Why? Why not? When? Visual Explanations of Agent Behaviour in Reinforcement Learning

Aditi Mishra* Utkarsh Soni† Jinbin Huang‡ Chris Bryan§

Arizona State University

ABSTRACT

Reinforcement learning (RL) is used in many domains, including autonomous driving, robotics, stock trading, and video games. Unfortunately, the black box nature of RL agents, combined with legal and ethical considerations, makes it increasingly important that humans (including those are who not experts in RL) understand the reasoning behind the actions taken by an RL agent, particularly in safety-critical domains. To help address this challenge, we introduce *PolicyExplainer*, a visual analytics interface which lets the user directly query an autonomous agent. PolicyExplainer visualizes the states, policy, and expected future rewards for an agent, and supports asking and answering questions such as: "*Why take this action? Why not take this other action? When is this action taken?*" PolicyExplainer is designed based upon a domain analysis with RL researchers, and is evaluated via qualitative and quantitative assessments on a trio of domains: taxi navigation, a stack bot domain, and drug recommendation for HIV patients.We find that PolicyExplainer's visual approach promotes trust and understanding of agent decisions better than a state-of-the-art text-based explanation approach. Interviews with domain practitioners provide further validation for PolicyExplainer as applied to safety-critical domains. Our results help demonstrate how visualization-based approaches can be leveraged to decode the behavior of autonomous RL agents, particularly for RL non-experts.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Treemaps; Human-centered computing—Visualization—Visualization design and evaluation methods

1 INTRODUCTION

Reinforcement learning (RL) has become a widely-used technique for training autonomous agents. The ability of RL agents to learn sophisticated decision-making in uncertain and complex environments has led to widespread application and success, including in video gaming, autonomous driving, robotics, healthcare, finance, smart grids, and education [20].

Unfortunately, as artificial intelligence (AI) and machine learning (ML) are increasingly deployed in safety-critical domains, there are emerging legal and ethical concerns due to the black box nature of models [2, 32]. For example, when a healthcare model recommends a drug treatment plan for a patient (e.g., [19]), can such recommendations be trusted? In human-robot collaborative environments, a human's reasoning for a decision might differ from a model's. Even when the same decision is reached, model recommendations are inherently untrustworthy without sufficient justification or explanation. This problem is well known in the AI/ML community [22], particularly since there exists no common language for a model to communicate decisions to the human and vice versa [11, 13].

Particularly for RL models that are being applied for decision-making in safety-critical domains, it is necessary that such agents are answerable to people who potentially have little or no AI/ML expertise. This motivates the current work, where we introduce *PolicyExplainer*, a novel visual analytics system for policy explanation to RL non-experts.

To our knowledge, PolicyExplainer represents *the first visual analytics system that supports the direct visual querying of and explanation from an RL agent to non-expert users*. PolicyExplainer supports users directly querying an RL agent via three of the most common RL policy questions [13, 21]: "*Why take this action? Why not take this other action? When is this action taken?*" PolicyExplainer is motivated based on a pre-study with AI/ML researchers, and is intended to be a first step for general-purpose interfaces for RL agent querying and explanation. In contrast to existing visual analytics interfaces for RL explanation [14, 17, 23, 30], PolicyExplainer is model-independent and supports both model-based and model-free algorithms. To evaluate PolicyExplainer, we conduct an empirical study with RL non-experts on three domains: the popular Taxi domain [9], an HIV drug recommendation domain [3], and a robot stacking boxes in an industrial environment (StackBot) domain. The results indicate that PolicyExplainer's visual explanation approach for agent question-and-answering is effective, particularly compared to text-based policy explanations created via a state-of-the-art natural language generation technique [13].

Succinctly, the contributions of this paper include the following. (1) We analyze design requirements for RL policy visualization and explanation for non-experts, based on a pre-study with RL researchers and reviewing recent AI/ML literature. (2) We define an explanation generation methodology for visual policy explanation in the form of *Why?*, *Why not?*, and *When?* questions. (3) We develop PolicyExplainer, a visual analytics interface that lets a user interactively query an RL agent and provides visual explanations of a policy. (4) Based on our experience in creating and extensively evaluating PolicyExplainer, we discuss how visualization-based explanations can increase user trust while lessening the cognitive effort required to understand the decision-making process of an RL agent, particularly for RL non-experts in safety-critical domains.

2 BACKGROUND ON REINFORCEMENT LEARNING

Reinforcement learning is a technique to train an autonomous **agent**, in which the agent interacts with the environment and learns to achieve some desired **goal** through trial-and-error. In contrast to supervised learning, the agent does not require a training set of labeled examples for the desired behavior; likewise, RL differs from unsupervised learning by not simply learning patterns from unlabeled data. Rather, the agent learns the desired behavior using its own experience interacting with the environment. An agent's overall goal can be defined in terms of a special signal called a **reward** that the agent gets for taking some action in the environment. Informally, the agent is tasked with learning a behavior that would maximize the total amount of reward it receives in the long term. Concretely, the problem of reinforcement learning can be formalized in terms of a **Markov decision process** (MDP).

*e-mail: amishr45@asu.edu
†e-mail:usoni1@asu.edu
‡e-mail:jhuan196@asu.edu
§e-mail:cbryan16@asu.edu

2022 IEEE 15th Pacific Visualization Symposium (PacificVis) | 978-1-6654-2335-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/PACIFICVIS53943.2022.00020

2165-8773/22/$31.00 ©2022 IEEE
DOI 10.1109/PacificVis53943.2022.00020

2.1 Markov Decision Process

MDP formulates a sequential decision-making task where, at each time step, the agent observes the current **state** of the environment and decides to take an **action**. This transitions the environment to its next state and the agent receives a reward. The agent keeps acting in the environment until it reaches a terminal state at some time step T. The reward the agent obtains for each step is discounted with a discount factor γ. The total reward obtained by the agent for the sequence of transitions is subsequently defined as the **return**, $G = \sum_{t=1}^{T} \gamma^{t-1} R_t$ where R_t is the reward obtained for the t^{th} transition.

Formally, a MDP is defined as a tuple $\langle \mathscr{S}, \mathscr{A}, \mathscr{T}, \mathscr{R}, \gamma \rangle$ where $\mathscr{S}$ is the set of all possible states of the environment, $\mathscr{A}$ is the set of all possible actions the agent can take at any state, $\mathscr{T}: \mathscr{S} \times \mathscr{A} \times \mathscr{S} \rightarrow [0,1]$ is the transition function where $\mathscr{T}(s,a,s')$ gives the probability that the environment will transition to a state s' when the agent takes an action a in state s, $\mathscr{R}: \mathscr{S} \times \mathscr{A} \times \mathscr{S} \rightarrow \mathbb{R}$ is the reward function that gives the reward the agent obtains when it takes an action a in state s causing a transition to state s', and γ is the discounting factor applied to the obtained rewards.

Given an MDP, a **policy** π maps each state to some action $a \in \mathscr{A}$. The **value** for any state is then the expected return the agent gets when it follows the policy π starting from the state. The goal of the agent is to learn a policy, referred to as the **optimal policy** π^*, that maximizes the value for each state. Lastly, the Q-value function, $Q_\pi(s,a)$ gives the value obtained obtained if the agent takes the action a in state s and then follow the policy π. If all the components of the MDP are known to the agent, then it can learn π^* as $\pi^*(s) = \text{argmax}_a Q^*(s,a)$, where the $Q^*(s,a)$ is obtained by solving the following Bellman optimality equation using dynamic programming algorithms like value iteration or policy iteration [28] : $Q^*(s,a) = \sum_{s'} \mathscr{T}(s,a,s')[\mathscr{R}(s,a,s') + \gamma . max_a Q^*(s',a)]$

2.2 Interpreting State Features

In this work, we assume each state s of the environment can be expressed as a feature vector $\langle f_1, f_2, \ldots, f_n \rangle$ where each feature f_i can be understood by the domain expert that will be using PolicyExplainer (this is in line with the policy explanation technique presented in [13], where states can be described in terms of binary features via classifiers). For example, for the HIV drug recommendation domain, states are defined via features like the number of infected lymphocytes, immune response, etc. PolicyExplainer provides explanations in terms of these features. For simplicity, we pick domains where the states were already defined in terms of features that a human can understand. Hence, the state features that the agent views during interaction with the environment would be the same as the one used for explanations. However, this is not a requirement for our system to work. See Section 9 for discussion about relaxing this requirement in future work.

2.3 Model-Based and Model-Free Learning

RL algorithms can be classified as either **model-based** and **model-free** [28]. In model-based RL, the agent learns the model of the environment (specifically, it learns the model components of the MDP incorporating the environment dynamics) and uses that to derive the optimal policy. Conversely, model-free RL algorithms do not require a learned model to obtain the optimal policy.

In this work, we use algorithms from both the classes to train our agent for different domains. For the model-based approach, we assume access to the state set $\mathscr{S}$, and employ a simple sampling strategy to learn the model parameters $\mathscr{T}$ and $\mathscr{R}$. With this strategy, for each state $s \in S$, each action $a \in A$ is executed k times. The value of $\mathscr{T}(s,a,s')$ is then set to k'/k where k' is the number of times the environment transitions to state s' when the agent took the action a in state s. The value of $\mathscr{R}(s,a,s')$ is set as the average reward obtained for the transitions $\langle s,a,s' \rangle$. Once the model is learned, we compute the optimal policy using policy iteration.

For the model-free approaches, we use function approximation based methods that learn the Q-function directly from agent's experience. The learning involves the agent interacting with the environment over several episodes improving its Q-function estimate. In each episode, the agent starts at some random initial state and follows an ε-greedy policy in which the agent chooses an action that maximizes its Q value with a probability of $1-\varepsilon$ or chooses to do a random action (uniformly sampled) with probability ε. The agent collects experience using the ε-greedy policy and then uses it to approximate the Q-function. For our domains, we used linear function and neural network based approximations. The former technique represents the Q-function as a weighted linear function of features f_i defined over the state: $Q_\theta(s,a) = \theta_1 . f_1(s) + \theta_2 . f_2(s) + \cdots + \theta_n . f_n(s)$ The weights are updated after each individual interaction with the environment, where the agent takes an action a in state s resulting in it transitioning to state s' getting a reward r, using the following update equation where α is the learning rate: $\theta_i = \theta_i + \alpha * [r + \gamma . max_a Q_\theta(s',a') - Q_\theta(s,a)] f_i(s)$

For the neural network based Q-function approximation, we trained a fully connected neural network using the same strategy as the one used to train deep Q-networks (DQN) in [25]. The neural network given by $Q(s,a,\theta)$, where θ represents the weights of the network, approximates the Q-value function corresponding to the optimal policy i.e. $Q^*(s,a)$. The agent's transition at each time step, $\langle s,a,r,s' \rangle$ is stored in a database D. The database is of a fixed length and stores the most recent transitions. After each action is executed (which is treated as an iteration i of the training algorithm), the Q-network is trained by optimizing the following loss function over a mini batch of transitions sampled uniformly from D: $L_i(\theta_i) = E_{(s,a,r,s') \sim D}[(r + \gamma max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))^2]$.

In PolicyExplainer the Taxi Domain was trained using Model Based RL, StackBot using Approximate Q-learning and the HIV Domain using a DQN.

3 Related Work

3.1 Explainability in Reinforcement Learning

As the use of RL continues to expand, there is an increasing interest in XAI as it applies to RL. In our work, the RL agent is considered as a black box and PolicyExplainer computes explanations for its decisions. Our explanation technique is inspired from LIME [27], which attempts to explain a classifier's decisions. In contrast to LIME, which works on one-shot decision making problems, we investigate if a similar technique can explain the decisions of an RL agent which solves a sequential decision-making problem. In addition, we provide functionalities in our interface that not only answer queries about why the agent took a particular action, but also answer contrastive queries about why an agent chose a particular action over an alternative action suggested by the user, and when, in general, does an agent takes a particular action.

There have been other works specific to explaining RL agent's policy, primarily focusing on pixel-based domains. For example, Greydanus et al. [12] utilized saliency maps to gain insights on how an agent learns and executes a policy in a 2D video game space. Similarly, Yang et al. [34] identified regions of interest by visualizing pixels of game images. Hayes et al. [13] proposed a set of algorithms to explain agent policies using a common modality of Natural Language (i.e text). Recent research [29] has explored policy explanation via answering contrastive queries, where an example question might look like: "*Why did the agent go right instead of going left?*" Here, two main entities—a fact and a foil—are contrasted to explain why the fact was chosen over the foil. PolicyExplainer visualizes contrastive explanations (the "*Why not?*" question) among the other types of questions. (Section 8).

3.2 Visualization for XAI and RL Explainability

For general discussion on the use of visualization for deep learning and XAI, several recent surveys are available [7, 15, 37]. Here, we focus on describing recent visualization tools specifically for RL analysis and explainability, which include the following:

MDPVis [23] is a system designed for debugging and optimizing MDPs by interacting with an MDP simulator. DQNViz [30] focuses on analyzing the training of a deep RL agent, from a high-level overview down to individual epochs. DRLViz [17] visualizes the internal memory of a deep RL agent as a way to interpret its decisions. Similarly, DynamicsExplorer [14] is a diagonistic tool for looking into the learnt policy under different dynamic settings. Each of the aforementioned systems have significant differences compared to PolicyExplainer. First, the primary objective of each tool is to support *internal* debugging of a trained model, with a focus on *RL expert users*. In contrast, PolicyExplainer supports *RL non-experts* by promoting the human's understanding of the RL agent's exhibited *external* behavior. Second, these systems are highly domain or technique-dependent, which limits their generalizability. For instance, DQNViz [30] provides a trajectory view that only supports pixel-based (specifically, Atari) video games. DRLViz [17] only supports models trained using RNNs. In contrast, PolicyExplainer supports both model-based and model-free RL. We additionally demonstrate PolicyExplainer across three significantly different types of domains (including a safety-critical HIV domain).

In actuality, PolicyExplainer can support *any* domain provided the states can be represented in human-interpretable features. For complex domains such as Atari games where states are represented as a collection of pixels (i.e., an image), it is currently a significant open problem in the AI community to learn an interpretable representation of the pixel features that can reasonably capture the domain's dynamics. To this end, tools like DRLViz are meant for RL expert users; they sidestep the issue by showing the state directly. In Section 9, we provide thoughts on how to approach these types of problems via concept-based "state abstractions," but as this is still an open AI problem, we omit pixel-based domains in the current paper.

4 Pre-Study and Design Requirements

To motivate a design for question-based visual explanation of agent decisions, we conducted a pre-study with three RL experts. Each had at least four years of research experience in AI/RL explanation for non-experts. This pre-study consisted of extended email correspondences and completing a survey, all aimed at understanding the role of explanation and interaction as it relates to RL agents for non-experts. Additionally, we reviewed recent papers that discuss issues of RL interpretability and transparency, which therefore provide motivation for agent explanations (e.g., [4, 8]). Based on the collected feedback and paper readings, we identified a set of four high-level design requirements **DR1**–**DR4**.

DR1: Provide an overview of the state space and policy in terms of its diversity and expected future rewards. RL agents might learn on domains with large state spaces whose dynamics can be modeled as networks. As these networks scale in complexity, it quickly becomes difficult for human users to understand them [35]. Multiple pre-study participants noted that the ability to navigate and explore the state space (and the actions taken in those states) is necessary for understanding the policy of an RL agent. Further, being able to show the states and the expected future rewards (given an optimal policy) helps users identify other states that have either *highly different* or *highly similar* rewards. *Visualization can provide an overview of the policy, with an emphasis on highlighting states with similar/different expected rewards.*

DR2: Provide visualizations for individual states. Pre-study participants discussed the importance of being able to inspect and review individual states. To understand a state, it must be represented or defined in a way that provides semantic meaning. When state features correspond to a spatiophysical domain, a straightforward solution is simply showing an image of the state (e.g., DQNViz visualizes the pixels displayed in Atari video games). However, this solution does not work if states do not have a physical domain. Consider a healthcare agent for recommending a patient's treatment plan, where state features consist of abstract health metrics. As opposed to simply providing a tabular representation of the states and values, *visualization can support analysis and comparison across the potentially thousands of states that make up a policy.*

DR3: Let the user ask questions to the agent. While DR1 and DR2 are important to provide generalized information about the agent's policy and the individual states in the domain, they do not provide explanations or justifications for the agent's decisions. Ultimately, when a user is examining the actions taken by the agent, they will focus on questions like, "*Why* was this action taken? *Why not* take this other action? *When* is this action taken?" *To explain the agent's decision-making process, visualizations (and interactions) should be designed to support a question-and-answering dialogue between the humans and the RL agent.*

DR4: Allow users to navigate the explanation space to prevent overloading. Finally, the explanation given by the RL agent highly depends on the state features. There might be multiple conditions in which a certain action is taken. However, giving the user all the reasons for an agent's decision (i.e., identifying every condition) might prove overwhelming. Instead, being able to identify important state regions with similar explanations, and *letting user interactively choose the explanation they wish to see, can limit cognitive overhead and help in better understanding the agent's reasoning.*

5 Policy Explanation

Lim et al. [21] found that *Why?* and the *Why not?* are the types of questions most commonly asked to intelligent systems. Relatedly, some of the most cited papers on policy explainability (e.g., [13]) highlight identifying state regions (i.e., *When?* questions) as an important task. We thus focus on generating visual explanations for these three question types. These explanations are used by PolicyExplainer to support interactive question-and-answer sessions between the user and RL agent. Informally, explanations are based on the idea of highlighting the state features that lead to an action being chosen as the optimal action. This approach has previously been used for explaining classification decisions [10, 26]; in our case, we apply this idea to explain optimal policies learned for a sequential decision making task.

To learn the salient features that effect the agent's policy, we first approximate the policy via supervised learning which learns a decision boundary based on the features that separate classes. We then extract how the classification algorithm uses these features to determine the output class. This means the algorithm must itself be interpretable. Because of this, we use a decision tree classifier to approximate the policy. A significant advantage of decision trees is that it is quite easy to track which features (and their ranges of values) lead to particular classification results.

As explained in Section 2, each state is defined as a feature vector $\langle f_1, f_2, \ldots, f_n \rangle$ and the policy π^* maps each state to its corresponding optimal action. We start by using this complete mapping as a set of training samples, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_{|\mathscr{S}|}, y_{|\mathscr{S}|})\}$, to train a decision tree T, where x_i represents the state features, and y_i is the optimal action. The decision tree is a binary tree where each non-leaf node n has some feature f and a corresponding threshold value θ associated with it. The edge e that connects the node to its left child represents the condition $f < \theta$ while the edge for the right child represents the condition $f \geq \theta$. We define the direction of an edge as *left* / *right* if it connects to the left / right child of the node.

Any input state to the decision tree can be mapped to a unique path in T from the root node to a leaf node by following the edges that the features of the state satisfy. We denote this unique path cor-

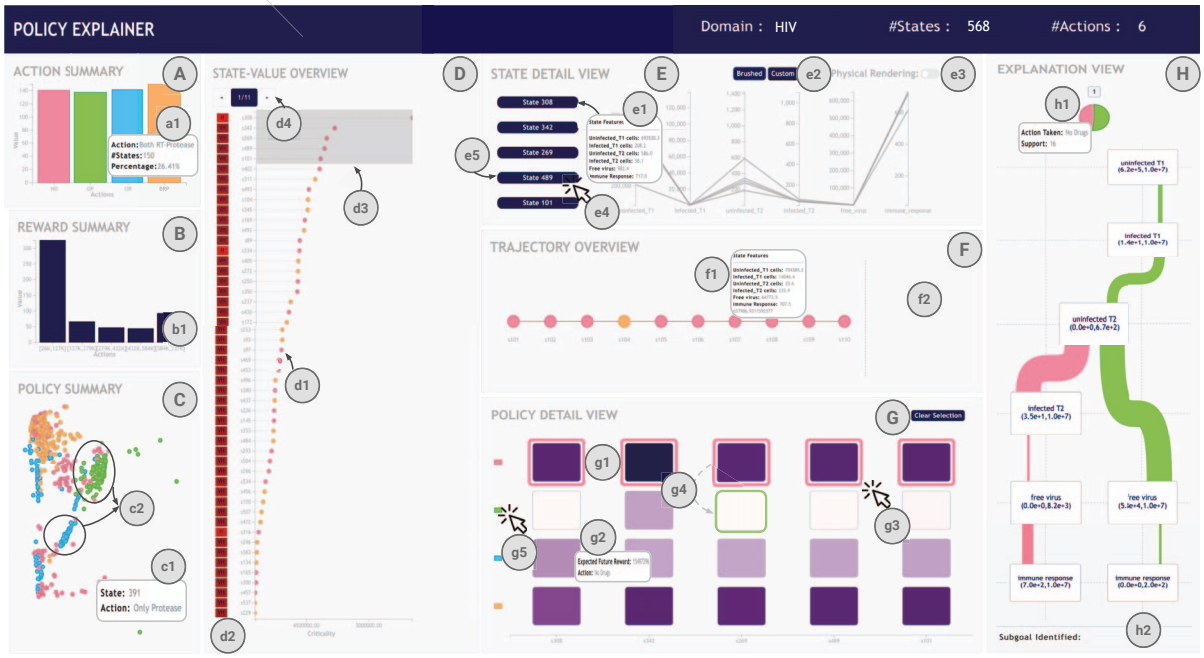

Figure 1: The PolicyExplainer interface, shown here with an HIV treatment domain [3, 24], consists of eight main linked sections, which support (A–C) summarizing the domain and the optimal policy, (D) summarizing state regions and expected rewards, (E–G) detailed analysis of states and policy, (F) a trajectory to see the agent progression, and (H) an explanation panel to answer *Why? Why not? When?* questions.

responding to a state s as $\mathscr{P}(s)$. The leaf node would be associated with an action a that should ideally be $\pi^*(s)$. Any path from the root to a leaf node in T represents a decision rule of the form "*if* condition$_1$ *and* condition$_2$ ... condition$_K$ *then* action" where each condition corresponds to a unique feature and it gives the range of values that feature can take for any input state to be classified as the action, while K is the total number of unique features on the path. We denote the rule corresponding to a path $\mathscr{P}$ as rule$(\mathscr{P})$. Given a path $\mathscr{P}$, the associated rule can be identified by Algorithm 1. PolicyExplainer answers user queries in terms of these rules as explained in the remaining parts of this section.

An important note here is, for the domains considered in this paper, the decision tree for each case is overfitted to generate rules. This is allowed because, as we have access to the entire state space and the policy, we do not need to perform any form of testing or validation, and can focus on achieving the highest training accuracy. This can also be done if only a partial state space exists along with its corresponding optimal action. In each case, the fidelity of the decision tree is $>= 99\%$, which means the explanations generated by the agent is highly accurate.

Answering *Why?* Questions. These questions take the form, "*Why would you take* $\{action\ a\}$ *in* $\{state\ s\}$*?*" To answer this query, we identify the specific conditions under which the action a is executed by the agent. This is achieved by identifying the path $\mathscr{P}(s)$ corresponding to the state s and then applying the procedure in Algorithm 1 to compute rule$(\mathscr{P}(s))$. The computed decision rule serves as the explanation.

Answering *Why not?* Questions. These are contrastive queries [29] that take the form, "*Why would you take* $\{action\ a^*\}$ *instead of* $\{action\ a_f\}$ *in state s?*" Here, a^* is the action chosen by the policy and a_f is the alternate action that the user might prefer to take. To answer these type of queries, we find a state s_f that is closest to the state s where the agent would execute action a_f. The distance between any two states is calculated as the Euclidean distance between their corresponding feature vectors. We then compute two decision rules, rule$(\mathscr{P}(s))$ and rule$(\mathscr{P}(s_f))$, that show conditions when action a^* and a_f are chosen. These rules are presented to the user as a response to their query. In other words, by comparing the results of the rules, the user can assess why the policy picks a^* over a_f. This is similar to [13] where contrastive queries are answered by comparing features of similar states under which the two actions are taken except that they use natural language for explanation.

Algorithm 1 Compute rule$(\mathscr{P})$

Input: path $\mathscr{P}$, set of all features in the domain $\mathscr{F}$, decision tree T
Output: Rule associated with $\mathscr{P}$

relevant_features $\leftarrow \{\}$
for $f \in \mathscr{F}$ **do**
  $f_{\text{range}} \leftarrow$ set of all possible values for f
end for
current_node $\leftarrow$ root node of $\mathscr{P}$
while current_node is not leaf **do**
  current_edge $\leftarrow$ edge connected to current_node in $\mathscr{P}$
  $C(f) \leftarrow$ feature associated with the current_node
  $C(\theta) \leftarrow$ threshold associated with the current_node
  relevant_features $\leftarrow$ relevant_features $\cup\, C(f)$
  if direction of current_edge is *right* **then**
    current_node_range $\leftarrow [C(\theta), \infty]$
  else
    current_node_range $\leftarrow [-\infty, C(\theta)]$
  end if
  $C(f)_{range} \leftarrow C(f)_{range} \cap$ current_node_range
  current_node $\leftarrow$ node adjacent to current_node in $\mathscr{P}$
end while
return $f_{\text{range}}\ \forall f \in$ relevant_features

Answering *When?* Questions. The final question type we support is of the form, "*When would you do take the* $\{action\ a\}$*?*" One common motivation for this question type is for the user to understand the most frequent cases where this action is chosen. To operationalize this, we first identify all the leaf nodes in T whose label is a. For each leaf node, we find the number of states $s \in \mathscr{S}$ whose path $\mathscr{P}(s)$ ends in that node. We pick the top three nodes with the highest number of states, and for each of these nodes, compute the path $\mathscr{P}$ from the root to that node. Finally, we compute rule$(\mathscr{P})$

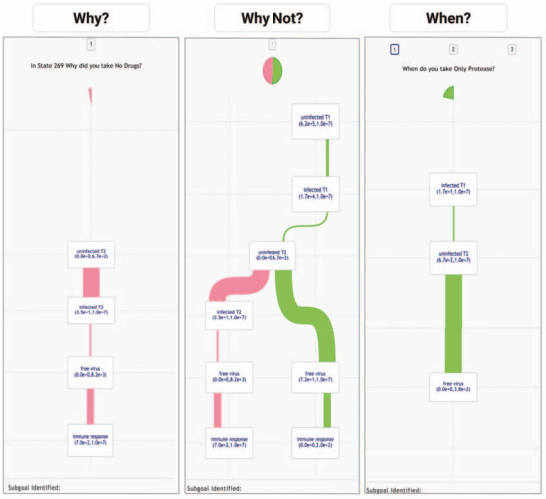

Figure 2: PolicyExplainer can answer three types of questions in its explanation panel. (*Why?*) Here, the user is asking why the No Drugs action was taken (action colors map to the key in Figure 1(a1)); there are four relevant features used in this decision. (*Why not?*) Here, the contrastive question being answered is, "*Why take the (pink) No Drugs action instead of the (green) Only Protease action?*" While one feature is shared between both actions, the rest influence the actions based on their values. (*When?*) The user is asking when the Only Protease action is taken. The system gives top three most general reasons for this action being the optimal one. Currently, the first explanation is toggled, which shows the influence of three features.

for each path and use that as the response to the query.

6 Visual Analytics System : PolicyExplainer

PolicyExplainer requires an agent trained using either model-based or a model-free approach. We extract data from the trained agent—not just the optimal policy, but also states, features, the Q-values and rewards for each state-action pair. If the state space is continuous, we find the most important states based on importance function defined in [4]. This extracted data is fed into the PolicyExplainer interface (shown in Figure 1), which consists of eight linked panels that provide general visual analytics about the policy and states (**DR1**–**DR2**) and lets users interactively ask *Why? Why not? When?* questions to the RL agent and receive visual explanations (**DR3**–**DR4**).

(A, B) Action and Reward Summary Panel. The action distribution and the reward distribution bar charts show summary statistics, respectively, of the frequency of actions and rewards which the agent takes or receives in its optimal policy π^*. (a1, b1) Hovering on the charts displays a tooltip showing exact values.

(C) Policy Summary Panel. The policy summary panel shows an overview of the optimal policy and is laid out via dimensionality reduction based on state features. Each state is encoded as a circle and the color encodes the optimal action. (c1) Hovering on the circle displays a tooltip with the state and the action chosen in the particular state. This panel along with the State value overview panel (D) is meant to support extracting interesting states for the user to explore. As seen in (c2), the panel highlights some clusters wherein states with similar features have the same recommended action.

(D) State & Value Overview. The state value overview provides a summary of the state values over the state space (**DR1**). A horizontal lollipop chart (d1) showcases a set of 50 states arranged in a descending order of criticality. The vertical y-axis consists of individual states and the horizontal x-axis represents the critical values. A state is defined to be more critical if there is a significant difference of rewards on randomly choosing an action and the reward gained on doing an optimal action [16]. Mathematically, this is represented with the following equation: $C(s) = max_a Q^{\pi^*}_{(s,a)} - \frac{1}{N_a}\sum_a Q^{\pi^*}_{(s,a)}$ where $C(s)$ is the criticality C of state s, π^* is the optimal policy, and $Q_{(s,a)}$ is the expected future reward in state s on taking action a. The red boxes beside each state (d2) represent the value of the state with labels that ranges from Very High to Very Low values which are also redundantly encoded using a sequential color scale, with darker red showing higher rewards and light orange shade showcasing lower rewards. A fixed-width brush (d3) can be scrubbed across the state space; this selects a set of states for further analysis in the state and the policy detail panel. For exploring and navigating large state spaces, users can scroll or tab across a page navigator tool (d4).

(E) States Detail View. The states detail view shows a detailed visualization of states (**DR2**). (e5) Selected states are shown in a blue colored panel. Each state is visualized as a line in a parallel coordinate chart, where each y-axis represents features of the state. (e1) Hovering on (e5) shows the exact features values for that state. (e4) Clicking on a state populates the trajectory panel (F). (e3) For states that can be spatiophysically represented, a toggle can switch to this view. Apart from selecting brushed states, the user can also customize the states they wish to see by (e2) clicking on the customize button and entering their desired states manually.

(F) Trajectory View. This panel summarizes the "trajectory" of the agent starting from a selected state to an end state (either the policy goal or a user-defined goal). When a state is selected (via clicking on (e5)), this panel loads a simple visualization with each state represented as a circle and the action optimally chosen by the agent is encoded as the color. Each consecutive state is linked by a straight line running between them. (f1) Hovering on the circle shows a tooltip with feature names, its corresponding values, and the reward gained by the agent upon taking the action. (f2) The space beside the visualization shows the animation of the trajectory the agent takes to reach the goal if the spatiophysical rendering exists (see the demo video in supplemental materials for an example. In Figure 1 we skip it, since it showcases the HIV domain which lacks a physical rendering.).

(G) Policy Detail View. When states are loaded in the states detail view, the policy detail is also populated. The available actions for each state are represented by a set of rectangular swatches (in the figure, each state has four available actions) colored by their Q-values if the agent takes that action. As the number of states increases, they become arranged into a 2D grid.

(g1) For each state, the swatch corresponding to the state's optimal action is given a colored border. The border color of this swatch corresponds to the type of action taken, using the color key from the action distribution chart in (A). (g2) Hovering on a swatch provides details about the action and its expected reward i.e it's Q value.

To support policy explanation, this chart contains three interactions which let the user ask "*Why?, Why not? When?*" questions to the agent (**DR3**). (g3) First, clicking on the optimal action for a state asks, "*Why was this action chosen?*" (g4) Similarly, dragging from the optimal action to another action for a state asks a contrastive question: "*Why not take this other action instead of the optimal action?*" (This interaction adds a stroke on the contrastive action swatch.) (g5) Finally, clicking the action icons on the left side of the chart asks, "*When is this action taken?*" Each of these questions loads the explanation panel (H) with the respective explanation.

(H) Explanation Panel. Finally, the explanation panel provides visual explanations for "*Why? Why not? When?*" questions asked in the policy detail view (G). Our approach for visual explanation is inspired from Young and Shneiderman's work visualizing boolean queries with flowcharts, where AND operations are represented as conditions on the same path, and OR conditions are shown with a forking path [36]. For PolicyExplainer, horizontal dotted lines map to state features, which are labeled in boxes along with their corresponding possible feature values appended to the lines. A flow

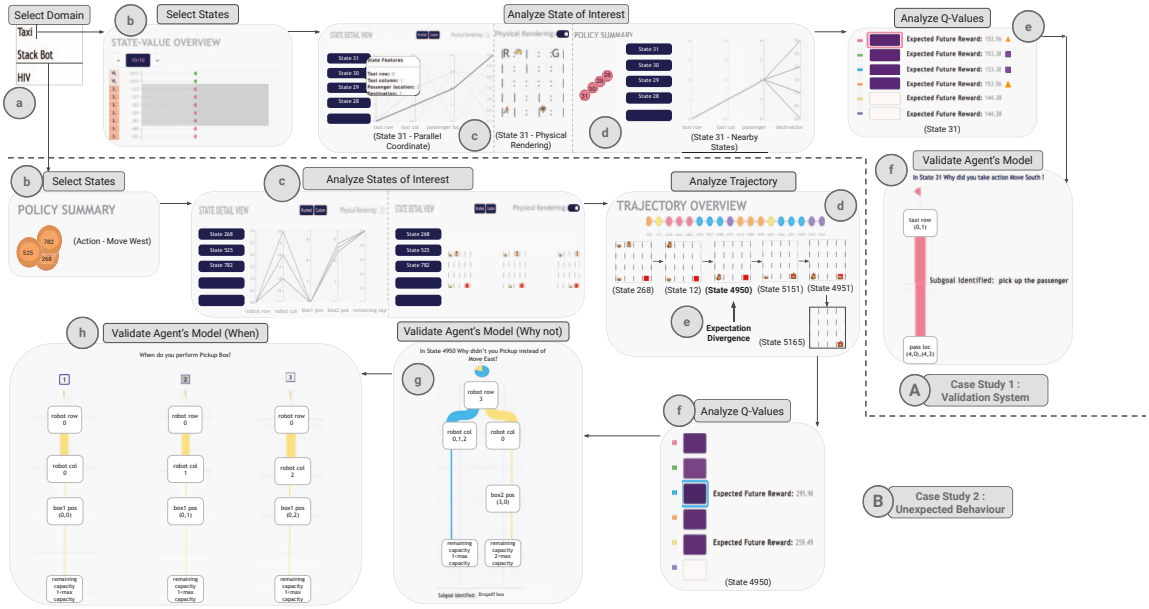

Figure 3: Gary's actions taken during the (A) case study #1 and (B) case study #2.

that links across several features (e.g., the green flow in Figure 4(H) links across five features) represents an action taken based on the conjunction of several features, akin an AND operation. Links are colored by their action type using the action key in (A); link thickness indicates the number of states that satisfy intermediate conditions. The order of the features matches the feature ordering from other panels to facilitate comparison of explanations. In contrast to Young and Shneiderman's boolean queries, policy explanation does not include the concept of OR operations. Therefore, forking indicates contrastive actions during "*Why not?*" questions, where two actions might share some features but split on others. The use of differently colored links to differentiate the actions helps to demonstrate this. Figure 2 shows example explanations for each question type.

(h1) At the top of this panel, a pie chart shows the number of states for which the explanation holds; in other words it denotes the coverage of the explanation. A hover tooltip provides the exact counts and uses linked highlighting to show these states in (C). (h2) For each explanation in a state, the subgoal being satisfied by the agent in the particular state is also identified if present.

7 Case Studies

To illustrate how PolicyExplainer can be used to explore a policy and question an RL agent, we present two use case scenarios using the Taxi and the StackBot domains.

7.1 Use Case 1: Reassuring Users of Agent Behaviour.

Taxi Domain. The taxi domain [9] consists of a 5×5 grid (500 states with 6 actions and 4 features) with walls separating some cells. The agent is represented as a taxi (🚕) which can move around the grid except through the walls. The grid contains four special locations: R, G, Y and B. The passenger (🧍) and the destination (📍) will be present at any four of these locations. The agent's task is to pick up the passenger and drop them at the destination. Every step the agent takes entails a reward of -1; on completing the task, the agent receives a reward of +20.

While relatively straightforward, the Taxi domain is an important and widely-used domain for demonstrating RL techniques including explainability. It is thus a good candidate domain for demonstrating PolicyExplainer's explanation methodology and user experience.

Gary's Analysis. Gary, a human who wants to validate his understanding of the agent behaviour and make sure that the agent's understanding of the environment matches with his understanding of what actions will the agent perform. He uses PolicyExplainer to query the agent; his specific actions are shown in Figure 3(A).

(a) Gary first loads the Taxi domain from the domain dropdown. This populates the action, reward summary panel and the states and values overview panel. (b) He uses the state-value overview to identify states of interest. He notices and selects a set of states with low expected cumulative future rewards, but with all states with the same optimal action (States 13-82). (c) Selecting these loads them into the state detail view. He finds State 31 to be particularly interesting, as both the passenger location and the destination are below the agent's position. Gary wants to validate his understanding of the agent behaviour by making sure the agent first performs the Move South action to pick up the passenger.

(d) Gary hovers over State 31 in the state detail view and realizes that nearby states are have only a single feature (Destination) changing, such as States 28, 29 and 30. These states have a similar feature: the agent is north of the passenger and the optimal action in these states is Move South. (e) Gary next looks into the policy detail view. Hovering over the tiles in the policy detail view shows an expected future reward. Gary notices that the expected reward for moving south is higher than moving north or east, but only barely. He realizes that taking moving north or east in State 31 would require agent to return back to State 31; thus, though these actions have high rewards, they are lower than the optimal Move South action. This reassures Gary that the agent will perform according to his expectations

Gary then clicks on the background rectangle in state 31 to ask a *Why?* question: "*Why did the agent perform the Move South action in State 31?*" (f) The explanation panel shows that the agent does indeed consider both its own position and the passenger's position when making decisions. The identified subgoal additionally tells Gary that the agent is performing the Move South action in State 31 to pick up the passenger.

7.2 Use Case 2: Decoding Unexpected Agent Behaviour.

StackBot Domain. The StackBot domain consists of a 4×4 grid containing 2 boxes (10,368 states with 6 actions and 5 features). The agent is represented as a robot (🤖) which can freely move around

the grid to pick up two boxes ( ). The robot's task is to drop these boxes at the goal location ( ). The robot has a capacity of two boxes that it can pick up and hold at a time. However, the agent has been trained to be cautious, its policy is that it will only pick up and hold one box at a time. This counter intuitive behavior is not known to the user. Every step the robot takes gives it a reward of -1, picking up a box gives +20, and successfully dropping off a box gives +350. The episode ends when all boxes have been dropped off at the goal location, which gives a final high reward of +500.

Gary's Analysis. Gary again wants to understand the RL agent's behavior. In contrast to Case Study #1, the StackBot agent behaves in an unexpected manner (only holding 1 box at a time, despite having a larger capacity). Gary must resolve his gap of understanding between how he thinks the agent should act (holding 2 boxes at a time) and the agent's actual behavior. His actions are shown in Figure 3(B); this use case is also shown in the demo video found in the supplemental materials.

Similar to the previous use case, (a) Gary loads the Stack Bot domain and reviews the action and reward summary panels, policy summary panel and the state and value overview panels.

(b) Reviewing these, he notices that some states place have the same optimal action of Move West. (c) These states have similar features: the robot column and the second box's position changing, but the robot row, first box's position and the remaining capacity remains the same. (d) Gary reviews the trajectory of State 268 (i.e., the actions taken from here to finish the task) and realizes that the agent could have picked up a second box but did not (i.e., it was at that box's location and had capacity). Instead, it continued moving to the goal location to drop off the single box it was carrying. (e) Gary identifies the state where the unexpected action of Going East happens instead of the expected action Pickup Box as State 4950. (f) He loads that state into the policy detail view to analyze its Q-values. The Q-value boxes reveal a minor dip in the expected future reward if the agent chose to pickup the box, which is counterintuitive to Gary's mental model of how the agent is supposed to work.

To gain insight into this unexpected behaviour, Gary asks a contrasive question to the agent: "*Why did the agent perform action Move East instead of the action Pickup Box in State 4950?*" (g) The explanation informs Gary that the agent moves east whenever any one of the box positions is to the left of the goal position (located at box (3,3)) and the remaining capacity is 1 (i.e., the robot is holding a box). On seeing this contrastive explanation, Gary notices that the robot picks up boxes when it is at the same location and its capacity is 2 (i.e., it is not holding any boxes). Gary wants to see if this behavior is always true regardless of the location, so he asks a *When?* question to the agent: "*When does the agent perform action Pickup Box?*" (h) The explanation shows that a pickup happens only when the robot is not holding a box (specifically, when its remaining capacity is equal to its maximum capacity). Gary now understands why the agent dropped of the first box rather than picking up the second box.

8 Evaluation

To empirically evaluate PolicyExplainer, we conducted two studies: a controlled usability study with ten graduate computer science students (p1–p10), and extensive usability reviews with three domain experts who research HIV and vaccinology (e1–e3). Notably, *none of our participants were experts in RL.* These evaluations serve two purposes: (1) To understand how PolicyExplainer's visual representations and question-and-answer dialogues support understanding of a learnt policy by non-experts. (2) To compare PolicyExplainer's visual explanation approach against a state-of-the-art text-based explanation baseline.

8.1 Study #1 Design and Setup

Baseline. As a baseline for comparing against PolicyExplainer's visual explanations, we utilize the text-based policy explanation technique from [13]. Despite being a relatively recent publication, this is one of the most cited papers for policy explanation via natural language generation, and is still considered a state-of-the-art approach in the community. This technique supports "*Why?*" and "*Why not?*" questions as well as understanding situational behaviour, making it analogous to our three question types supported by PolicyExplainer. Explanations are based on the software's control logic and a user query; the output consists of all possible conditions for the asked question. Unfortunately, like most text-based explanation approaches, our assumption is that even for simple domains (like Taxi) this approach can quickly lead to long run-on sentences that are cumbersome for humans to parse through (see **DR4**). One motivation for PolicyExplainer is that visualization can potentially both improve interpretation and alleviate cognitive load by representing explanations in easy-to-understand and interactive visual encodings.

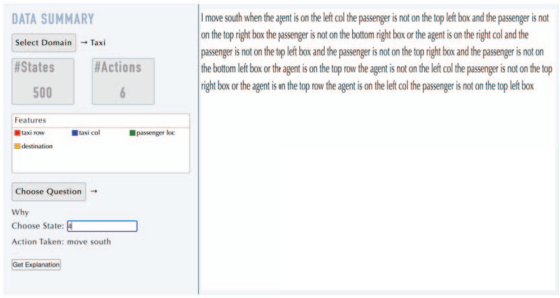

Figure 4: The baseline interface created for the user studies, which employs text-based explanations from [13].

We downloaded the GitLab code for this technique [1] and tweaked the code to obtain answers for the *Why?*, *Why not?* and *When?* question types discussed in this paper. We then created a simple frontend interface to support the user interactively querying an RL agent, shown in Figure 4.

Domains. Three domains were used in Study #1: the Taxi and StackBot domains described in Section 7, and a safety-critical HIV drug treatment domain. The HIV domain [3] consists of an RL agent recommending a drug cocktail for a patient. States in this domain (568 total) consist of six features: (uninfected CD4+ T-lymphocytes, infected CD4+ T-lymphocytes, uninfected macrophagus, infected macrophagus, free virus, immune response). Four actions are available to the agent: (No drugs, Only Protease, Only RT, Both Protease and RT). Based on the consequences of an action in a given state (either positive or negative), the agent is given a positive or a negative reward.

Design. The study design consisted of five stages:

(1) Interface Assignment and Training Stage. First, the participant was assigned one of the interfaces. A hands on training was given, explaining available system features and interactions. Participants could ask questions and play around with the interface until they felt comfortable enough to proceed.

(2) Task Stage. For this stage, participants were shown explanations provided by the assigned interface for the three types of supported questions: *Why?*, *Why not?*, and *When?*. Participants were tasked to rate the explanations for the question type, which we refer to as tasks t1, t2, and t3, respectively. In PolicyExplainer the users could use visualizations from other panels to aid their understanding of the explanation generated.

Participants completed three trials for each task, or nine total trials. Tasks were timed; participants were told to notify the administrator if or when they thought they understood the agent's explanation for its decision. For each task, we had pre-selected six states from the Taxi domain; three were chosen for the participant's assigned interface and the others were held out. At the completion of the nine trials, participants completed a short survey rating the understandability of the explanations based on a 7-point Likert scale.

(3, 4) Training and Task Stages with the other Interface. After completing the task stage with an initially-assigned interface, participants repeated the training and task stages with the other interface. Trials in the second iteration of the task utilized the states that were held out in the first iteration. To minimize potential confounds, the order of interface assignments, the selection of states for each interface, and the trial ordering was counterbalanced among participants.

(5) Freeform Analysis Stage. Finally, participants opened PolicyExplainer and could freely explore the three domains. No specific task was assigned in this stage, but participants were encouraged to put themselves into the following scenario: They are a supervisor in a company with autonomous agents employed and were told to report back any unexpected situations that occur, with their understanding of the agent's reasoning. In this stage, we wanted to understand how PolicyExplainer's features and overall user experience support RL interpretability, so the baseline interface was not used. Participants had ten minutes to complete this stage, and utilized think aloud protocol to verbalize their cognitive processes. At the end of the stage, participants completed a short usability survey and, if desired, could provide additional commentary about PolicyExplainer and baseline.

Participants and Apparatus. Ten graduate computer science students were recruited from Arizona State University (average age = 24.6, SD = 1.42; 7 males, 3 females). Although some of the graduate students were familiar with AI/ML, all reported little-to-no experience in RL. Each session lasted between 45–60 minutes.

During study sessions, both interfaces were shown in Google Chrome in full screen mode at 3840×2160 resolution. Sessions were held in a quiet, office-like environment with no distractions.

8.2 Study #1 Results

Where applicable, we report Mann-Whitney U tests to indicate if there is a statistical difference in explanation understandability between PolicyExplainer and baseline (using a threshold of $p = 0.05$) by providing U and p values.

8.2.1 Task Stage Performance

Two types of data points were measured during task stage trials: the time taken to complete each trial, and ratings from the participant about the understandability of explanations from each interface.

For PolicyExplainer, the average completion time in seconds for each task was t1= 39.7, t2= 58.5, and t3= 56.2. Interestingly, there were several baseline trials where participants gave up halfway through trying to understand the explanation (for all three question types), with the justification that the explanations were too verbose and difficult to understand. Participant *e10* succinctly gave a reason for this: "*The text explanations were hard to understand, I felt like giving up while doing the tasks since it was too much of mental effort.*" Thus, while the text explanation trials took more time on average (t1= 97.8, t2= 86.9, t3= 55.8), this data is skewed it does not include the abandoned trials. That said, for completed trials, participants described their interpretation of the generated explanation to the administrator. We found that, for all explanations, participant interpretations of generated explanations were correct.

Three questions were asked to participant about the understandability of the explanations for the agent behavior, shown in Figure 5(Qn1–Qn3). For each question, PolicyExplainer performed significantly better in terms of understanding ($U = 0, p < 0.005$), gaining trust ($U = 2, p < 0.005$) and the cognitive effort required to understand presented explanations ($U = 0, p < 0.005$). These results indicate that participants felt PolicyExplainer's visual explanations were much easier to understand compared to the baseline's state-of-the-art text explanations.

8.2.2 Freeform Stage: User Comments and Survey Ratings

Here, we report comments and feedback collected during and after the freeform analysis stage. Figure 5(Qn4–Qn13) shows participant survey feedback about using the system during this stage. PolicyExplainer's functionality and interface features were highly rated by almost all participants. Since the baseline was not used in the freeform stage, it does not have corresponding ratings for these questions, though several participants compared the two interfaces during and after this stage.

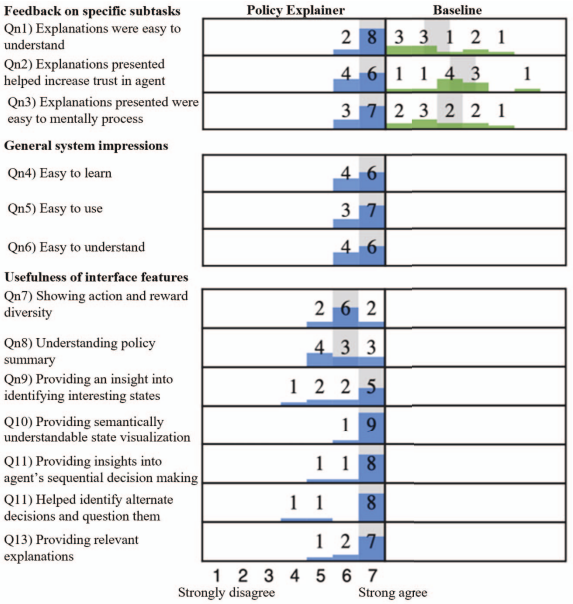

Figure 5: Participants' ratings about various system aspects after the Freeform Stage. Median ratings are indicated in gray.

Visual explanations were preferred to text-based explanations. All ten participants were able to correctly interpret the visual explanations generated by PolicyExplainer, and preferred it over the baseline's text explanations. One possible reason for this, referenced by four subjects (p3, p7, p9, p10), is that PolicyExplainer's visual explanations were more succinct compared to the verbose text explanations from the baseline. "*The text explanations were hard to understand. However, the visual one was better since the . . . explanations were succinct*" (p10). The idea that the text explanations explicitly required more mental effort was a common theme, stated by three participants (p4, p9, p10). "*The text explanations were hard and made no sense so I gave up*" (p4). "*This work on visualizations for explainability makes more sense than text-based systems because the mental effort in the latter is too much.*" (p9). These comments echo the Likert score ratings in Figure 5(Qn1–Qn3).

Decoding agent behaviour across panels. PolicyExplainer contains several panels (apart from the explanation panels), which several participants mentioned they used to contextualize the agent's reasoning better and validate their understanding of the agent behaviour. For examples, four participants (p2, p4, p7, p10) found the Q-values for certain actions intuitive, despite being RL non-experts. "*Like in state 4, its really interesting to see the rewards for move south and move east are the same, since the agent would complete the task in the same number of steps, but if you look at the Q value for move west and north though they are same its a bit lower than the optimal action because of taking an extra step but still being in the same location*" (p7). "*Q values had encoded rewards which aligned with our expectation of the agent* " (p2).

Two participants (p9, p10) especially liked that the state value overview arranged states by criticality. "*In state 276, the taxi already had a passenger so doing anything else that takes it away from the goal position, thus will have a more negative reward than the optimal action*" (p9). Another participant commented how the features presented in the agent explanation mimic what a human would think if they were the agent. "*Here in this explanation of why the agent*

moved south, it's interesting to see that the agent doesn't look at the destination feature before it picks up, it's just its own position and passenger position. Once it does, it then looks at the destination location. This thinking sort of aligns in the way humans would think which is interesting to see in a robot" (p4).

All ten participants found the state and trajectory visualizations, along with the subgoal, to be helpful in understanding and validating the explanations. "*The subgoal is easy but interesting to see, since it tells us what the robot is trying to achieve and I can easily validate it from the trajectory view. Same goes with the state visualizations, they are easy to understand and easily help me to understand the explanation which is way harder in the text based format*" (p4).

Making sense of unexpected StackBot behaviour. All ten participants used the StackBot domain, and all ten considered it to be the most interesting domain, since the agent performed counterintuitive actions. The system's explanations helped them to understand the agent's reasoning. "*The questions I asked in StackBot helped me see the application of this interface better. It shows a very clear use of the system. Especially the 'Why not?' question*" (p1).

On-demand training to improve usability. Though overall of the users found PolicyExplainer easy to learn, use, and understand (Figure 5(Qn4–Qn6)), two participants (p3, p4) mentioned that additional training time could help them more intuitively understand the system's functions and improve the user experience. Each suggest including on-demand user guides and tutorials. "*Once you explained the interface I found the visualizations easy. Some sort of tutorial is needed though to understand the interface.* " (p3). "*Maybe you could add a tutorial for users for the interface.*" (p4). This functionality was not necessary during the study, as the administrator was present to assist participants if they were stuck or confused.

8.3 Study #2: HIV Domain Experts

For Study #2, we evaluated PolicyExplainer with three researchers (e1-e3) over several weeks. These experts research HIV and vaccinology, particularly in low-income countries. Each had at least four years research experience, but none had technical familiarity about RL. Communication included emails and videoconference interviews, as well as pair analytic sessions with both PolicyExplainer and the baseline interface, primarily using the safety-critical HIV domain (though the other domains were also demoed).

Pair Analytics. *Pair analytics* [5] is an established method for visualization evaluation by capturing reasoning processes in visual analytics. To evaluate an interface such as PolicyExplainer, a visualization expert well-versed with the system functionality "drives," while the study participant freely makes analysis and investigative decisions based on their own expertise and desires. Freeform verbal discussion between the driver and participant is the basis for understanding of the participant's sensemaking process as well as what specific insights are uncovered during investigation.

Domain Expert Feedback As hoped, the domain experts provided several comments about PolicyExplainer's explanations for HIV treatment based around their previous clinical experiences. All the three experts found PolicyExplainer easy to use and understand, particularly compared to the baseline, and were able to correctly interpret generated explanations. "*The interface is easier to understand*" (e1). Likewise another professor noted that, "*It's streamlined and clean. Very spacious and clear to look at actually*" (e2). One expert particularly liked the policy cluster panel: "*There are some real nice clusters here and State 1 is an outlier. Very interesting. This cluster does suggest that the agent did learn giving same drugs to similar patients* "(e2).

Two of the three experts noted that the question-and-answer dialogues were helpful in letting them think through and analyze diverging view points. As one commented, "*This interface also helps ask questions so it actually makes us think and check about the other possibilities. Especially the Why Not [question]* "(e3).

In terms of expanding PolicyExplainer to work with safety-critical domains, one participant suggested incorporating user feedback into the system which updates the policy, thus making the agent adapt to the patient's needs. "*Can you change the agent behavior and the agent adapts based on the physician's response? Not sure how hard it is but an adaptive interface would be so interesting. This aligns with something we call as intervention consideration for different patients*" (e2). Another expert suggested showing confidence values instead of Q-values. Another expert also suggested such a system could be effectively tailored for clinicians in rural and poor settings: "*AI agents and visualizations are not that abundant in healthcare. This interface can be used as an assistive agent in low income countries*" (e3). In such settings, where users would likely have less technical proficiency and visualization literacy, additional narrative cues could be employed: "*Adding more annotations to the visualizations and the interface would be easier for us who don't know computer science*" (e2).

9 Discussion

Here, we discuss how the process of developing and evaluating PolicyExplainer demonstrates how visualization-based approaches can be leveraged to decode the behavior of autonomous RL agents, particularly for RL non-experts. We also discuss some of the system's current limitations, and how they can be addressed in the future.

PolicyExplainer effectively showcases real-world and safety-critical domains. Each of the three tested domains represent reasonable RL problems; the Taxi domain, while simple, is used for demonstrating RL techniques, and the HIV and StackBot domains mimic "real world" problems in terms of complexity. In particular, the HIV domain represents a safety-critical domain where model recommendations require justification and interpretability to be trusted. In both studies, participants correctly interpreted the generated explanations from PolicyExplainer and considered them informative, indicating that systems like PolicyExplainer can effectively promote trust and interpretability for users who are not experts in RL.

Applicability to RL experts. While PolicyExplainer was designed for RL non-experts users, we believe these types of tools also have potential applicability for RL experts. For example, when designing and debugging an agent, PolicyExplainer can be used to validate if the agent is acting according to human preferences; if not, the developer can tune and re-train the model (such a usage scenario falls into the domain of interactive reinforcement learning; for an overview, see [6]).

Increasing Scalability and Explanatory Robustness in PolicyExplainer. Based on PolicyExplainer's current implementation, one area for future work is scaling to more complex domains, such as autonomous cars and robot-hand manipulation, where the states, actions, and even the attributes defining the states might have a huge (or even continuous) space. One strategy is to generate human-interpretable abstractions over such large and/or complex state representations based on the "concept" techniques presented by Kim et al. [18]. Such "state abstractions" could significantly reduce the state/action space while still providing users with explanations of the domain's dynamics. Alternatively, aggregation and navigation techniques such as those from Wang et al. [31] and Wongsuphasawat et al. [33] can be adopted for the interface's explanation and trajectories as a way to reduce visual complexity.

In a similar manner, we are also exploring ways to increase the scalability and explanatory robustness of visual designs; such as increasing the number of actions that can be shown in the Policy Detail View or sorting how features are presented in the Explanation View to increase interpretability. However, it is an open question as to how such modifications can best be done in a way that balances overall human interpretability and usability while still capturing the domain dynamics, particularly for RL non-expert users.

10 Conclusion

We view PolicyExplainer as a first attempt to make a generalizable visual interface to support interactive querying and explanation of an RL agent. By creating visualizations to answer human queries about agent decision-making, these tools can increase trust in automated systems particularly for non-expert users. PolicyExplainer supports both model-based and model-free RL across a variety of domains, and employs a succinct policy explanation methodology to visuallly answer *Why?, Why not?*, and *When?* questions. A user study indicates these visual explanations are preferable to text explanations created by natural language generation; visual explanations were found to increase trust in the decisions given by an RL agent, including in domain experts for a safety-critical healthcare domain. Future work will expand on domains that can be handled by visual analytics approaches, including those with extremely large state and actions spaces that are not easily human-interpretable.

11 Acknowledgment

This research was supported by the U.S. National Science Foundation through grant OAC-1934766.

References

[1] https://gitlab.tue.nl/ha800-hri/hayes-shah.

[2] D. Abel, J. MacGlashan, and M. L. Littman. Reinforcement learning as a framework for ethical decision making. In *AAAI Workshop: AI, Ethics, and Society*, vol. 16, p. 02. Phoenix, AZ, 2016.

[3] B. M. Adams, H. T. Banks, H.-D. Kwon, and H. T. Tran. Dynamic multidrug therapies for hiv: Optimal and sti control approaches. *Mathematical Biosciences & Engineering*, 1(2):223, 2004.

[4] D. Amir and O. Amir. Highlights: Summarizing agent behavior to people. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1168–1176, 2018.

[5] R. Arias-Hernandez, L. T. Kaastra, T. M. Green, and B. Fisher. Pair analytics: Capturing reasoning processes in collaborative visual analytics. In *2011 44th Hawaii international conference on system sciences*, pp. 1–10. IEEE, 2011.

[6] C. Arzate Cruz and T. Igarashi. A survey on interactive reinforcement learning: Design principles and open challenges. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, pp. 1195–1209, 2020.

[7] J. Choo and S. Liu. Visual analytics for explainable deep learning. *IEEE computer graphics and applications*, 38(4):84–92, 2018.

[8] S. Deshpande, B. Eysenbach, and J. Schneider. Interactive visualization for debugging rl. *arXiv preprint arXiv:2008.07331*, 2020.

[9] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[10] F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 0210–0215. IEEE, 2018.

[11] M. Fox, D. Long, and D. Magazzeni. Explainable planning. *arXiv preprint arXiv:1709.10256*, 2017.

[12] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding Atari agents. In *International Conference on Machine Learning*, pp. 1792–1801. PMLR, 2018.

[13] B. Hayes and J. A. Shah. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pp. 303–312. IEEE, 2017.

[14] W. He, T.-Y. Lee, J. van Baar, K. Wittenburg, and H.-W. Shen. Dynamicsexplorer: Visual analytics for robot control tasks involving dynamics and lstm-based control policies. In *2020 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 36–45. IEEE, 2020.

[15] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE transactions on visualization and computer graphics*, 25(8):2674–2693, 2018.

[16] S. H. Huang, K. Bhatia, P. Abbeel, and A. D. Dragan. Establishing appropriate trust via critical states. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3929–3936. IEEE, 2018.

[17] T. Jaunet, R. Vuillemot, and C. Wolf. Drlviz: Understanding decisions and memory in deep reinforcement learning. In *Computer Graphics Forum*, vol. 39, pp. 49–61. Wiley Online Library, 2020.

[18] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pp. 2668–2677. PMLR, 2018.

[19] M. Komorowski, L. A. Celi, O. Badawi, A. C. Gordon, and A. A. Faisal. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature medicine*, 24(11):1716–1720, 2018.

[20] Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[21] B. Y. Lim, A. K. Dey, and D. Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2119–2128, 2009.

[22] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[23] S. McGregor, H. Buckingham, T. G. Dietterich, R. Houtman, C. Montgomery, and R. Metoyer. Interactive visualization for testing markov decision processes: Mdpvis. *Journal of visual languages & computing*, 39:93–106, 2017.

[24] J. Miller, C. Hsu, J. Troutman, J. Perdomo, T. Zrnic, L. Liu, Y. Sun, L. Schmidt, and M. Hardt. Whynot, 2020. doi: 10.5281/zenodo.3875775

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[26] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization. doi: 10.23915/distill.00007

[27] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[28] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[29] J. van der Waa, J. van Diggelen, K. v. d. Bosch, and M. Neerincx. Contrastive explanations for reinforcement learning in terms of expected consequences. *arXiv preprint arXiv:1807.08706*, 2018.

[30] J. Wang, L. Gou, H.-W. Shen, and H. Yang. Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE transactions on visualization and computer graphics*, 25(1):288–298, 2018.

[31] J. Wang, W. Zhang, H. Yang, C.-C. M. Yeh, and L. Wang. Visual analytics for rnn-based deep reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[32] J. Whittlestone, K. Arulkumaran, and M. Crosby. The societal implications of deep reinforcement learning. *Journal of Artificial Intelligence Research*, 70:1003–1030, 2021.

[33] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12, 2017.

[34] Z. Yang, S. Bai, L. Zhang, and P. H. Torr. Learn to interpret Atari agents. *arXiv preprint arXiv:1812.11276*, 2018.

[35] V. Yoghourdjian, D. Archambault, S. Diehl, T. Dwyer, K. Klein, H. C. Purchase, and H.-Y. Wu. Exploring the limits of complexity: A survey of empirical studies on graph visualisation. *Visual Informatics*, 2(4):264–282, 2018.

[36] D. Young and B. Shneiderman. A graphical filter/flow representation of boolean queries: A prototype implementation and evaluation. *Journal of the American Society for Information Science*, 44(6):327–339, 1993.

[37] Q.-S. Zhang and S.-C. Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.