding task shall be executed before task O_0.

Note: predecessors set and before set are different relationships. The former indicates all tasks that are preceding this task, i.e., must be executed before this task, and the latter indicates all tasks that can appear before this task, i.e., can be executed before this task. If a task cannot be executed after this task i, then this task can only be executed before this task, i.e. this task is one of the predecessors of task i.

Definition II.9. *Let $\overline{M_o^a}$ be a successor-relationship vector for task O where the elements in $\overline{M_o^a}$ denote all the tasks that can **only** be executed after O is complete. Let $\overline{M^a}$ be the matrix for all the m tasks' successor-relationship vectors.*

As an example, the successor-relationship vector for each task in Figure 1 is shown below. In a successor-relationship vector such as for task O_0, value 1 in the vector indicates that that corresponding task shall be executed after task O_0.

Theorem II.1. *If $v_i \cup X_i^a$ contains all the tasks, then node v_i is a root node.*

For example, take the scenario in Figure 1. For task 0, the union of the set $\{0\}$ corresponding to task 0 and the set of its after set $\{1, 2, 3, 4, 5, 6\}$ is the full set $\{0, 1, 2, 3, 4, 5, 6\}$, so task 0 is a root node. Similarly, task 1 can be obtained as a root node.

Theorem II.2. *If $v_i \cup X_i^b$ contains all the tasks, then node v_i is a root node.*

For example, take the scenario in Figure 1. For task 6, the union of the set $\{6\}$ corresponding to task 6 and the set of its before set $\{0, 1, 2, 3, 4, 5\}$ is the full set $\{0, 1, 2, 3, 4, 5, 6\}$, so task 6 is a leaf node.

Theorem II.3. *Let R_v be the set of leaf nodes, and let $R' \subseteq R_v$. Let v_i be a node. If $R' \cap X_i^a = \emptyset$ and $R' \cup X_i^a = \mathcal{V}$, then there is an edge between v_i, v' where $v' \in R'$.*

For example, take the task dependency for the scenario in Figure 1. In this graph, the root set is $\{0, 1\}$. For task 2, the intersection of its after set $\{1, 3, 4, 5\}$ with a subset $\{0\}$ of the root set is the empty set $\{\ \}$, and the union is the full set $\{0, 1, 3, 4, 5, 6\}$ (excluding task 2) , so there exists an edge $(0, 2) \in \mathcal{E}$ between task 0 and task 2.

Theorem II.4. *Let R_v be the set of leaf nodes, and let $R' \subseteq R_v$. Let v_i be a node. If $R' \cap X_i^b = \emptyset$ and $R' \cup X_i^b = \mathcal{V}$, then there is an edge between v_i, v' where $v' \in R'$.*

For example, take the task dependency for the scenario in Figure 1. In this graph, the root set is $\{6\}$. For task 4, the intersection of its before set $\{0, 1, 2, 3, 5\}$ with a subset $\{6\}$ of the root set is the empty set $\{\ \}$, and the union is the full set $\{0, 1, 2, 3, 5, 6\}$ (excluding task 4), so there exists an edge $(4, 6) \in \mathcal{E}$ between task 4 and task 6.

The relationships between different sets are summarized as follows:

$\widetilde{M^u} = [\widetilde{M_1^u}, \widetilde{M_2^u}, \ldots, \widetilde{M_m^u}]^T$

$\widetilde{M^b} = [\widetilde{M_1^b}, \widetilde{M_2^b}, \ldots, \widetilde{M_m^b}]^T$

$\widetilde{M^a} = [\widetilde{M_1^a}, \widetilde{M_2^a}, \ldots, \widetilde{M_m^a}]^T$

$M^b = [M_1^b, M_2^b, \ldots, M_m^b]^T = \widetilde{M^a} + M^u$

$= [\widetilde{M_1^b} + M_1^u, \widetilde{M_2^b} + M_1^u, \ldots, \widetilde{M_m^b} + M_1^u]^T$

$M^a = [M_1^a, M_2^a, \ldots, M_m^a]^T = \widetilde{M^b} + M^u$

$= [\widetilde{M_1^b} + M_1^u, \widetilde{M_2^b} + M_1^u, \ldots, \widetilde{M_m^b} + M_1^u]^T$

M^a is transpose matrix of M^b, i.e. $M^a = {M^b}^T$, $\widetilde{M^a}$ is transpose matrix of $\widetilde{M^b}$, i.e. $\widetilde{M^a} = \widetilde{M^b}^T$, M^u is symmetric matrix, i.e. $M^u = {M^u}^T$.

Constraints (or rules) are crucial for ensuring proper functioning of a multi-task environment in reinforcement learning problems. Obtaining all the rules through a single policy is difficult, as the agent may only partially comply with the rules during training. Another challenge is that providing a stable explanation during the training process is difficult as the environment-specific rules are hard to obtain through a complete RL model. The policy obtained through RL only partially reflects some of the rules in the environment. To achieve a more comprehensive understanding of these rules, we implement a post-hoc architecture in Figure. 2. to address this issue, our contribution is mainly on the left side of the figure.

III. Methodology

A. Architecture

The Figure. 2 consists of two phases: the RL phase and the explanation phase, the former focuses on training policies

in RL and the latter on an online data collection phase and an offline explanation generation phase. We pre-process the data obtained in offline phrase by function Φ in Equation 1, where Φ is used to retain the necessary information (H') in the trajectories of policies (H) and the explanation model Ψ in Equation 2 is constructed to generate explanations as needed.

$$H' = \Phi(H) \tag{1}$$

$$(V, E) := \Psi(H') \tag{2}$$

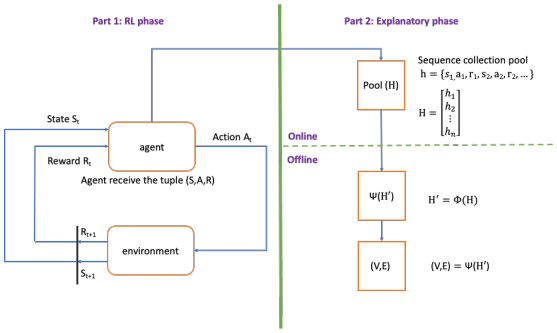


Fig. 2. Task Explanation Architecture diagram

Data pre-processing Φ entails incorporating prior knowledge, such as identifying the end-of-task state, to filter the data for Environmental rules or relationships between tasks generation. The filtering eliminates unneeded state s and action a in h or H, retaining only necessary data (from H to H'). The main objective in the data pre-processing is to obtain before- and after-relationship matrices (M^a and M^b) for tasks. Once these relationships are extracted, a task graph (V, E) is generated. This graph represents the hidden constriants in the environment which are not known to the agent during the training stage. However, through sufficient training, these hidden constriants or rules are learned and are used for generating explanations.

B. Data pre-processing

We leverage this architecture to convert an explanation problem into a data mining problem. H cannot be directly analyzed due to the diversity of states in the presence of numerous trajectories of policies.

The markers for each task completion need to be sorted in advance. Then these markers are numbered according to the task index. Finally the trajectories H are traversed keeping only the sequences of completed task numbers as H'.

The agent usually receives the corresponding reward after completing a task. suppose the agent completes task i at time t and takes the tuple (s_t, a_t, r_t, s_{t+1}) in this trajectory. The reward returned by the environment at this time is usually greater than 0, because the reward returned by the environment in a multitasking environment tends to be 0 except for the reward returned by the environment when the task is completed. If the tuple (s_t, a_t, r_t, s_{t+1}) does not consider reward r, then (s_t, a_t, s_{t+1}) can still be used as a marker to judge task completion because (s_t, a_t, s_{t+1}) is a marker for task i

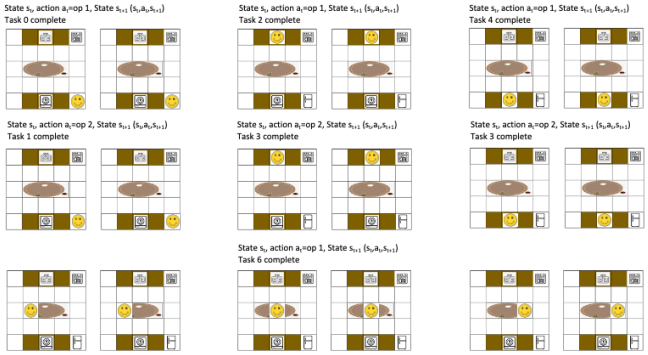


Fig. 3. Transformation diagram :the tuple (s_t, a_t, s_{t+1}) corresponding to the task completion marker, where each state is presented in pairs: the first representing s_1 and the second representing s_2. The actions executed by the agent transitioning from s_1 to s_2 are annotated above the corresponding state pair in the figure. Markers correspond uniquely to tasks 0 through 5, with the exception of task 6, which is associated with multiple markers.

completion, then (s_t, a_t, s_{t+1}) is in one-to-one correspondence with task i, and vice versa does not hold.

An example of a scenario in Figure 1 is given for the data pre-processing. A trajectory consisting of 22 time steps is shown in Figure 4, where each image representing a state and the action being performed is recorded on the top of the image. The new sequence is obtained by matching the task completion marks in the trajectory with the help of the transformation diagram in Figure 3. The information in the trajectory that cannot match the tuple (s_t, a_t, s_{t+1}) in the transformation diagram is directly ignored and that can match (s_t, a_t, s_{t+1}) will be used to generate a new sequence based on the task number . The trajectory is symbolized as h=(s_0,↓,s_1,↓,s_2,op1,s_2,op2,s_2,↑,s_1,↑, s_0,↑,s_3,←,s_4,←,s_5,↑,s_6,op1,s_6,op2, s_6,↓,s_7,↓,s_8,↓,s_9,↓,s_{10},op1,s_{10},op2, s_{10},↑,s_9,↑,s_8,op1,s_8),where the action space of agent is [←, →, ↑, ↓, op1, op2, no-op]. From the transformation diagram, it can be obtained that (s_2,op1,s_2) is the marker for task 0 completion, (s_2,op2,s_2) is the marker for task 1 completion, (s_6,op1,s_6) is the marker for task 2 completion, (s_6,op2,s_6) is the marker for task 3 completion, (s_{10},op1,s_{10}) is the marker for task 4 completion, (s_{10},op2,s_{10}) is the marker for task 5 completion, and (s_8,op1,s_8) is the marker for task 6 completion. Replacing the markers with task markers in the order of matching to get the new sequence [0,1,2,3,4,5,6] . Then the sequence obtained after data processing is [0,1,2,3,4,5,6].

This extends to general forms. Assume a trajectory is h = (s_1,a_1,s_2,s_3,...,s_{t1},a_{t1},s_{t1+1},...,s_{t2},a_{t2},s_{t2+1},...,s_{t3},a_{t3},s_{t3+1},...), where states s_{t1+1}, s_{t2+1}, s_{t3+1} are task 1, task 2 and task 3 at completion respectively. But it is not possible to determine whether the current task can be completed by these states alone. The sequence of arriving at these states is not unique, but the sequence of completing the task is deterministic, i.e. $(s_{t1}, a_{t1}, s_{t1+1})$, $(s_{t2}, a_{t2}, s_{t2+1})$, $(s_{t3}, a_{t3}, s_{t3+1})$ are the

Fig. 4. A trajectory for the scenario

markers for the completion of task 1, task 2 and task 3 respectively. Storing these sequences in a list and encoding them by task number, traversing the trajectory h in H to match the sequences in this list yields H'.

So the judgment criterion should be based on the sequence (s_t, a_t, s_{t+1}) instead of (s_t, s_{t+1}) and $(a_0, ..., s_t, a_t, s_{t+1})$. (s_t, s_{t+1}) in Markov Process can be used as a flag to determine whether a state is the end of a certain task. This process does not involve action and each state is unique in the state set. In MDP, however, there exist multiple actions that can lead state s_t to state s_{t+1}. For example, if s_t and s_{t+1} are the same state and the action set contains multiple non-moving actions (possibly functional actions), these actions do not change the state vector. So it is not possible to identify whether the task is completed or not just by the state-pair. The reason why an action sequence cannot serve as a criterion for judgment is that it is dependent on the initial state, and different initial states can lead to different action sequences. Furthermore, even for the same initial state, there can be a diverse range of action sequences that can be used to accomplish the same task.

C. Algorithm

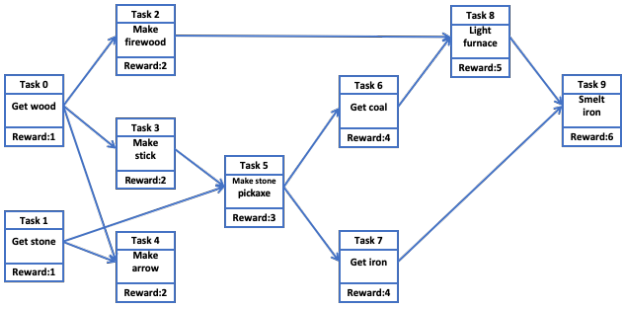


Fig. 5. An example of task graph

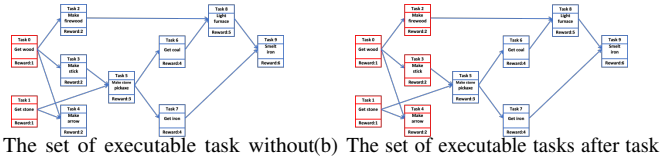
Getting the graph from H is difficult in RL because the state sequences in the trajectory only record the path that is the completion of the task, however the path does not imply the existence of an edge between two nodes. Because the connectivity between executable tasks will change with the completion of some tasks, while the structure of tasks dependency does not change. For example, in Fig 5, the partial trajectory to complete task 3 (in H') is given below (Note: the number in the sequence indicates the task number).
trajectory 1:[0,2,3,...], trajectory 2:[0,1,2,3,...], trajectory 3:[0,1,3,2,...], trajectory 4:[1,0,4,3,2,...].

The sequence of task execution reveals that task 0 and task 1 can be executed in either order, without a dependency between them as there is no connection between the two task nodes in Fig 5. This is evident from the absence of an edge in the network. However, executing task 0 changes the list of

Objective: To obtain the set of edges $\bar{\mathcal{E}}$, such that $\bar{\mathcal{E}}$ is close to $\mathcal{E}$ of graph $(\mathcal{V}, \mathcal{E})$
begin
  $\bar{\mathcal{E}} = [\,]$
  Initialize M^b : before matrix
  Initialize $\mathcal{V}$: the set of nodes in graph $(\mathcal{V}, \mathcal{E})$
  Initialize $L_{\mathcal{V}}$: the length of set of nodes in graph $(\mathcal{V}, \mathcal{E})$
  Initialize $\mathcal{V}^+$: the set of temporary nodes in each iteration
  for $L_{\mathcal{V}} > 0$ **do**
    1. Find root nodes in the current scenario
    $\mathcal{V}^+ = [\,]$
    for v_i *in* $\mathcal{V}$ **do**
      if *len*$(M^b[i])$ == $L_{\mathcal{V}}$ **then**
        $\mathcal{V}$.remove(v_i)
        $\mathcal{V}^+$.add(v_i)
      end
    end
    2. Remove the column and row where the root nodes are located
    for v_i *in* $\mathcal{V}^+$ **do**
      for v_j *in* $\mathcal{V}$ **do**
        if $M^b[j][i]$ == 0 *and* $\forall M^b[j][k]$!= 0 $(v_k$ *in* $\mathcal{V} - \mathcal{V}^+)$ **then**
          $\bar{\mathcal{E}}$.append$((j, i))$
          # $\bar{\mathcal{E}}$.append$((i, j))$ if input is matrix M^a
        end
      end
    end
    3. Remove root nodes from set of nodes
    for v_i *in* $\mathcal{V}^+$ **do**
      $\mathcal{V}$.remove(v_i)
      $M^b \leftarrow M^b$.remove$(M^b[i,:]$ and $M^b[:,i])$
    end
    $L_{\mathcal{V}} \leftarrow$ length$(\mathcal{V})$
  end
end

Algorithm 1: Generation of graph via before matrix (after matrix)

executable tasks from [0,1] in Fig.6(a) to [0,1,2,3,4] in Fig.6(b), indicating that any task in this new list can be executed next. Thus, the connectivity of executable tasks is altered as a result of task 0 execution.



(a) The set of executable task without task completion

(b) The set of executable tasks after task 0 and task 1 completions

Fig. 6. The set of executable tasks

The algorithm above was designed to obtain the graph $(\mathcal{V}, \mathcal{E})$ in a dataset H'. The algorithm is divided into three steps, which are iterated to obtain the edges in the network. Step 1: Get the root nodes/leaf nodes in the current set of tasks identified according to Theorem 1 or 2; Step 2: Get the edges in the after/before matrix according to the Theorem 3 or 4; Step 3: remove the nodes in the set of root/leaf nodes in the graph. This algorithm yields (partial) edges and a complete layered graph structure from the data pool.

For graph generation, two ideas can be devised: 1) top-down graph generation from root nodes, and 2) bottom-up graph generation. The first step of the former is to find the root nodes in the current network structure, while the latter is to find the leaf nodes and then find the neighboring layers to get the layer structure of the graph respectively. The two methods obtain different layers of the graph hierarchy because the number of layers of any node from the root nodes is hardly the same as the number of layers of the leaf nodes. For each layer obtained, the root or leaf node layer under the current structure is stored in the list and removed from v, and the obtained layer is set as the new root or leaf node layer by iterating until the last layer is found. At this point, the structure of the graph layer can be obtained, and the set of edges collected can be obtained from the over theorem. However, the sets of edges obtained by the two methods can complement each other.

IV. Experiment and Empirical Analysis

In complex environments, training a single policy in RL can be time-consuming, and training multiple policies will take even more time. To verify the feasibility of our algorithm, we conducted a two-phase experiment. In the first phase, we tested the algorithm in a simple Markov Processes (MP) environment. In the second phase, we collected data in a more complex mutil-task (MDP) RL environment using the RL algorithm and then analyzed the data offline with our algorithm to obtained constraints in the environment so that to build a task graph.

While MDP problems are more complex than MP problems, the main focus of this paper is not on solving these problems, but rather on mining dataset H used in the process of sovling them for extracting new information that can be used to build explanations. From this perspective, the pre-processed dataset H' obtained in the MP environment is structurally and functionally equivalent to the H' obtained in the MDP environment in our study.

The accuracy acc is used as an evaluation metric for this algorithm, since it is a mining task that mines the pool of data for hidden relations between nodes and stores these relations in the set $\bar{\mathcal{E}}$, and then compares them with the set of edges $\mathcal{E}$ in the network $(\mathcal{V}, \mathcal{E})$ to obtain the accuracy, where the information of the network $(\mathcal{V}, \mathcal{E})$, $\mathcal{V}$ is known while $\mathcal{E}$ is unknown for data processing, $\mathcal{E}'$ is generated from explanatory model Ψ and then compared with the real $\mathcal{E}$.

$$acc = \frac{lenth(\bar{\mathcal{E}})}{lenth(\mathcal{E})} \tag{3}$$

Note: As mentioned earlier the edges that obtained from our method are based on the matrices M^a or M^b, so there is no case of generating the wrong edges, although there may be cases of missing edges.

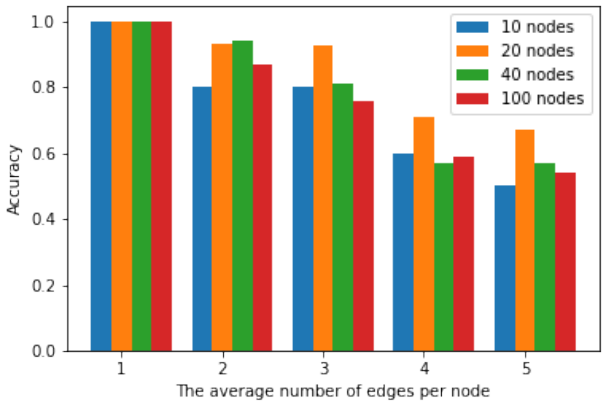


Fig. 7. Accuracy of edge generation for graphs of different sizes

A. Environment

In the first phase of the experiment, we investigated a total of 20 scenarios where the networks had 10, 20, 40, or 100 nodes, and the average number of edges per node ranged from 1 to 5 by algorithm 2. Each scenario will give 100 completely different topologies. Each topology graph is considered as a constraint in an environment. The input to Algorithm 2 consists of three parts, the pre-set graph, the number of episodes and steps. Graph represents the rules in the environment, the number of episodes and steps represent the number of samples and the number of steps performed per sample respectively.

In the second phase of the experiment, we employed the multi-tasking grid world environment described in [19] to train policies and sample data, utilizing the hierarchical reinforcement learning (HRL) method in [21].

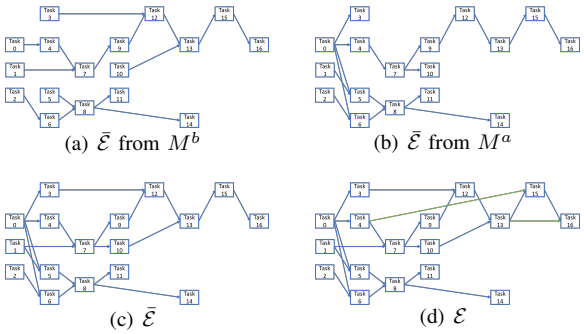
Note [21] and [19] are two methods aiming to solve the same problem, and they both use environment mining. [19] presents some optimizations to the environment described in [21]. In an

optimized environment, the agent will automatically search for and move to the nearest object (All tasks of the agent have to be executed in a specific object), which undoubtedly reduces the exploration costs significantly. The environment contains 26 subtasks with complex dependencies between them, which are represented as graphs. In total, there are 400 subgraphs of these dependencies provided.

B. Function

Stage 1: These pre-sets are considered as constraints or rules for the graph, i.e. $(\mathcal{V}, \mathcal{E})$, to be sampled to get sufficient sequences. The sequences are considered as H, filtered to get H' and then $(\mathcal{V}, \bar{\mathcal{E}})$ is generated by the algorithm. The results of the experiment are shown in Figure. 7.

The experiments reveal that the algorithm can generate almost all edges when the average number of edges is 1. When the average number of edges is 2 or 3, the algorithm's accuracy in generating edges is above 80%, and can even exceed 90%, even when the graph size reaches 100 nodes. However, when the average number of edges in the graph is larger than 3, the algorithm's performance drops significantly, and the accuracy rate remains at around 56%.



(a) $\bar{\mathcal{E}}$ from M^b (b) $\bar{\mathcal{E}}$ from M^a (c) $\bar{\mathcal{E}}$ (d) $\mathcal{E}$

Fig. 8. Visualisation $\mathcal{E}$ of the edge set

As the graph size increases, the accuracy of the generated edges decreases, but all the edges obtained are correct, which indicates that many edges are lost in the process. By visualizing our generated graph and comparing it with the original graph, we found that missing edges have two properties: 1) the edge involves a node that may exist in multiple layers (e.g., both in layer 3 and layer 5), and 2) the nodes of the edge span multiple layers, i.e., two nodes are not adjacent in the layers. This does not imply that our algorithm is entirely incapable of handling graphs with these two types of properties. However, if the graph contains more of these structures, the generated graphs will be less accurate.

Stage 2: After stage 1, we applied our method to the RL environment to test it and try to get the constraints or rules, i.e. $(\mathcal{V}, \mathcal{E})$, in the environment. Use the RL method to train the policy π and collect H, then use the prior knowledge in the environment to obtain H'. M^a and M^b are obtained from H'. Then with the help of our algorithm to generate the obtained restrictions $(\mathcal{V}, \bar{\mathcal{E}})$.

Objective: To obtain the set of edges $\mathcal{H}$
Function *search_reachable_nodes(O_t, G)* is a search algorithm return a successor task set of task O_t from the adjacency matrix G of graph $(\mathcal{V}, \mathcal{E})$. **begin**
  $\mathcal{H} = [\ ]$
  Initialize episodes : the number of episode
  Initialize steps : the number of step
  Initialize G : the adjacency matrix of graph $(\mathcal{V}, \mathcal{E})$
  Initialize $L^e = L_0^e$: the sets of executable task
  Initialize L^c : the sets of completed tasks
  L^O: a set of tasks that can be completed after the completion of task O
  for *episode in range(episodes)* **do**
    $L^e = L_0^e$, $O_0 = \pi(L^e)$, $L^c = \{O_0\}$, $O_t = O_0$, $h = [\ ]$
    h.append(O_t)
    for *step in range(steps)* **do**
      L^o = search_reachable_nodes(O_t, G)
      L' = [ ]
      for O_i *in* L^O **do**
        if O_i *not in* L^e **then**
          L'.append(O_i)
        end
      end
      $O_{t+1} = \pi(L^e + L')$
      if O_{t+1} *not in* L^e **then**
        L^e.add(O_{t+1})
      end
      L^c.add(O_{t+1})
      h.append(O_{t+1})
    end
    $\mathcal{H}$.append(h)
  end
end

Algorithm 2: Sampling Code

In the environment, the trajectories of a policy obtained from one complete RL training tend to reflect only one behavior of the agent. Although the diversity of trajectories is high in the exploration phase of RL, however, after the policy converges, the trajectories will become gradually singler. It is difficult for us to get the complete rules if we only pass a few complete training. Due to the low sample richness of one training, we trained the policy 100000 times and recorded all trajectories in order to collect enough policies (π). After preprocessing we obtained M^a and M^b, and the final generated graph.

Five environments with different restriction structures were selected and after running, the experimental results were recorded in Table I. From the experimental results, it can be found that our method performs very well in handling this task, with accuracy rates all over 90%.

C. Visualisation Rules in Environment

An example of the Visualisation Stage 2 experiment is expanded in this section. The graph structure of the constraints in the environment is shown in Fig. 8(d). Each node in the

TABLE I
EXPERIMENTAL RESULTS IN RL ENVIRONMENT

No.	$\mathcal{V}$	Layers	$\bar{\mathcal{E}}$	$\mathcal{E}$	Acc
1	16	8	20	22	0.909
2	18	10	20	22	0.909
3	16	6	19	19	1
4	19	10	22	24	0.916
5	16	9	18	20	0.9
6	17	7	20	21	0.952

graph represents a task and the edges represent the relationships between tasks. The matrices M^a and M^b be obtained from the trajectory data collected from the RL phase. The graphs generated with M^b or M^a as input with the help of the algorithm are shown in Fig. 8(a) and Fig. 8(b), respectively. The graphs generated by the union of the sets of edges of both graphs are shown in Fig. 8(c).

A more complete environment rule can be obtained from these generated graph structures (the green edges in Fig. 8(d) are the missing edges). If the graph is obtained by only one input matrix (M^a or M^b) as input to the algorithm much rule information is lost, but combining the results of both gives a more full set of rules.

V. CONCLUSION

In this research, set in the context of unmanned systems, we introduced a post-hoc explanation framework and an algorithm designed to elucidate and display the policy and execution processes of unmanned systems based on Reinforcement Learning (RL). Our methodology involved two pivotal experimental phases aimed at demystifying the rules governing these systems. The first phase involved sampling graphs from 20 varied scenarios in unmanned system environments, gathering ample sequences for graph generation, and assessing the performance and limitations of our algorithm. The second phase took place within a dedicated RL environment for unmanned systems, where we repeatedly trained policy trajectories to derive the environmental rules. These rules were then visualized, providing a clear and comprehensible representation of the strategies and operational procedures inherent to RL-based unmanned systems. This approach effectively demonstrates our method's capability to interpret and visually articulate the intricacies of policy and execution in unmanned systems driven by RL.

VI. ACKNOWLEDGMENT

REFERENCES

[1] Y. Shi, K. McAreavey, and W. Liu, "Evaluating contrastive explanations for ai planning with non-experts: a smart home battery scenario," in *2022 27th International Conference on Automation and Computing (ICAC)*. IEEE, 2022, pp. 1–6.

[2] Y.-P. Liu, L.-H. Zhao, Y.-W. Shi, S.-Y. Ren, and J.-L. Wang, "Finite-time passivity and synchronisation of complex networks with multiple output couplings," *International Journal of Control*, vol. 96, no. 6, pp. 1470–1490, 2023.

[3] L.-H. Zhao, J.-L. Wang, and Y. Zhang, "Lag output synchronization for multiple output coupled complex networks with positive semidefinite or positive definite output matrix," *Journal of the Franklin Institute*, vol. 357, no. 1, pp. 414–436, 2020.

[4] R. Li, J.-L. Wang, and Y.-W. Shi, "Passivity-based formation control for second-order multi-agent systems with linear or nonlinear coupling," *International Journal of Control*, vol. 96, no. 5, pp. 1190–1201, 2023.

[5] H. Ma, K. McAreavey, R. McConville, and W. Liu, "Explainable ai for non-experts: Energy tariff forecasting," in *2022 27th International Conference on Automation and Computing (ICAC)*. IEEE, 2022, pp. 1–6.

[6] L.-H. Zhao, S.-Y. Ren, Y.-W. Shi, and Y.-P. Liu, "Generalized lag output synchronization of the multiple output coupled complex dynamical networks," in *2019 Chinese Control Conference (CCC)*. IEEE, 2019, pp. 930–935.

[7] K. Kuru, D. Ansell, W. Khan, and H. Yetgin, "Analysis and optimization of unmanned aerial vehicle swarms in logistics: An intelligent delivery platform," *Ieee Access*, vol. 7, pp. 15 804–15 831, 2019.

[8] X. Zhou and J. Zhou, "Data-driven driving state control for unmanned agricultural logistics vehicle," *Ieee Access*, vol. 8, pp. 65 530–65 543, 2020.

[9] G. Cheng and J. Han, "A survey on object detection in optical remote sensing images," *ISPRS journal of photogrammetry and remote sensing*, vol. 117, pp. 11–28, 2016.

[10] L. Dong and J. Shan, "A comprehensive review of earthquake-induced building damage detection with remote sensing techniques," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 84, pp. 85–99, 2013.

[11] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, 2009, pp. 13–18.

[12] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*, 2010, pp. 1–10.

[13] T. Bewley and J. Lawry, "Tripletree: A versatile interpretable representation of black box agents and their environments," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 415–11 422.

[14] W. Guo, X. Wu, U. Khan, and X. Xing, "Edge: Explaining deep reinforcement learning policies," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 222–12 236, 2021.

[15] N. Bougie, T. Onishi, and Y. Tsuruoka, "Interpretable imitation learning with symbolic rewards," *ACM Transactions on Intelligent Systems and Technology*, 2023.

[16] A. Sieusahai and M. Guzdial, "Explaining deep reinforcement learning agents in the atari domain through a surrogate model," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 17, no. 1, 2021, pp. 82–90.

[17] C. Rupprecht, C. Ibrahim, and C. J. Pal, "Finding and visualizing weaknesses of deep reinforcement learning agents," *arXiv preprint arXiv:1904.01318*, 2019.

[18] M. L. Olson, R. Khanna, L. Neal, F. Li, and W.-K. Wong, "Counterfactual state explanations for reinforcement learning agents via generative deep learning," *Artificial Intelligence*, vol. 295, p. 103455, 2021.

[19] S. Sohn, H. Woo, J. Choi, and H. Lee, "Meta reinforcement learning with autonomous inference of subtask dependencies," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HkgsWxrtPB

[20] G. Liu, O. Schulte, W. Zhu, and Q. 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*. Springer, 2019, pp. 414–429.

[21] S. Sohn, J. Oh, and H. Lee, "Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies," in *Advances in Neural Information Processing Systems*, 2018, pp. 7156–7166.