Autonomous Agents and Multi-Agent Systems (2023) 37:34
https://doi.org/10.1007/s10458-023-09615-8

A novel policy-graph approach with natural language and counterfactual abstractions for explaining reinforcement learning agents

Tongtong Liu[1] · Joe McCalmon[1] · Thai Le[2] · Md Asifur Rahman[1] · Dongwon Lee[3] · Sarra Alqahtani[1]

Accepted: 9 July 2023

Abstract
As reinforcement learning (RL) continues to improve and be applied in situations alongside humans, the need to explain the learned behaviors of RL agents to end-users becomes more important. Strategies for explaining the reasoning behind an agent's policy, called *policy-level explanations*, can lead to important insights about both the task and the agent's behaviors. Following this line of research, in this work, we propose a novel approach, named as CAPS, that summarizes an agent's policy in the form of a directed graph with natural language descriptions. A decision tree based clustering method is utilized to abstract the state space of the task into fewer, condensed states which makes the policy graphs more digestible to end-users. We then use the user-defined predicates to enrich the abstract states with semantic meaning. To introduce counterfactual state explanations to the policy graph, we first identify the critical states in the graph then develop a novel counterfactual explanation method based on action perturbation in those critical states. We generate explanation graphs using CAPS on 5 RL tasks, using both deterministic and stochastic policies. We also evaluate the effectiveness of CAPS on human participants who are not RL experts in two user studies. When provided with our explanation graph, end-users are able to accurately interpret policies of trained RL agents 80% of the time, compared to 10% when provided with the next best baseline and 68.2% of users demonstrated an increase in their confidence in understanding an agent's behavior after provided with the counterfactual explanations.

Keywords Reinforcement learning · Explainable AI · XRL · Autonomous

1 Introduction

Neural networks have been successfully applied to reinforcement learning (RL) problems in areas of control [1] and games [2]. Due to impressive performance in these areas, RL has started appearing in performance-sensitive real-world applications like chip design [3], server management [4], and robotics [5]. However, the black-box nature of neural network decisions increases the need for end-user trust when deploying a new RL system [6].

Extended author information available on the last page of the article

Explainable reinforcement learning (XRL) attempts to build this trust by explaining the behavior of an RL agent that uses neural networks in its decision making process.

In general, we argue that a truly useful XRL has to satisfy several desiderata. First, XRL has to explain the entire policy (i.e., behavior) of the agent to end-users who likely have insufficient knowledge of RL. Policy-level explanations attempt to reveal the policy of an agent for many different regions of the state space, and "before" the agent has taken any actions [7–9]. However, existing XRL methods often focus on explaining the decisions of an RL agent in specific instances [10, 11]. Second, XRL needs to be generalizable for both deterministic and stochastic RL policies in continuous or discrete state spaces. However, existing XRL approaches explain only deterministic policies [8] or discrete state spaces [9]. Third, XRL needs to deliver explanations, which end-users can interpret, in a natural language format with as minimal user intervention as possible. Forth, the end-users should be given control over the size and thoroughness of the explanation based on their needs. Moreover, the explanations must accurately describe the agent's policy, which can be evaluated using the fidelity test [12]. Finally, XRL must answer the "why not?" questions in form of counterfactual explanations [13].

In this paper, satisfying these desiderata, we propose a generalizable policy-level explanation approach, named as **Comprehensible Abstract Policy Summaries** (**CAPS**), that provides "what" and "why not" explanations for both stochastic and deterministic policies of an RL agent and displays the policy as a directed graph, embedded with intuitive natural language (NL) explanations and counterfactual explanations. CAPS first collects, from the user, simple NL predicates which describe potential aspects of the agent's state (e.g., "car moves right" or "car stops at the top of the mountain" in the mountain car environment.) CAPS then collects no more than 500 timesteps from the RL agent trajectories as representatives of the agent's behavior. In order to make the explanation process tractable, CAPS uses a clustering algorithm, CLTree [14], that abstracts the agent's states into a hierarchy of different configurations of clusters. Each cluster groups similar states into one abstract state. Then, a heuristic optimization technique is developed to select the best configuration of the clusters, which is determined by the accuracy of state transitions and the end-user interpretability. For each cluster, CAPS also identifies whether the agent considers the states in the cluster to be *critical*, extending the methodology proposed in [6]. Then, using the generated clusters, CAPS forms the agent's policy (π) and transition function as a directed graph, where the nodes are the clusters of states, forming abstract states, and the edges represent the actions chosen by π, as well as the probability of transitioning from one abstract state to the next. To enrich the generated graph with more semantic meaning, CAPS labels the abstract states (i.e. graph nodes) with concise NL explanations using the user-defined language predicates and Boolean algebra. By controlling the height of the CLTree, CAPS gives the end-user the choice of generating different policy graphs with different sizes such that each size corresponds to different levels of abstraction. Lastly, we include two metrics to the graph nodes namely; timesteps and failure probability to provide counterfactual explanations of why the agent takes certain actions. The timesteps measure the time the agent will take from each state to the goal state while the failure probability measures how likely the agent will fail the task if it is placed in each state. We design a tool for the users to interactively ask "why" and "why not" using CAPSgraphs.

To demonstrate the utility of CAPS, we use 5 RL environments. We trained the RL agents in the environments using an algorithm for stochastic policies, PPO [15] and an algorithm for deterministic policies, DQN [16]. Our experimental results indicate that CAPS is generalizable with minimal user interventions. To measure how close the explanation graphs are to the agent's policy, we ran fidelity testing for each environment and

compared it against two baselines which share the most in common with our approach [8, 9]. The results show that the accuracy of the policy graphs produced by CAPS is either superior or comparable to those produced by the baselines. We also conducted two user studies; the first tests the general explainability of the generated graphs with different abstraction levels against the two baselines while the second study investigates the users' understanding of the counterfactual explanations generated by CAPS in the form of timesteps and failure probability metrics. Results show that users presented with CAPS graphs identify the correct state and next action of an agent 80% of the time, compared to just 10% for the next best baseline. For the counterfactual explanations, at least 68.2% of users demonstrated an increase in confidence after getting the chance to ask 3 counterfactual questions in the "what if" form.

Our contributions are summarized as follows:

1. We introduce a policy explanation approach for creating abstract policy graphs with different state abstraction levels, which provides the end-users with more control over the size of the graph.
2. We provide an optimization heuristic for maximizing the accuracy of our policy graphs, while minimizing the size of the graph.
3. We highlight which abstract states the agent considers most important and critical to the task which builds trust in end-users [6].
4. We propose a novel algorithm for generating a single natural language representation for each abstract state. Our method overcomes the issue of previous natural language grounding methods, where too many contradictory state regions can degrade the quality of the explanation as in [7].
5. We develop a method for counterfactual explanations by perturbing the identified critical states. The counterfactual explanations are provided using two metrics: timesteps and failure probabilities.

2 Related work

Researchers have highlighted important pixels in a frame of an Atari game [19–21], and generated contrastive Atari frames which would cause the agent to take another action [10]. [11] use reward decomposition to identify the part of the goal a specific action was meant to achieve. Other methods identify states which an RL agent determines is important for the outcome of the episode. The entropy of the agent's policy and the maximum difference of the value function have been used to determine states in which a certain action needs to be taken to avoid failure [6, 22]. Highlighting such states improves a user's trust in the agent [6].

For policy-level explanations, the developed approach in [17] generates decision trees that mimic the agent's policy, and then interprets the rules formed by the learned tree. Structured models of the environment have been created in [13] to trace the outcomes of the agent's policy and produce explanations. These models are not generalizable across tasks and are difficult to create in large environments, or in environments where the complete state transition dynamics are unknown. Another method has been proposed in [18] to generate an explanation of a policy by comparing it to a user-generated 'foil' policy. It, however, requires the end-user to have a foil policy in question and know the dynamics of

Table 1 Summary of surveyed related works versus CAPS

Algorithm	[9]	[7]	[8]	[17]	[13]	[18]	CAPS
Policy Explanation	✓		✓		✓	✓	✓
Explains the Transition Function	✓		✓		✓		✓
Targets End-users		✓		✓	✓	✓	✓
Assumes Minimal End-user Knowledge	✓	✓					✓
Highlight Critical States							✓
Explains Continuous State Spaces		✓		✓			✓
Explains Stochastic Policies			✓				✓
Optimizes Size of Explanation							✓
Natural Language Explanations		✓			✓	✓	✓
Counterfactual Explanations					✓	✓	✓

the environment. In [7], policy-level explanations have been generated in response to user queries. They also propose a method for translating state regions to natural language. Our proposed approach improves their methods for translating state regions, and generates policy explanations which encompass more scenarios than just those that the user queries. We claim that the policy graphs are more interpretable for the policy-level explanations since they explain both the policy and the transition function, and have the capacity to generate the query explanations as shown in [7].

An abstracted Markov chain has been built in [9] and is displayed as a directed graph to explain the agent's policy. This approach clearly visualizes the structure of the environment, and the agent's path while traversing it. However, the generated policy graph is not interpretable beyond its shape, because the nodes representing abstract states are not given natural language labels. For stochastic policies and larger environments, the produced graphs can quickly become too large [9].

Zahavy et al. [8] create state abstractions in Atari environments using clustering. Their method produces abstract policy graphs, but requires heavy manual feature engineering and knowledge of the environment. In addition, the quality of the abstract groupings is dependent on the engineered features, and they do not provide a way to interpret their policy graphs beyond environments with image representations. Our method alleviates these weaknesses by clustering the state abstractions before any feature engineering is applied, but the two can be used in conjunction if the user already has extensive knowledge of the environment.

The initial stage of this work has been published in [23]. In this paper, we extend [23] to provide counterfactual explanations answering the users' questions of "why" and "why not". Moreover, we design a detailed user study to evaluate the counterfactual explanations of CAPS. The comparison between CAPS and prior work is listed in Table 1.

3 Background

3.1 Reinforcement learning

An RL agent learns by acting within an environment. Their interaction can be characterized as a Markov Decision Process (MDP) described by the tuple (S, A, P, R), where S is

the set of states, A is the set of actions available to the agent, P is the transition function such that $P(s_t, a_t)$ produces a distribution over all possible next states at time t, and R is the set of all possible rewards that an agent can receive for actions taken in states. The goal of an agent is to learn the policy, $\pi(s_t) = a_t$, which would maximize the total discounted reward over the whole task. To help discover the optimal policy, an agent learns to approximate the value function, $v(s_i)$, which is the expected discounted reward that can be gained by being in the state s_i and following the policy, π. Formally, the value function is:

$$v(s_i) = \mathbb{E}\left(\sum_{t=i} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \right) \tag{1}$$

Note that the state-value function can be obtained from the action-value function: $V_\pi(s_0) = Q_\pi(s_0, \pi(s_0))$. These are used in methods based on Q-Learning, Sarsa(λ), and actor-critic methods [24]. Therefore, the value-function is generally available alongside the policy of a trained agent. In the case of stochastic policies, π produces a probability distribution over A, $\pi(\cdot \mid s)$, which can then be sampled to choose a_t. The goal of CAPS is to display information about the policy, π, and the transition function, P, to the user as an easily understood directed graph.

3.2 CLTree

CAPS relies on clustering, an algorithm for finding the natural groupings of data points within the entire set. Specifically, we use the clustering algorithm, CLTree [14], which trains a decision tree to separate the data that belongs to the set, from data which is artificially inserted into the set uniformly. By learning the feature boundaries which separate real data from fake data, the nodes in the tree form natural clusters. In addition, these clusters can be combined or separated by traversing up or down the tree. Therefore, by pruning the tree, each cluster can be more inclusive or exclusive according to the user's needs.

4 The proposed approach: CAPS

The main deliverable from CAPS is a directed graph to describe the agent's policy in NL predicates and a set of metrics for counterfactual explanations. We formulate the output graph G as $G = (V, E)$ to explain the RL policy components, states, actions, and transiting probabilities between the states. V is a set of vertices representing the RL agent's states S from the MDP representation of the environment. E is the edges connecting the vertices matching the actions $A = \pi(S)$ in the MDP and defined as $E \subseteq \{(v_1, v_2) \mid (v_1, v_2) \in V^2\}$. Since G is a directed graph, we can annotate E with the transition probabilities P as defined in the MDP.

Directly mapping the states S, actions A, and transitions P from the original MDP to V and E in G is intractable for the end-user interpretation. For instance, generating G for an episode of 100 timesteps collected from a policy π could, in the worst case,

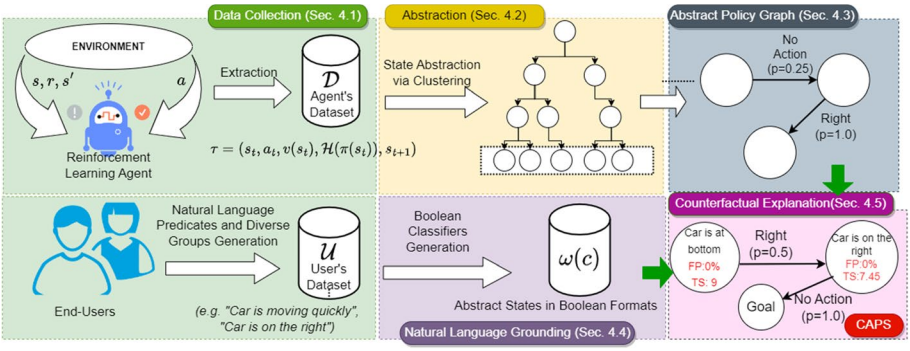


Fig. 1 The overall architecture of the CAPS approach

include 100 vertices to represent each state in the episode. Given the definitions of the RL agent's policy π and the graph G, we frame this problem within CAPS framework as a five step process as depicted in Fig. 1 and listed as follows:

1. Collects simple natural language (NL) predicates from users that describe potential aspects of the agent's environment.
2. Uses a clustering algorithm to group similar states into one abstract state. A heuristic optimization technique selects the best configuration of the clusters, as determined by the accuracy of the state transitions and end-user interpretability.
3. Identifies for each for each cluster also whether the agent considers the states in the cluster *safety-critical*.
4. Forms the agent's policy (π) and transition function as a directed graph $G = (V, E)$, where the nodes v are the clusters of states and the edges E represent the actions chosen by π as well as the probability of transitioning from one abstract state to the next.
5. Labels the abstract states (graph nodes) with concise NL explanations using the user-defined language predicates and Boolean algebra to enrich the graph with more semantic meaning,.

The aforementioned steps are explained in details in the following subsections.

4.1 Data collection

In this step, we collect two small datasets, U, from the end-user, and D, from the RL agent's execution. The user is asked to create natural language predicates that concern them about the environment and the agent's task. For example, for a self-driving car agent, the user could give predicates like "the car is moving fast", "there is a stop sign", or "car on the right". Those predicates will be used later to append G with semantic meaning that supports the end-user's interpretation. For the agent's dataset D, we collect a small number (< 500) of timesteps from its execution traces, possibly collected over multiple episodes, consisting of tuples, $\tau = (s_t, a_t, v(s_t), \mathcal{H}(\pi(s_t)), s_{t+1}) \in D$. s_t is the state of the agent at time

t, which will be used to form the vertices V of G, while a_t is the action that the agent's policy chooses based on the current state, $a_t = \pi(s_t)$, and will be mapped to the edges E in G. The state value $v(s_t)$ is collected to estimate how good it is for the agent to be in state s_t for the future reward from Eq. 1. This value helps groups states which the agent views as similarly important to the task's success (Sect. 4.2). The policy entropy $\mathcal{H}(\pi(s_t))$ defines the entropy of the probability distribution generated by π in state s_t. We use the policy entropy to highlight clusters of states as critical (Sect. 4.2.1). For deterministic policies which do not output a probability distribution, we instead use the maximum difference of the Q-function among actions as a proxy for policy entropy, as in [6]. This is defined as $max_{a_i,a_j \in A}(Q(s_t, a_i) - Q(s_t, a_j))$, where $Q(s_t, a_i)$ is the learned value of taking action a_i in state s_t.

4.2 Policy state abstraction via clustering

The motivation of CAPS is to generate more human-interpretable graphs than the Markov chains made from the base MDP and the RL agent's policy. Therefore, the number of nodes in the resulted graph of CAPS should be much lower than the number of distinct states in the base MDP. To achieve this, we adopt the notion of state abstraction from RL literature [24, 25]. State abstraction involves grouping the grounded states of the agent's MDP into groups of similar states, reducing the size of the entire state space. Prior work [9] has abstracted the states using feature importance or significance. In CAPS, we instead use a clustering algorithm eliminating the need to manually engineer the state features as in previous work. We can directly apply clustering algorithms to abstract the state space if we assume that groups of similar states S_1, S_2 share similar policies $\pi(S_1) = \pi(S_2)$ or value functions $v(S_1) = v(S_2)$ as in [25]. This assumption, however, does not always hold, thus we include the value function $v(s_t)$ and action a_t from the collected tuples τ for more accurate clustering.

Using CLTree algorithm [14], we cluster each $\tau \in D$ into similar groups. The generated leaf clusters C in the tree represent the abstract states. As a result, each abstract state $c \in C$ groups all states which are *interchangeable* under the agent's policy such that either the agent behaves similarly when starting in any $s \in c$, or $\pi(\forall s \in c) = a$ [8, 9], or the agent assigns similar values to each $s \in c$ [9]. The advantage of using CLTree clustering over K-means or a modified K-means [8] is that the user has direct control over the size of the graph's nodes V. From a finished CLTree pruned to a specific depth, traversing the tree can increase or decrease the level of abstraction, giving a dynamic understanding of the policy. In addition, by viewing the agent's policy at different abstraction levels, RL developers can gain additional information about how the environment and the agent's policy function. As an example, Fig. 2 shows two policy graphs, without natural language labels, at two sequential levels of the tree.

4.2.1 Heuristic optimization for clustering

Using CLTree for the state abstraction equips the user with the ability to decide the level of abstraction by choosing the height of the tree. To increase the usability of CAPS for users

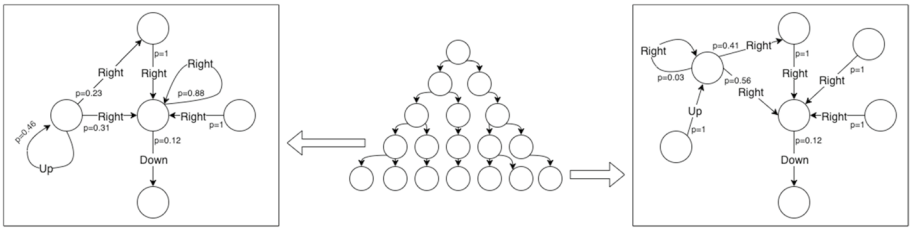


Fig. 2 Example of Abstract Policy Graphs produced by CAPS before translation for CLTree of height 4 and 5 (left, right)

with minimal to no knowledge about RL and CLTree, we develop a heuristic optimization technique to find the optimal set of clusters C^*. Our heuristic is motivated by the fact that as the policy graph G grows in the number of clusters C (i.e. $\mid V \mid$), the error between the true environment transition function P from the MDP and the one estimated by G, as well as the error between the agent's true policy π and that presented in the graph, decreases. However, in order for G to be interpretable to a human end-user, we must limit the number of vertices V in the graph.

Let L be the set of tree heights, and C_l be the set of clusters at a height, $l \in L$. We wish to choose l such that the clusters in $C^* = C_l$ form an accurate policy graph with respect to the value function $v(s)$ and the policy of the agent π within each cluster. In addition, we want to penalize the size of the graph, so that the policy graph G remains interpretable for humans with less vertices V. To achieve that, we introduce two heuristics; the value score and the cluster policy. Informally, the *value score* represents the error between the environment's transition function $P*$ as estimated by the clusters, and the true transition function P in the MDP. It does this by comparing the true value functions of the states in each cluster $v(s) \forall s \in C$ and the estimated value function according to the estimated transition function. This heuristic is inspired by the Value Mean Square Error(VMSE) evaluation criterion in [8]. The second heuristic is the entropy of the *cluster policy*. The *cluster policy*, $\Pi(c)$ defines the probability of taking an action, a, given a cluster, c, as the proportion of $s \in c$ such that $\pi(s) = a$. We minimize the average entropy of the *cluster policy*, $\mathcal{H}(\Pi(c))$, across all $c \in C$, so that it is clear to the end-user which action the agent will take in the abstract state, c.

Recall, $v(s)$ is the value of the state s, as determined by the agent's value function (Eq. 1). We formally define the *value score*, $\zeta(C(l))$, as:

$$\zeta(C(l)) = \frac{1}{\mid C(l) \mid} \sum_{c \in C(l)} (V_{gt}(c) - V_{est}(c))^2 \tag{2}$$

where $V_{gt}(c)$ is the ground-truth value of an abstract state, c, found by:

$$V_{gt}(c) = \frac{1}{\mid c \mid} \sum_{s \in c} v(s)$$

and $V_{est}(c)$ is the estimated value of an abstract state, c, found by:

$$V_{est}(c) = \gamma \sum_{c_i \in C(l)} P^*(c, c_i) V_{gt}(c_i)$$

where γ is the discount factor from Eq. 1 and $P^*(c, c_i)$ is the probability the agent transitioning from cluster $c \in C_l$ to cluster $c_i \in C_h$, estimated as the proportion of (s_t, a_t) pairs $\in c$ which transition to $s_{t+1} \in c_i$:

$$P^*(c, c_i) = \frac{1}{\mid c \mid} \sum_{(s,a,s_{t+1}) \in c} I(s_{t+1}) \tag{3}$$

where I is an indicator function deciding whether s_{t+1} is in c_i.

Then, we formally define the height of the tree, l, pertaining to the optimal clusters as:

$$\arg\min_{l \in L} \left[\zeta(C(l)) + \left(\frac{1}{\mid C(l) \mid} \sum_{c \in C(l)} \mathcal{H}(\Pi(c)) \right) + \alpha \mid C(l) \mid \right] \tag{4}$$

where α is a parameter controlling the penalty of larger policy graphs and $\mathcal{H}(\Pi(c))$ is the entropy of the cluster policy.

4.3 Policy-level explanation via abstract policy graph

Given a set of abstract states, we create a policy-level explanation by modeling the policy as a directed graph G. Each vertex $v \in V$ represents an abstract state from the clusters generated by CLTree and the edges E are transitions induced by a single action from one abstract state to another, each accompanied by a transition probability. We use Algorithm 2 by Topin and Veloso [9] to append the actions and transition probabilities to our graph G.In Algorithm 2 [9], the creation of edges between abstract states to generate the abstract policy graph is accomplished using a mapping function l and a Markov chain transition matrix. The function l maps states from the original MDP (grounded states) to abstract states in G. Initially, the transition matrix is set to zero. Then, the mapping function l is applied along with the transition tuples (s, a, s') within each set of abstract states to calculate the transition probabilities. Specifically, if a transition tuple x is in the set c_i, the original state x_s belongs to the abstract state represented by c_i. The destination state x'_s indicates a connection between c_i and $l(x'_s)$. The transition probability from c_i to c_j corresponds to the fraction of tuples in c_i that lead to a state in c_j. The terminal transitions are identified and lead to a special abstract state. This abstract state represents termination and is symbolized by the highest-numbered row and column in the transition matrix. This algorithm supports only deterministic transitions by allowing one action with the probability of 1 to connect each pair of abstract nodes. We, however, modify the graph generation algorithm in [9] to include the stochastic transition functions by estimating the transition function P^* from Eq. 3 then attaching the actions taken to the edges with their probabilities. This change increases the readability of CAPS's generated graph since the nodes have the NL predicates to describe the abstract states and the edges describe how likely the agent will take certain actions. Figure 2 shows what an example G looks like before incorporating the NL labels to the nodes, at two different heights of the CLTree.

4.4 Natural language grounding

In order to provide interpretable explanations of the agent's abstract policy graph G, we ground the graph nodes (i.e. abstract states) in NL. We use a set of Boolean classifiers that we build based on the user's given predicates in U. We use these classifiers to translate internal knowledge of the MDP states within the abstract states into NL through two levels of translations; MDP state translation and abstract state translation.

4.4.1 MDP state translation

Each state from the base MDP has many features that we can directly translate into natural language predicates. However, those state features may not add any meaningful semantic knowledge to the end-user interpretation of the state. For example, in self-driving car simulation, each environment state would have information about each pixel in the state but the user is only interested in the traffic signs and their impacts on the car's decisions. We approach the challenge of succinctly describing the grounded MDP states as a minimal set cover problem, seeking the smallest logical expression of user-predicates that precisely cover the states. We use Boolean algebra over the set of user-predicates to translate the grounded states into a natural language sentence.

Recall, our collected preliminary data U has the environment-specific user-predicates. We use those predicates to create binary classifiers (F) evaluating the features of each state. The semantic meaning of the *boolean predicates* correspond to some aspect of the state space which we are interested in using to explain the agent's action. Given a set of user-predicates, F, which are binary functions, we map an MDP state $s \in c$ into binary vector, $\omega(s)$, with length $\mid F \mid$, where each element represents whether s satisfies the particular user-predicate, $f \in F$. $\forall c \in C$, and $\forall s \in c$, we map s to $\omega(s)$ through:

$$\hat{\omega(s)} = \left[\begin{cases} 1, & \text{if } f(s) = \text{True} \\ 0, & \text{otherwise} \end{cases} \right] \forall f \in F \tag{5}$$

The *i-th* element in the vector, $\omega(s)[i]$, is equal to 1 when the *i-th* function f_i value for s fulfils the boolean logic encoded in predicate i and 0 otherwise.

We modify the method of [7] for grounding state regions in NL to improve the explanation quality. Specifically, we omit using the Quine-McCluskey algorithm for the Boolean logic minimization and instead we include all predicates to each MDP state at this stage. Later on in grounding abstract states stage, we simply minimize the predicates using a specific threshold because the Quine-McCluskey algorithm has exponential memory and runtime requirements with respect to the size of the predicate set.

Another advantage of our approach of translating the MDP states into NL predicates is that we map the NL to the states regardless of the clustering process. Hence, the set of predicates does not affect the accuracy of the transition function, or the fidelity of the policy graphs. Unlike [8] which uses manual feature engineering to create semantic meaning for the policy. The approach in [8] does the clustering on the hand-crafted features instead of clustering the state space features like in our approach. Thus, their clustering accuracy is correlated with the accuracy of the features which negatively impacted the fidelity scores as we show in the experiment section.

Algorithm 1 Condense Semantic Meanings into a Single, Translated Representation

1: **Input**
2: ω: Binary vectors $\forall s \in c$ computed by Eq.5
3: β, δ: Predicate frequency and correlation threshold
4: DG: The diverse groups for the user-predicates
5: **Output**
6: Exp: The English explanation for each state in the graph G
7: **procedure** GENERATE EXPLANATION
8: **for** c in C **do**
9: **for** i in user-predicate **do** $\omega(c)[i] \leftarrow Eq.6$
10: **for** i in user-predicate **do**
11: **if** $\omega(c)[i] == 1$ **then** $F \leftarrow F \cup i$
12: **end for**
13: **for** (i, j) in F **do**
14: **if** $f_i, f_j \in DG$ **then** $Exp(c) \leftarrow Exp(c) \cup f_i \cup or \cup f_j$
15: **elif** $\rho_{i,j} > \delta$ **then** $Exp(c) \leftarrow Exp(c) \cup f_i \cup and \cup f_j$
16: **else** $Exp(c) \leftarrow Exp(c) \cup f_i \cup or \cup f_j$
17: **end for**
18: **end for**
19: **return** Exp
20: **end procedure**

4.4.2 Abstract state translation

In Eq. 5, each grounded state in the abstract state $\forall s \in c$ is translated into a binary vector $\omega(s)$ of NL predicates. We then condense the set of all binary vectors in the cluster $\{\omega(s) \mid s \in c\}$, into a single vector of predicate values, $\omega(c)$, which provides a concise explanation of the cluster c (Algorithm 1). Algorithm 1 has three steps; finding the frequent predicates for each abstract state, calculating the correlation between those predicates, and translating them into one NL sentence. We label a predicate in a cluster as frequent or not with Eq. 6:

$$\hat{\omega(c)}[i] = \begin{cases} 1, \text{ if } \frac{\sum_{\forall s \in c} \omega(s)[i]}{|c|} > \beta \\ 0, \text{ otherwise} \end{cases} \tag{6}$$

The result, $\omega(c)$, is a binary vector that has 1 at the *ith* predicate if the proportion of each predicate's appearance in all the MDP states, s, in this abstract state, c, is above a threshold β (lines 8-12). Unlike [7], which only presents a predicate as part of the explanation if it appears in all MDP states, our algorithm uses a threshold β to control how frequent a predicate value must appear in the grounded states $\forall s \in c$ to be included in the explanation of the abstract state c. Hence, CAPS guarantees presenting the user with at least some relevant descriptions. We find that in practice, the method of [7] often finds no commonalities

Table 2 Example labels for abstract states produced by CAPS versus [7] for the Cartpole environment

CAPS	Hayes et al. [7]
"Pole is standing up and cart is either moving left or right and cart is either on the left or in the middle"	No Explanation Produced
"Pole is either stabilizing to the right or standing up and cart is moving right and cart is in the middle"	"Pole is not falling left and cart is moving right and cart is in the middle"
"Pole is either stabilizing to the left or standing up and cart is moving left and cart is in the middle"	"Cart is moving left"

between the grounded states, especially as the amount of input data grows (Table 2). We suspect this is because even a single contrastive state can invalidate a predicate following their method.

The second step of Algorithm 1 (lines 13–16) decides which Boolean algebra operators (i.e. or, and) should be used to connect the frequent predicates, so that each node in the graph has a semantically meaningful description. This is done by first requiring the user to group their predicates into *diverse* groups such that no two predicates in the same group can both be true in any grounded state. For example, in a self-driving car, the predicates "car is moving fast" and "car is moving slow" should be in the same group since the car cannot be moving fast and slow simultaneously. If a pair of frequent predicates for an abstract state are part of the same diverse group, we connect those predicates with an "or" operator. Otherwise, we use the Pearson correlation coefficient [26] to find how strong the correlation between the expressive predicates. We use "and" for strong correlations and "or" otherwise. For each pair of predicates i, j, we compute the Pearson correlation coefficient ρ as:

$$\rho_{i,j} = \frac{cov(i,j)}{std_i std_j} \tag{7}$$

where cov is the sample covariance, std_i and std_j are the sample standard deviations of the predicates i and j in $s, \forall s \in c$. We use a threshold δ to control how correlated two predicates must be in order to justify an "and" relationship between them. Algorithm 1 improves upon [7] by including the largest number of predicates in the explanation as possible, while still presenting combinations of predicates that actually occurred in the input data. Using the diverse groups plays a crucial role, since we can still include two predicates in the explanation which can never coexist in the same state, simply by joining them with an "or" conjunction.

4.5 Counterfactual explanation via action perturbation

According to [27], explanatory questions in AI are classified into three types: "What?" (Associative reasoning), "How?" (Interventionist reasoning), and "Why?" (Counterfactual reasoning). The policy-level explanation given by CAPS in the abstract policy graph focuses on explaining the long term decision-making of the agent following the learned policy answering the first 2 questions of "what" and "how". "Why?" questions have been only investigated for RL agents that operate in visual environments [10, 28]. The "why" question requires counterfactual reasoning [10] about alternative outcomes that have not

happened. Here, we build our counterfactual reasoning method for the "why" question based on the associative and interventionist reasoning initially given by CAPS's abstract policy graph. Specifically, we answer the "why?" and "why not?" questions by first identifying the critical states for the agent to successfully accomplish its task, and then perturbing those states to generate alternate behaviors. We explain the impact of those perturbations on the agent's behavior using two metrics, the episode length and the failure probability.

4.5.1 Highlighting critical states

Critical states are defined as states in which it is important to take a certain action [6]. To build trust in an end-user, it is helpful to highlight what the agent does in these critical states, as well as why the agent takes certain actions at those states. For stochastic policies, such as PPO [15], the set of critical states under the policy K_π have been defined as [6]:

$$K_\pi = \{s \mid \mathcal{H}(\pi(\cdot \mid s)) < t\} \tag{8}$$

where $\mathcal{H}(\pi(\cdot \mid s))$ is the entropy of the probability distribution generated by π in state s. Extending this to the setting of abstract states, we define the critical value of an abstract state, $\kappa(c)$, as:

$$\kappa(c) = \frac{1}{\mid c \mid} \sum_{s \in c} \mathcal{H}(\pi(\cdot \mid s)) \tag{9}$$

In this case, a lower critical score, and therefore a lower average entropy, corresponds to a more critical state. We highlight c as a critical abstract state if its critical value is in the bottom n_{th} percentile of the abstract states in its graph. n is arbitrary and only affects the number of critical abstract states which are presented. We chose to use $n = 10$ in our experiments. We highlight those states in red in the abstract policy graph as shown in Fig. 4 and use them to generate meaningful counterfactual explanations.

4.5.2 Action perturbation

Our goal is to perturb the agent's policy in the critical states—i.e., to force the agent to take different actions, and consequently generate counterfactual behaviors for the agent. Given a critical abstract state s_c and the optimal action a at s_c selected by the policy $a = \pi(s_c)$, we generate a counterfactual behavior that differs in some sense from the expected behavior given by π. The counterfactual behavior is generated by enforcing the agent to take a counterfactual action a' rather than action a and roll out its behavior from $\pi(s' \mid a', s_c)$ using its learned policy. We collect a dataset for counterfactual behaviors starting from the critical abstract states of the learned policy in $\delta = (s_c, a, TS, FP)$ where s_c is a critical state, a is the optimal action at state s_c according to the learned policy π, TS and FP are episode length (i.e.timesteps) and failure probability, respectively.

- **Episode Length (TS)**: measures the time steps the agent takes to accomplish its task. Perturbing the agent's policy could result in the agent taking longer time to reach its goal state. Hence, we use this metric to answer the question of "why the agent is not taking action a' instead of a at state s_c?" or "what if the agent takes action a' instead of a at state s_c?" by attaching the timesteps to each abstract state.

- **Failure Probability (FP)**: measures how likely the agent will fail its task when it takes action a' instead of following its policy and taking a at s_c. To estimate the failure, we build on the risk estimation approach proposed in [29] for uncovering failures in RL agents by sampling the agent in states where its performance is low. The intuition behind the failure probability is to estimate how likely the agent will fail the task, from the current state following its policy. Suppose we have an oracle function, g, which computes the probability that the agent will fail the task in the next H timesteps from state s_t. Then, the failure probability can be derived as $\hat{a}_t = g(s_t)$. We then approximate the function g, such that $\hat{g}(s_t) \approx g(s_t)\ \forall s_t$. Since s_t can potentially take infinite values, [30] chooses to approximate g through a neural network, trained with supervised learning. We follow their methodology, which involves first collecting a dataset $\{(s_t, g(s_t))\}$ that contains around 100, 000 samples. This dataset can be collected by observing the agent during execution. We use a neural network with two fully-connected layers of 64 neurons each, and train it to predict $g(s_t)$ from s_t. To compute failure probability for abstract states, we average the failure probabilities for each clustered state in each abstract state and display that in the nodes of CAPS's graph along side the English predicates.

Note on Domains Without Goals. CAPS is designed to explain the policy of RL agents regardless of whether the agent's task is goal-based or not. For the goal-based tasks, the terminal node in the CAPS' graph would represent reaching the goal. On the other hand, in tasks without an explicit goal, the terminal node can be chosen as the termination state of the agent using a specific threshold for the episode's length. To compute the FP for the tasks without explicit goals, we simply consider the episode's length above the threshold as a failure. Moreover, the CAPS' user can have a different definition for the termination of the episode based on the task itself. For example, in a driving simulator domain, the episode termination can be defined using a time threshold for the episode's length. In the experiment section (Sect. 5.2.2), we generate a graph for Blackjack environment which is not an explicit goal-based environment and defined the terminal node as "End of Game" which represents either winning or losing the game. The players could play indefinitely by taking action "stick", which can be shown in Fig. 5's critical state. We, however, think the domains without a clear goal or success/failure state might require different design choices when it comes to explainability tools including CAPS.

5 Experiments

5.1 Experimental settings

We tested CAPS on 5 environments, where 4 environments have discrete state spaces, namely Blackjack, Cliffworld [24], Gridworld [31], and Taxi [32], while the last environment is Mountain Car [32] which has continuous state space. In addition, Blackjack has a stochastic transition function while the rest of the environments have deterministic transition functions. To test the generalizability of CAPS, we tested the environments using two different RL agents: one trained with a deterministic algorithm, DQN [16], and one with a stochastic algorithm, PPO [15].

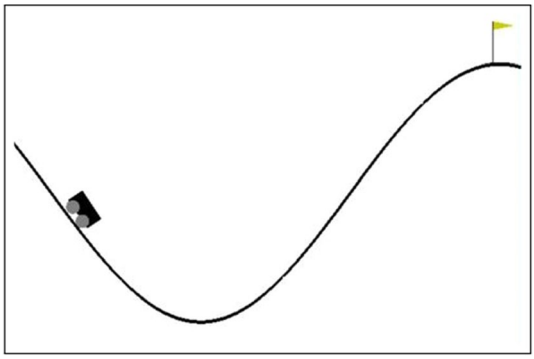


Fig. 3 The Mountain Car environment from Open AI Gym

5.2 Results

We here describe two of the environments tested and show their corresponding policy graphs generated by CAPS. The rest of the environments and their policy graphs are included in Appendix A.

5.2.1 Mountain car

In the Mountain Car environment, a car starts at the bottom of a valley, and must build up enough speed to reach the top of the hill on the right (Fig. 3). The car has 3 actions, either accelerate left, accelerate right, or choose to not accelerate. The state space consists of two continuous features, car position, with 0 corresponding to the bottom of the valley, and cart velocity. Figure 4 shows the policy graph for Mountain Car, produced by CAPS. Nodes highlighted in red are selected as critical by CAPS.

The user-defined predicates for this environment are broken up into two diverse groups. The first describes the position of the car, and includes the predicates "At the bottom", "On the left slope", "On the right slope", "High up on the left slope", and "High up on the right slope". The second group describes the velocity of the car: "Not moving", "Moving right slowly", "Moving left slowly", "Moving left quickly", and "Moving right quickly".

Given only Fig. 3 of the environment and a description of the task, an end-user might suppose the best way to solve the task is to have the car accelerate right until reaching the goal flag. However, such a strategy does not work, because the car fails to build up enough momentum to surmount the hill. The optimal strategy, as discovered by the RL agent, is to first alternate between left acceleration, right acceleration, and no acceleration to build up momentum, and then to stop accelerating once the car is high enough up the left hill. Momentum is then enough to carry the car to the goal. Such a strategy can be seen in the CAPS graph (Fig. 4). The car starts at the bottom and has a stochastic policy that sees it alternate between accelerating left, right, and not at all. Once it builds up enough momentum to reach a position high up on the left slope, it identifies its current state as critical and stops accelerating with high probability. These actions bring the car to a point in which it is rapidly rising up the right slope, and eventually reaches the goal.

Figure 4 shows the optimal policy of the agent and provides the TS and FP of the agent at each abstract state. TS and FP are calculated following the optimal policy manifested in the collected trajectories. The user can understand the optimal policy of the agent by

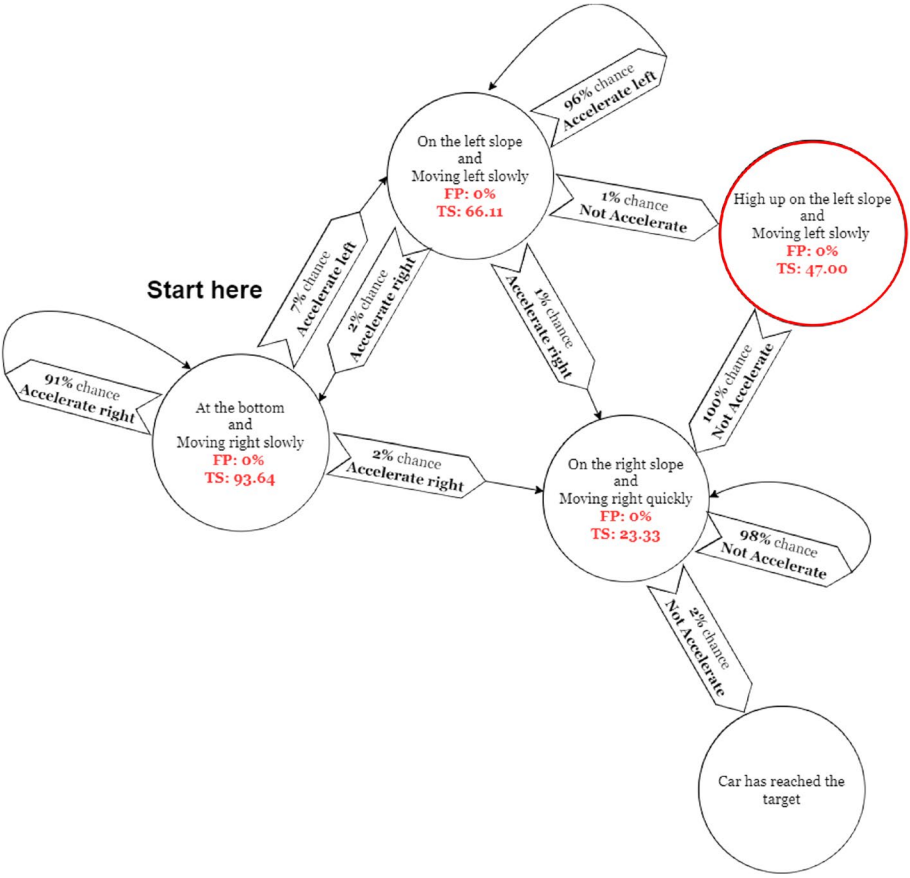


Fig. 4 The policy graph for Mountain Car produced by CAPS. Abstract states outlined in red are critical

placing the agent at the starting state *"At the bottom and Moving right slowly"* and then trace the agent's path by following the action from the one with highest probability to the lowest probability. The edges are generated by collecting no more than 500 timesteps from the RL agent trajectories, clustering each *state-action pair* to create the abstract states, and then calculating the probability distribution for different actions in the same abstract state. For example, the agent moving from the abstract state *"High up on the left slope and Moving left slowly"* will not accelerate 100% of the time, and the probability of failing the task by starting from this state and choose to not accelerate is 0% as shown by FP in that state. The expected remaining steps to finish the task is 47 steps as shown by TS.

5.2.2 Blackjack

Blackjack is a popular card game with a discrete state space that has been solved using RL. The objective of the game is to win money by obtaining a point total higher than the dealer's without exceeding 21. Determining an optimal blackjack strategy proves to be a difficult challenge due to the stochastic nature of the game.

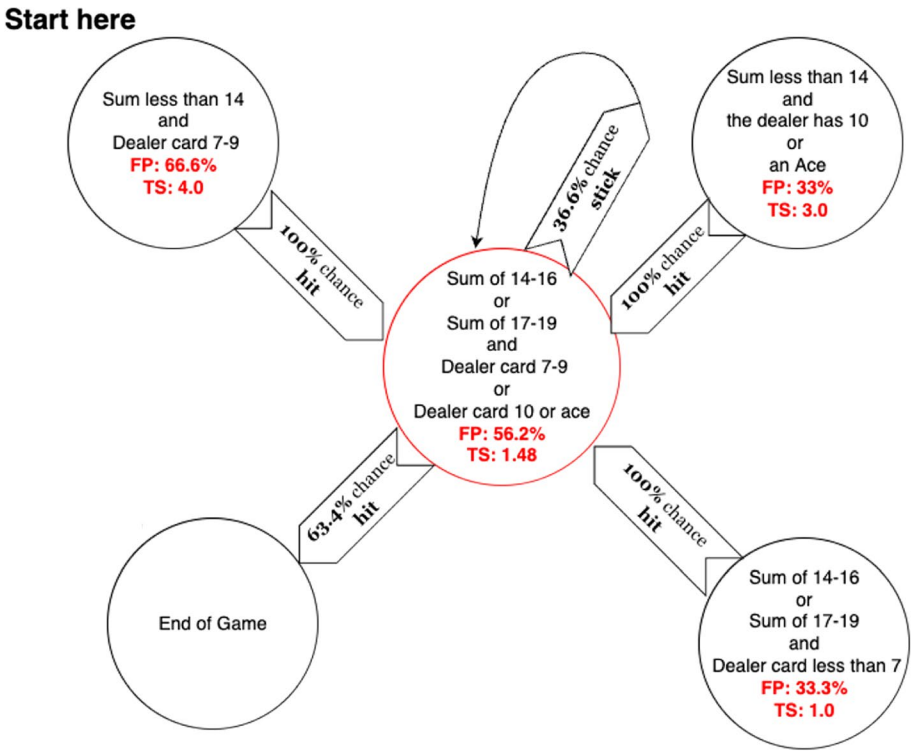


Fig. 5 The CAPS explanation for Blackjack [24]

The game works by assigning each card a point value. Cards 2 through 10 are worth their face value, while Jacks, Queens, and Kings are worth 10 points. An ace is worth either 1 or 11 points, whichever is the most beneficial. This game is placed with an infinite deck (or with a replacement). The game starts with dealer having one face up and one face down card, while player having two face up cards. The player can request additional cards (hit=1) until they decide to stop (stick=0) or exceed 21 (bust). Hence, the possible actions include hitting, standing, splitting, or doubling down. After the player sticks, the dealer reveals their face-down card, and draws until their sum is 17 or greater. If the dealer goes bust the player wins. If neither player nor dealer busts, the outcome (win, lose, draw) is decided by whose sum is closer to 21. The reward for winning is +1, drawing is 0, and losing is -1. The observation of a 3-tuple of: the players current sum, the dealer's one showing card (1-10 where 1 is ace), and whether or not the player holds a usable ace (0 or 1). This environment corresponds to the version of the blackjack problem described in Example 5.1 in [24][1]

The CAPS graph for Blackjack (Fig. 5) can be interpreted as follows. If the player or dealer has a usable ace, that is displayed as "Player has an Ace" or "Dealer has Ace", respectively.

[1] The Open AI Gym design of this game is here: https://github.com/openai/gym/blob/master/gym/envs/toy_text/blaackjack.py.

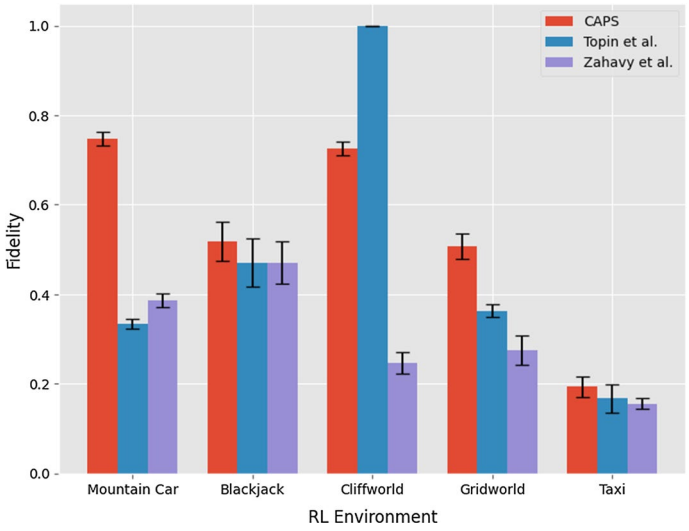


Fig. 6 Fidelity for three baselines in the five environments. Data was collected using a PPO policy

The player has two possible actions, hit or stick. From the top left node, we see that if the player's hand is between 0 and 16, it will choose to hit every time. There are many edges leading out from this node because the environment has a stochastic transition function: choosing to hit from this node does not always transition the environment to the same next node. There are two other nodes where the player's hand is less than 19, but the dealer's hand is good (7-10 or ace). In these nodes, the player will often choose to stay but sometimes will hit. In all other nodes, the player has a good hand and the dealer does not. The player will stay in these scenarios, with two of them being labeled critical. Since this environment is stochastic, the probability of the agent to fail can be above zero as shown in FP of all states in Fig. 5.

5.3 Fidelity accuracy

Alongside the policy graphs, we compare the accuracy of CAPS policy graphs to baselines through a fidelity metric. The *fidelity* of the policy graph is defined as the proportion of actions taken by the trained RL agent that are the same as the actions the policy graph says the agent should take [9]. To calculate fidelity for each environment, we simulate about 2000 timesteps of environment interaction. We also average the fidelity of these trials over 10 different generated graphs by each baseline. The results for CAPS and the two baselines which produce policy graphs, [8, 9], are shown in Fig. 6. In Fig. 6, we see that the fidelity of CAPS is either superior or comparable to [8, 9], except in the case of Cliffworld. In this environment, the policy is deterministic enough that [9], which attempts to memorize (s_t, a_t) pairs, can do so without loss of fidelity. We construct CAPS graphs to favor interpretability and conciseness over perfectly fitting the observed data, but we see that the fidelity accuracy does not suffer under this approach. We also see in Fig. 7 that CAPS policy graphs are similar in fidelity when the data was generated with PPO versus with DQN. This indicates that CAPS is invariant to the agent's learning algorithm, and is compatible with deterministic and stochastic policies.

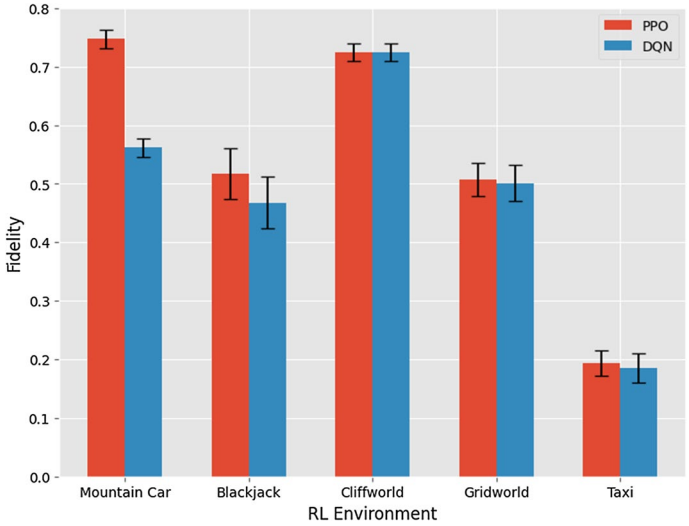


Fig. 7 Fidelity of CAPS in the five environments for two agents trained with PPO and DQN

5.4 Heuristic optimization: analysis study

We perform the analysis on the different scores defined in Sect. 4.2.1 with respect to graph size (in nodes). In Fig. 8a, we plot the relationship between our score heuristic and the graph size. We observe an inflection point where the increase in accuracy is no longer worth the extra nodes on the graph. That point is returned as the optimal graph size. We see in Fig. 8b that the *value score* decreases as graph size increases, indicating that larger graphs can represent the transition function of the environment more effectively. Also, we see that the entropy of the *cluster policy* (Fig. 8c)decreases as graph size increases. This is natural since we cluster the data in part based off of the taken action.

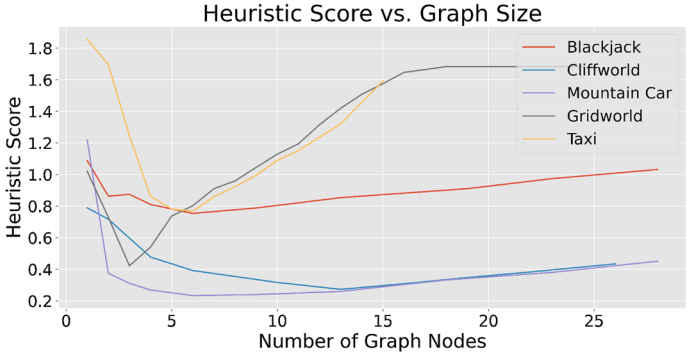
6 User studies

CAPS includes not only developers and researchers but also non-technical end-users as its target audience. We evaluate the effectiveness of CAPS graphs on human participants who are not RL experts in two user studies. Our first user study investigates the impact of CAPS graphs on humans' associative reasoning and interventionist reasoning while the second user study evaluates humans' counterfactual reasoning. The design and results of each study are given below.

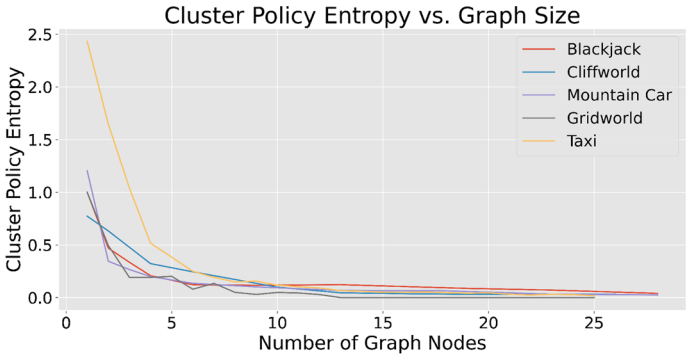
6.1 User Study 1: Associative and interventionist reasoning

6.1.1 Design

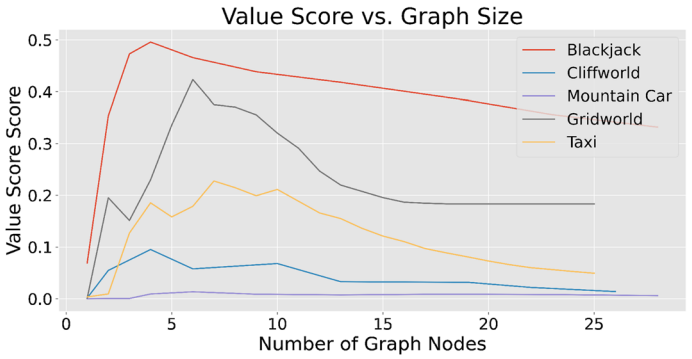
We designed a user-study to quantitatively measure the amount of a CAPS graph provides over two other directly-comparable graph-based policy explanation methods [8, 9]



(a) Heuristic Score vs. Size of CAPS Graph



(b) Cluster Policy Entropy vs. Size of CAPS Graph



(c) Value Score (Eq.2) vs. Size of CAPS Graph

Fig. 8 Plots of the different heuristics used in Eq. 4 versus the number of nodes in a CAPS graph

from the perspective of the end-users. For this study, we present the Mountain Car environment to 163 Amazon Mechanical Turk (AMT) workers, along with a single frame from an episode. Given an explanation graph, we then ask the workers:

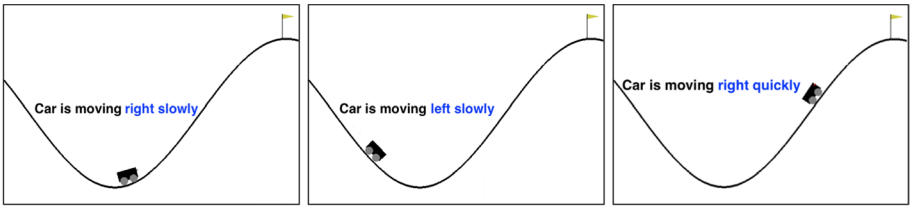


Fig. 9 Three selected distinctive frames of an episode of the *Mountain Car* RL environment that were used for the user-study. The AMT workers are presented with a single scenario—i.e., frame, at a time

Q1. Which state represents the current scenario of the environment?
Q2. Which action the agent will likely execute next?

We reveal the correct answer to *Q1* at the beginning of *Q2*. The Mountain Car environment is selected because actions of a learned RL agent can be counter-intuitive to a layperson, hence it requires an explanation graph for correct interpretation.

In fact, *without* an explanation graph, AMT workers are only able to achieve an overall accuracy of 3% on *Q2* (Table 3). We then manually select three frames (i.e., scenarios) from an episode, and test this same set of frames across *all* baseline. We show workers either a graph from CAPS, a graph from [9] (*Topin and Veloso*), a graph from [8] (Zahavy et al.), or no graph, as a baseline. Note that the two baseline graphs do not contain NL explanations. Figure 9 shows the three selected frames for the user-study with distinctive current locations (right, left, bottom) and behaviors (moving slowly, quickly) of the car in the Mountain Car environment. In Fig. 10, given a scenario, AMT workers are asked to identify the abstract state from an explanation graph generated by CAPS. In Fig. 11, given a scenario, AMT workers are asked to identify the next probable action of the agent from an explanation graph generated by CAPS. Figures 12 and 13 show the explanation graphs generated by Topin et al. [9] and Zahavy et al [8], respectively, for the Mountain Car environment that was used in the user-study.

6.1.2 Recruitment and experimental controls

To ensure the quality of the workers, we only recruit those who have (1) HIT approval rate ≥98%, (2) have number of HITs approved ≥1000, and (3) are classified as "master" workers by AMT—i.e., who demonstrated excellent performance for a wide range of tasks in the past. Moreover, we also utilized a count-down timer to ensure a minimum attention span period of 60 and 30 s for *Q1* and *Q2*, respectively. To further motivate the workers to well examine the given tasks, we instructed and gave each worker an additional +50% of the original payment amount for each correct response to either Q1 or Q2, or double payment amount if they correctly answer both questions. *Excluding* the potential bonus, we pay each worker $10–$12/h (v.s. 2021 federal minimum wage of $7.25), which corresponds to a minimum commitment of 1.5 min working time, up to 3 min for both questions. To remove possible bias, we also ensure that the pools of workers across the tasks of different baselines are not overlapped. We maintained the same recruitment criteria and experiment controls as described across all baselines. Our user-study was approved by the Institutional Review Board.

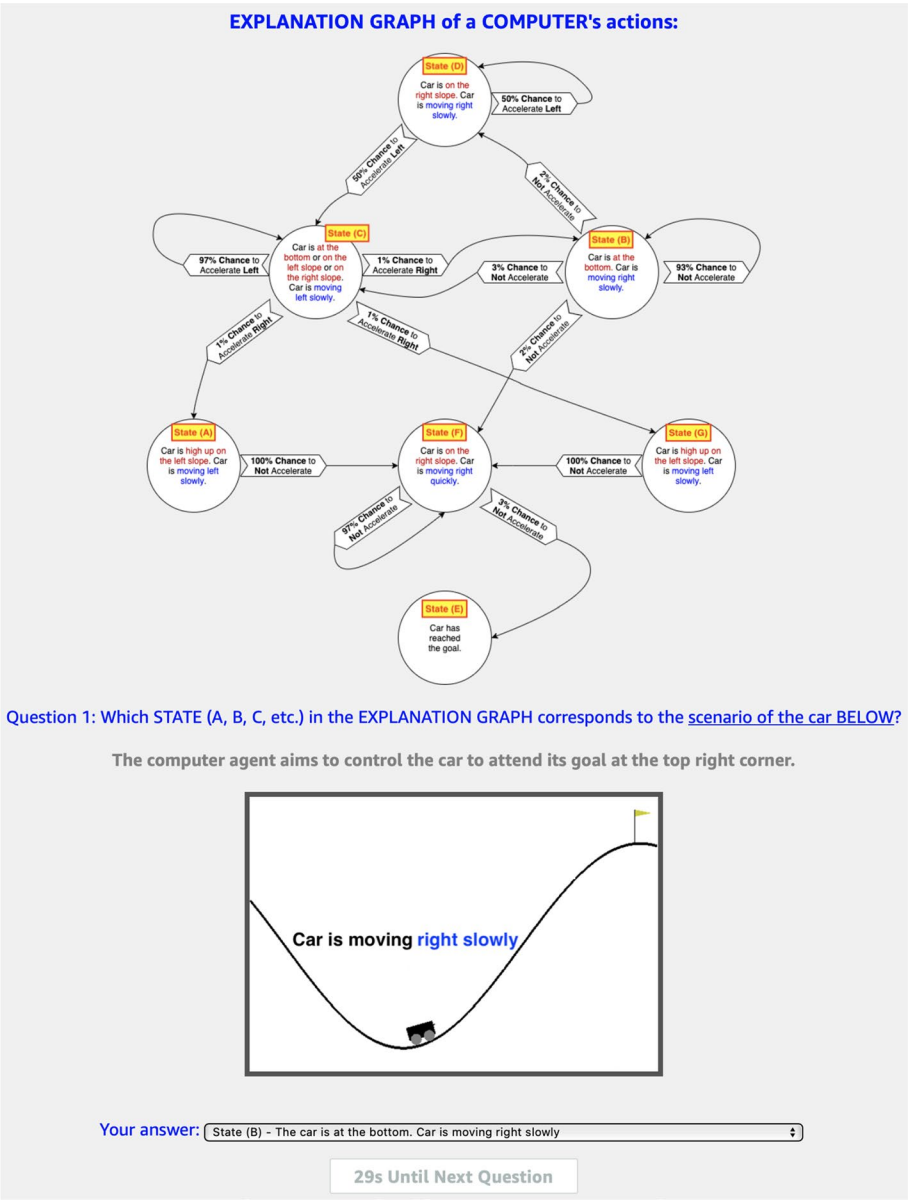


Fig. 10 First question of the user-study interface on AMT on the *Mountain Car* RL environment. The worker is asked to identify the abstract state (in circle shape) from the explanation graph that corresponds with the current scenario of car. A count-down timer is utilized to ensure a *minimum* attention span from the worker before she or he moves to the next question

6.1.3 Results

Table 3 summarizes the results. First, all explanation methods help the end-users to better understand the learned RL agent. Moreover, it is *statistically significant* (p-*value* ≤ 0.01)

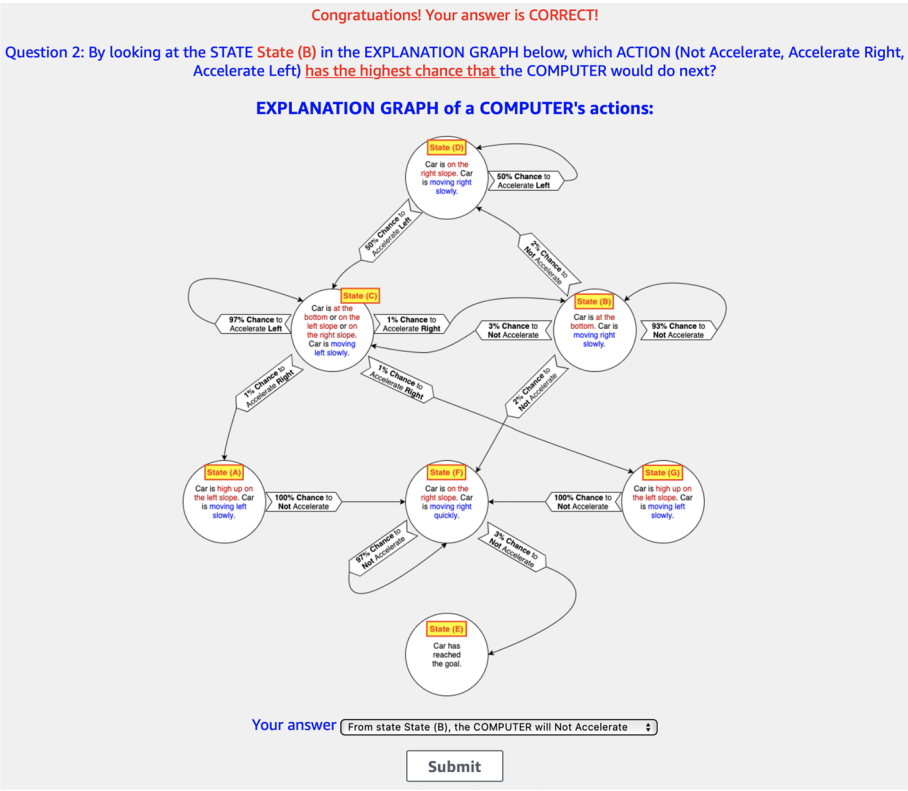


Fig. 11 Second question of the user-study interface on AMT on the *Mountain Car* RL environment. The worker is asked to identify the next probable action that the RL agent would carry out given the current scenario of the car. If the worker incorrectly answers the first question (Fig. 10), they will be informed with the correct abstract state of the explanation graph before answering this question

that CAPS performs better than all baselines on average (Table 4). In fact, our method is the most comprehensible, enabling the end-users to accurately interpret *both* the current states and the agent's next actions (S+A) up to 70% accuracy, a four times improvement from the next best baseline ([8]) (Table 3). Interestingly, except for the *Topin and Veloso* graph, the more time the AMT workers spent on the tasks, the better their responses became. Particularly, the top 20 slowest responses recorded an accuracy of over 85% and 90% accuracy on *Q1* and *Q2*, respectively, when provided with CAPS graph (The detailed results are in Appendix B). Furthermore, adding either NL texts (CAPS) or visual illustrations ([8]) for each abstract state enables the end-users to make decisions more promptly and effectively (The detailed results are in Appendix B).

Given the assumption that the end-users are involved in the *abstract state translation* step of CAPS (Sect. 4.4.1), they should be fully aware of the mapping from a given scenario of the car to its respective abstract state on the CAPS graph. Table 3 shows that those who fit into this assumption—i.e., users who selected the correct abstract state in *Q1*, are the most accurate in interpreting the agent's next actions, with an overall accuracy of

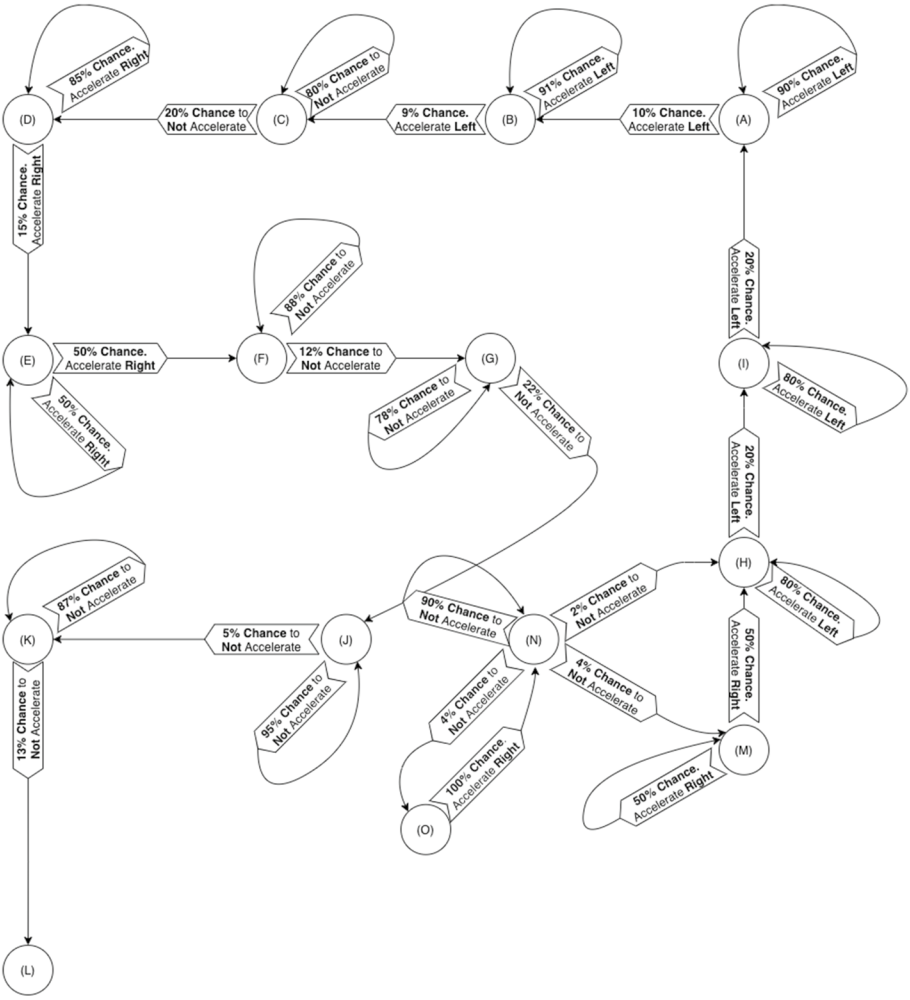


Fig. 12 Explanation graph generated by Topin et al. [9] for the Mountain Car environment that was used in the user-study

around 90% (S→A). Even though this accuracy is 100% in case of *Topin and Veloso*, there was only 1 response that correctly answered *Q1* (Table 3). We also evaluate the trade-off between explainability of a generated CAPS graph, its size, and its cluster policy entropy (Sect. 4.2.1). Figure 14 (Left) shows that the optimal graph size found by CAPS is best positioned in terms of both comprehensibility (the higher the better) and the policy entropy (the lower the better). Once an abstract state is correctly identified, it then becomes relatively easy (≥90% accuracy) for the end-users to correctly identify the agent's next action (Fig. 14, Right).

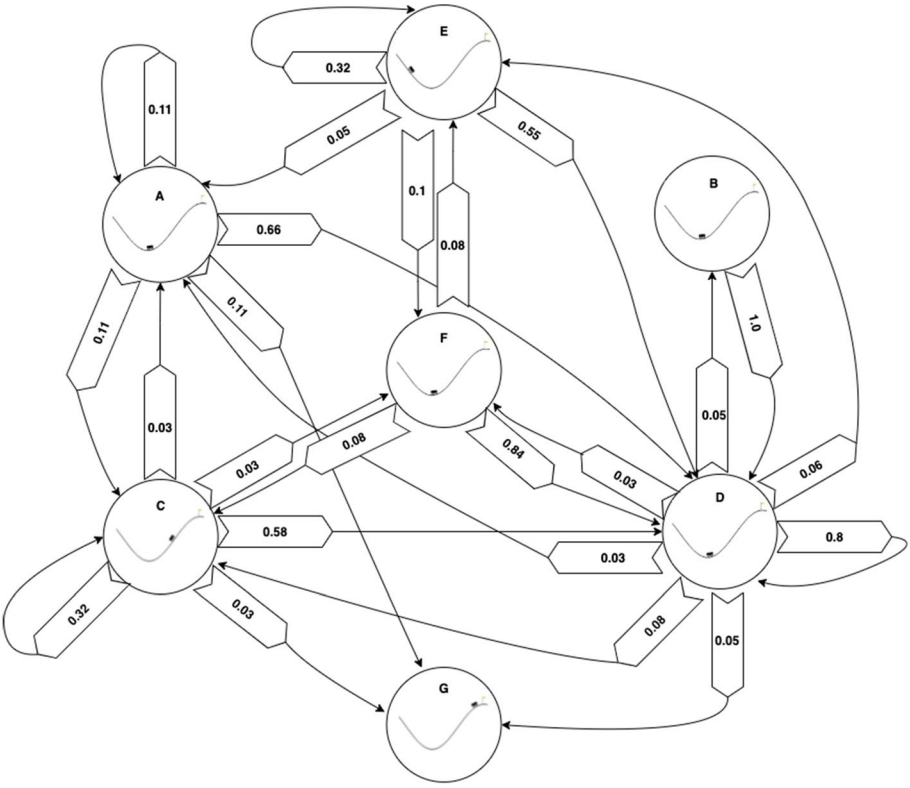


Fig. 13 Explanation graph generated by Zahavy et al. [8] for the Mountain Car environment that was used in the user-study

Table 3 Comparison of user-study results on (*Time*)—the total time spent on the task, accuracy of selecting the (*S*)—correct abstract state, (*A*)—correct action, (*S*+*A*)—correct abstract state and action, (*S*→*A*)—correct action *after* correctly selecting abstract state

Method	#	All responses				
	Ans	Time↓	S↑	A↑	S+A↑	S→A↑
Without graph	33	37 s	–	0.03	–	–
Topin et al. [9]	39	130 s	0.03	0.28	0.03	1.0 (1/1)
Zahavy et al. [8]	45	115 s	0.42	0.47	0.16	0.37 (7/19)
CAPS (Optimal)	46	114 s	0.78	0.87	0.70	0.89 (32/36)

Table 4 *p-values* (all are ≤ 0.01) of hypothesis tests on whether CAPS can provide the AMT workers with additional explanatory values to accurately select *Q1* (S), *Q2* (A) or both (S+A)

	CAPS > NoGraph	CAPS > [9]	CAPS > [8]
p-value (S)	–	1.5e−17	1.6e−4
p-value (A)	2.3e−21	7.8e−10	1.1e−5
p-value (S+A)	–	2.4e−13	1.1e−8

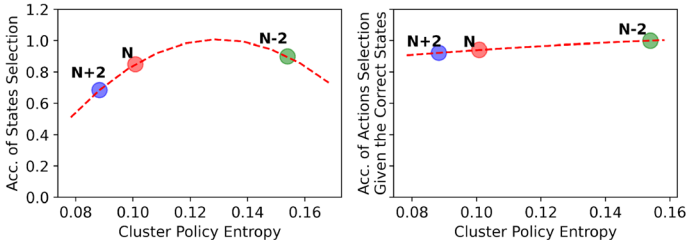


Fig. 14 Relationship between comprehensibility and cluster policy entropy with varied CAPS graph sizes

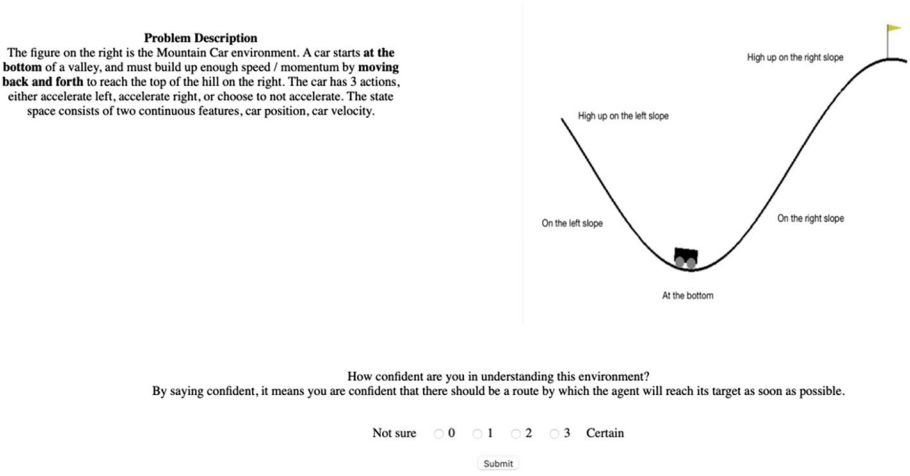


Fig. 15 A screenshot from the first page in the user-query tool providing a description of the Mountain Car environment and measuring the end-user's confidence level of their understanding of the environment

6.2 User Study 2: Counterfactual reasoning

6.2.1 Design

We designed the second user-study to quantitatively measure the extent of that a CAPS graph provides given the counterfactual metrics of timesteps, failure probability and their impact on the end-users' confidence in the agent's behaviors. For this study, we developed a query tool for end-users to interactively examine the RL agent's policy in different regions of the state space in two environments, namely Mountain Car and Gridwold. We shared the link of the tool with 50 undergraduate and graduate students from different disciplines at Wake Forest University. For each environment, we presented a screenshot of the environment to the users along with its description. An example for the Mountain Car environment is shown in Fig. 15. Given an explanation graph, we then asked the users how confident they are in the environment.We later used the answers to this question to subjectively correlate the users' confidence of the agent's behavior and their answers to the rest of the questions in the study (Q1–Q3 below). On the second page of the user study (Fig. 16),

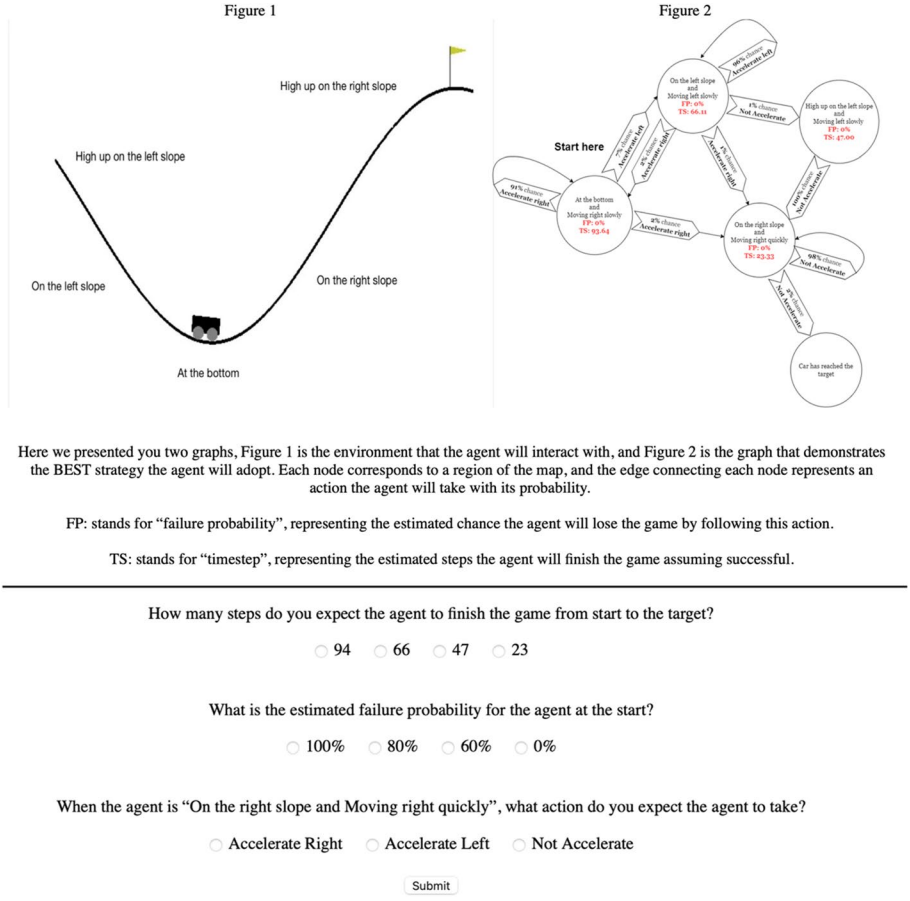


Fig. 16 A screenshot from the second page in the user-query tool providing the CAPS graph for Mountain Car environment and asking three questions about that graph

we presented the users with the CAPS graph of the agent's learned policy for the environment and asked them three questions as follows:

Q1. How many steps do you expect the agent will take to finish the task?
Q2. What is the estimated failure probability for the agent at the start?
Q3. What action do you expect the agent to take at state x (a certain state in the presented CAPS *graph)?.*

Then, we gave the end-user the opportunity to interact with the agent and its CAPS graph crafting new scenarios for the agent. To do this, we ask each user to provide at most three "what if" questions as shown in Fig. 17. Based on the user's question(s), the tool displays the relevant CAPS graph with a question regarding the optimality of the agent's behavior given the scenario crafted by the user. An example is displayed in Fig. 18. If the user doesn't think the displayed behavior is optimal, we then ask another question about "why this behavior is not optimal?" (Fig. 19) with four options related to timesteps and failure

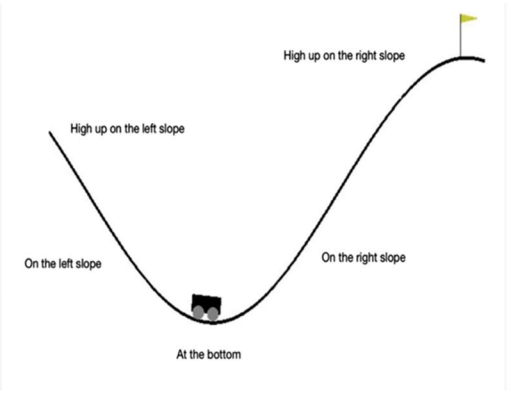


Fig. 17 A screenshot from the user-query tool providing the user the opportunity to ask "what if" questions about the RL agent's behavior in Mountain Car environment

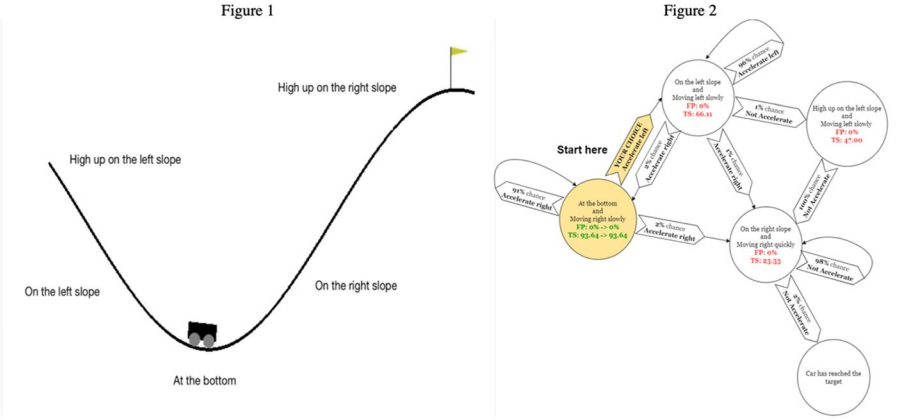


Fig. 18 A screenshot from the user-query tool answering the user's question of "what if" about the RL agent's behavior in Mountain Car environment

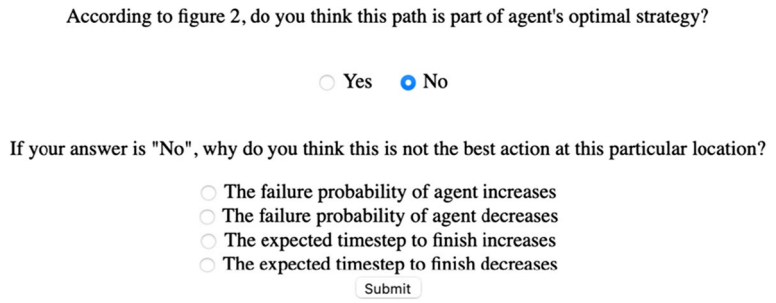


Fig. 19 A screenshot from the user-query tool providing counterfactual explanation in terms of timesteps and failure probability for the user's crafted scenario in Mountain Car environment

probability. Finally, the user study is ended by re-evaluating the user's confidence level in understanding the agent's behavior. This is done by asking a direct, subjective question of "how confident are you in understanding the agent's behavior?". We did the same study for Gridworld environment. The Mountain Car environment is selected because actions of a learned RL agent can be counter-intuitive to a layperson, hence the environment requires an explanation graph for correct interpretation. On the other hand, Gridworld environment is selected for its simplicity to help us evaluate the direct effectiveness of the counterfactual metrics on how well the users understand the agent's behaviors.

6.2.2 Recruitment and experimental controls

For this study, we only recruited college-level students or above from different majors in Wake Forest University. We *made no assumption* regarding their knowledge and experience in either ML or RL. Regarding the incentivies for the students, they entered a poll of three Amazon gift cards of 25 upon the experiment completion. Our user-study was approved by the Institutional Review Board.

6.2.3 Results

We investigate the following research questions in this study:

- **RQ1:** Can counterfactual explanations help human users, who are non-experts in RL,[2] increase their confidence in understanding the RL agent's decision-making? (Subjective measure).
- **RQ2:** Are counterfactual metrics (i.e., timesteps and failure probability) effective for helping users understand an agent's decision-making process? (Objective measure).

Gridworld Results: The users answered RQ1, *subjectively*, by providing ratings to the confidence questions *before* and *after* the study (Fig. 15). Table 5 demonstrates the success

[2] In our study, 74.5% of participants self-evaluated themselves as having either no knowledge about RL at all or had heard about it but do not know any technical details.

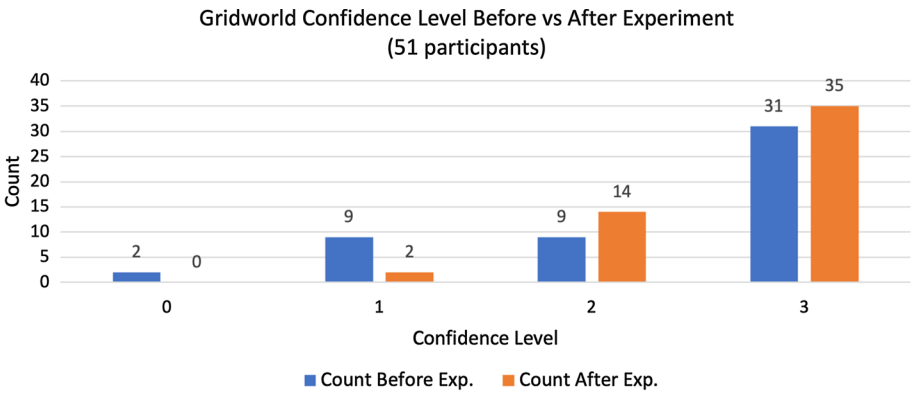


Fig. 20 The change in the users' confidence in their understanding of Gridworld agent (RQ1)

Table 5 More details about the change in the users' confidence in RL agent in Gridworld (RQ1)

Change	Count	Percentage	Q1 Acc	Q2 Acc	Q3 Acc	Asked Qs Acc
Increase	15	29.4%	80.0%	100.0%	100%	96.9%
Decrease	10	19.6%	40.0%	70.0%	90.0%	82.6%
No change	26	51.0%	57.7%	100.0%	96.2%	93.2%
Total/average	51	100.0%	60.8%	94.1%	96.1%	N/A

of CAPS in increasing their understanding of the RL agent in Gridworld environment. We observe that more users fall into confidence levels of 2 and 3 after the study compared to that before the study, and the number of participants who remain in 0 confidence is none and remain in 1 is only 2. Table 5 summarizes such changes in more details. This table shows that around 20.4% of the participants increase their confidence levels. Although the confidence level of 51.0% of the participants remains unchanged, that of the remaining portion (22 of the 26 count) of the participants start with a level-3 confidence which means the counterfactual explanations fail to improve only 8% of the participants' confidence levels (Fig. 20).

To answer RQ1 by objectively measuring the users' understanding of the given explanations, we report the average accuracy of the users' answers to Q1–Q3 as shown in Table 5. The average accuracy among the users is beyond 90% except for Q1 which asks about the timesteps. We believe that the users used the environment illustration to directly count the timesteps rather than retrieving information from the CAPS graph. We propose that if we change the order of Q1 and Q2, the results might get better because the users had to get the failure probability from the CAPS graph which is shown in Q2's accuracy. Another important point is that users with improved confidence tends to have a higher accuracy in answering Q1–Q3. This shows that CAPS improves user's understanding in the environment both *subjectively* and *objectively*.

To evaluate the effectiveness of allowing the users to interactively ask "what if" questions on their understanding of the agent's policy (RQ2), we show the association between the number of the users' questions, their confidence level, and the accuracy of

Table 6 The association between the change in the users' confidence in GridWorld RL agent, the asked questions, and their answers to the user study questions (RQ2)

Initial Conf	Count	Num. of Asked Qs (s.d.)	Avg. Conf. After	Q1 Acc	Q2 Acc	Q3 Acc	Asked Qs Acc
0	2	2.50 (0.50)	2.50	50.0%	100.0%	100.0%	100.0%
1	9	2.00 (0.67)	2.88	100.0%	100.0%	100.0%	100.0%
2	9	1.88 (0.73)	2.33	44.4%	88.9%	88.9%	94.1%
3	31	2.38 (0.82)	2.67	54.8%	93.5%	96.8%	89.2%

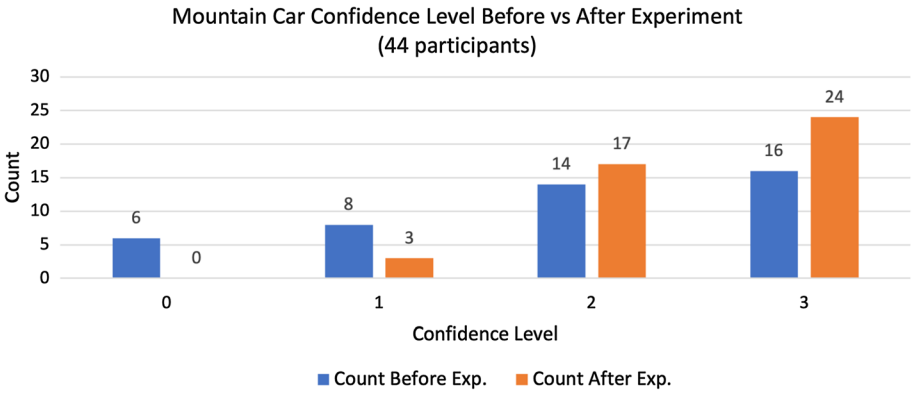


Fig. 21 The change in the users' confidence in their understanding of the Mountain Car agent (RQ1)

Table 7 More details about the change in the users' confidence in RL agent in MountainCar (RQ1)

Change	Count	Percentage	Q1 Acc	Q2 Acc	Q3 Acc	Asked Qs Acc
Increase	21	47.7%	95.2%	100.0%	100.0%	92.9%
Decrease	8	18.2%	75.0%	100.0%	100.0%	83.3%
No change	15	34.1%	60.0%	93.3%	93.3%	88.0%
Total/average	44	100.0%	79.5%	97.7%	97.7%	N/A

their answers to Q1–Q3 in Table 6 and information on the last column of Table 5. Here, we notice a general trend that the lower the initial confidence is, the higher the number of questions the users asked, and higher accuracy in answering their asked question. Except for the initial confidence level of 3, the average final confidence for all other three levels increases. An important takeaway here is that if the users started with relatively high confidence, they tend to think they know how the environment works, therefore they do not rely on the CAPS graph and end up answering Q1 wrong. Table 5 also illustrates that users with increased confidence have a higher accuracy in answering their asked questions.

Table 8 The association between the change in the users' confidence in Mountain Car RL agent, the asked questions, and their answers to the user study questions (RQ2)

Initial Conf	Count	Num. of Asked Qs (s.d.)	Avg. Conf. After	Q1 Acc	Q2 Acc	Q3 Acc	Asked Qs Acc
0	6	2.00 (0.57)	2.50	100.0%	100.0%	100.0%	100.0%
1	8	2.25 (0.43)	2.63	87.5%	100.0%	100.0%	100.0%
2	14	1.50 (0.73)	2.43	71.4%	92.9%	92.9%	92.9%
3	16	1.75 (0.90)	2.44	75.0%	100.0%	100.0%	85.7%

Mountain Car Results: Fig. 21 shows the changes in the end-users confidence level *before* and *after* taking the study. Similar to Gridworld, here we observe that more users fall into confidence levels of 2 and 3 after the study compared to that before the study. However, we find that more users start at a confidence level of 0 and 1, indicating that, compared to Gridworld, MountainCar environment is more difficult to understand. Table 7 summarizes the change in the confidence level of the users in more details. The table shows that around 47.7% (roughly half) of the participants increased their confidence level of understanding the agent's policy compared to 30% in Gridworld. We can conclude that our tool is more effective when the environment is more difficult.

As in Gridworld, the given explanations for the MountainCar agent increased the accuracy of the users' answers to Q1–Q3 by 95% as shown in Table 7. In Table 8, we show the association between the number of the users' questions, their confidence level, and the accuracy of their answers to Q1–Q3 which are similar to the results of Gridworld. The users who started with confidence levels 1 and 2 asked more questions, resulting in higher afterward confidence, and higher question accuracy for all three questions. If the user started with relatively higher confidence, they tend to think they know how the environment works, therefore they do not rely on the CAPS graph and ended up answering Q1 wrong.

Our user studies show that the explainable graphs generated by CAPS help increase the users' trust and understanding of the agent's behavior. We hope that by creating a method for policy-level explanations which does not require much effort, there is more incentive to add explanations to new RL algorithms.

7 Challenges and open questions

In this work, we have shown how policy-level explanations and counterfactual explanations for RL tasks helped end-users improve their understanding of RL agents. However, there are some questions remain unanswered. In this section, we identify main challenges and open research directions on this path that should be tackled.

Stochasticity Most methods for XRL assume that agent's decisions and state space changes are deterministic, and that there exists one true sequence of actions for the optimal policy. In many RL tasks, however, the environments can exhibit stochastic behavior and influence the agent's behavior. Moreover, events outside of the agent's control (i.e., perturbations or noise) can also change the environment, and the agent's actions can result in stochastic consequences. This is common in real-world applications, where an agent's

decisions can change and performing an action might not have a deterministic outcome. Hence, XRL methods should account for stochastic nature of the environment. In this work, we studied one stochastic environment in Blackjack and CAPS was able to generate an explanation graph for this task. However, as can be seen in Fig. 5, the counterfactual explanation using FP was not precisely clear, the probability of the agent to fail can be above zero. We intend to extend our counterfactual explanation method to consider the stochasticity of the environment as future work.

Continuous Environments RL tasks in real-world are often conducted in continuous state and action spaces. Providing explanations for those tasks is challenging due to the explosion of those spaces which affects the readability of the generated explanations either in text or graph format. CAPS can be used in those environments after appropriate discretization of the spaces. We showed an example of such explanations for Mountain Car (Fig. 4). The discretization of the state space in this environment was done through a clustering algorithm to group similar states into abstract states. However, we found it more challenging to explain tasks with continuous action space. On one hand, when the action space is not discretized, it is difficult to map each action to an edge in the graph which will eventually reduce the readability of the graph. On the other hand, when the action space is discretized, it could lead to a significant information loss. One simple solution for small environments is to manually bound the actions to specific ranges such that each action range can be represented in the graph as a single edge. Further research is required to explain environments with continuous action space.

Multi-Agent Environments The interpretation and explainability of RL agent's policy in multi-agent RL (MARL) environments is still relatively unexplored, lacking well-established principles and best practices. We hypothesize that the explainability for agents in MARL environments can be addressed through the following questions:

1. What are the design principles required for explainable methods to provide sufficient explanations of MARL agents' decisions or actions to the end-users?
2. To what extent are the RL policies transparent, i.e. how can decisions or actions be understood in the context of other agents' actions?

The MARL explainable methods should explain the learned policies at the agent level and the team level to increase users' trust in the system.

Evaluation Evaluation of explanations is challenging as how their quality should be perceived is highly subjective and dependent on the user. In supervised learning, explanations are evaluated against out-of-sample data or test sets. For RL tasks, however, the still images miss some important information like velocity which we included in the graph using some descriptive words as "slowly" and "quickly". We realize those descriptive terms are subjective and we plan to extend this work by developing an interactive tool for CAPS including short video clips for the agent in environments similar to Mountain Car. Moreover, there is no established benchmark environments in which explanations could be effectively evaluated. Identifying benchmark environments is necessary for evaluating and comparing explainable methods in RL. Additionally, RL benchmark environments, if available, need to also cover multi-agent, continuous space, and stochastic tasks.

8 Conclusion

We introduce a novel method, named CAPS, for generating comprehensible policy graphs. The generated graphs are designed to explain the policy of an RL agent to an end-user with minimal knowledge of machine learning. We present a state abstraction strategy that gives us control over the abstraction and size of the policy graph. It also allows us to gain additional information by observing how the graph changes as we make the abstraction more or less granular. To make the graph more user-friendly, we also propose a novel method for condensing entire abstract states into concise, natural language descriptions. The counterfactual state explanations are then added to the policy graph in two steps;identifying the critical states in the graph then developing a novel counterfactual explanation method based on action perturbation in those critical states.

We evaluate the effectiveness of CAPS on human participants who are not RL experts in two separate user studies. The first user study investigated the understanding of the end-user of the agent's behavior by answering a set of "what" questions about the agent's actions in different states. The second user study allows the participants to interactively ask counterfactual questions, such as "What if the agent takes action A instead of B?" before measuring their understanding of the agent's behavior. The results showed that when provided with our explanation graphs, end-users were able to accurately interpret policies of trained RL agents 80% of the time, compared to 10% when provided with the next best baseline and 68.2% of users demonstrated an increase in their confidence in understanding an agent's behavior after provided with the counterfactual explanations.

In future, we intend to extend CAPS to the safety RL field. Specifically, we plan to investigate the CAPS's ability to explain the compromised behavior or faulty agents. We have published our initial results in that direction as an abstract here (https://vizsec.org/files/2022/vizsec_p3_abstract.pdf). Moreover, we intend to extend CAPS to MARL environments. We believe CAPS may need to be significantly modified to interpret the RL policies at the agent level and the team level to increase users' trust in the system.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10458-023-09615-8.

Author Contributions TL developed the counterfactual explanation and co-designed the second study. JM developed the initial CAPS without the counterfactual explanation and co-designed the first case study. TL co-designed the user studies and analyzed them. MAR co-designed the user studies and generated the fidelity tests. DL advised the co-authors in the development process and co-wrote the manuscript. SA advised the co-authors in the development process and wrote the manuscript. All authors participated in the brainstorming stage of this work.

Funding The work was in part supported by NSF awards #1950491, #1909702, and #2105007.

Data availability Our code is available for reproducibility in: https://github.com/mccajl/CAPS.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this article.

Ethical approval Our user studies were approved by the Institutional Review Board at Wake Forest University with the number IRB00024657. The approved IRB followed the *Exemption Category 3*: Research involving benign behavioral interventions is conjunction with the collection of information for an adult subject through verbal or written responses (including data entry) or audiovisual recording if the subject prospectively agrees to the intervention and information collected. (A) The information obtained is recorded by the investigator

in such a manner that the identity of the human subjects cannot readily be ascertained, directly or through identifiers linked to the subjects; (B) Any disclosure of the human subjects' responses outside the research would not reasonably place the subjects at risk of criminal liability or be damaging to the subjects' financial standing, employability, educational advancement, or reputation; or (C) The information is recorded by the investigator in such a manner that the identity of the human subjects can readily be ascertained, directly or through identifiers linked to the subjects.

Participant consent You are invited to participate in a research study explaining the behavior of artificial intelligence agents. We are investigating whether summarizing the agent's behavior using English improves the end user's understanding and increases their trust in the agent's behavior. In this study, you will complete several questionnaires to measure your understanding of the agent's behavior. You may discontinue your participation at any time without penalty by closing your browser window. Any responses entered to that point will be deleted. You may also choose not to answer any question(s) you do not wish to answer for any reason. You may choose to skip any question(s) for any reason. We encourage you to print or save a copy of this page for your records (or future reference). By clicking on "I agree", you indicate that you are at least 18 years old and that you agree to participate in this research project. You will advance to the experiment. If you do not wish to participate, please close your browser window. Completing the experiment should take about 15 min. You will earn 20 cents for each minute you spend on the experiment (there will be a time limit for each question) and you will get 10 cents extra for each correct answer.

References

1. Benbrahim, H., & Franklin, J. A. (1997). Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems, 22*, 283–302.
2. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5603.
3. Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Bae, S., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Babu, A., Le, Q.V., Laudon, J., Ho, R., Carpenter, R., & Dean, J. (2020). Chip placement with deep reinforcement learning. https://arxiv.org/pdf/2004.10746.pdf.
4. Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J., & Wang, Y. (2017). A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning, pp. 372–382 https://doi.org/10.1109/ICDCS.2017.123.
5. Peters, J., & Schaal, S. (2006). Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2219–2225). https://doi.org/10.1109/IROS.2006.282564.
6. Huang, S. H., Bhatia, K., Abbeel, P., & Dragan, A. D. (2018). Establishing appropriate trust via critical states. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*.
7. Hayes, B., & Shah, J. (2017). Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE international conference on human–robot interaction (HRI)* (pp. 303–312).
8. Zahavy, T., Ben-Zrihem, N., & Mannor, S. (2016). Graying the black box: Understanding dqns. In M. .F. Balcan & K. .Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning. Proceedings of machine learning research* (Vol. 48, pp. 1899–1908). PMLR.
9. Topin, N., & Veloso, M. (2019). Generation of policy-level explanations for reinforcement learning. In *The thirty-third AAAI conference on artificial intelligence, AAAI 2019* (pp. 2514–2521). AAAI Press. https://aaai.org/ojs/index.php/AAAI/article/view/4097.
10. Olson, M. L., Khanna, R., Neal, L., Li, F., & Wong, W.-K. (2021). Counterfactual state explanations for reinforcement learning agents via generative deep learning. *Artificial Intelligence, 295*, 103455. https://doi.org/10.1016/j.artint.2021.103455
11. Juozapaitis, Z., Koul, A., Fern, A., Erwig, M., & Doshi-Velez, F. (2019) Explainable reinforcement learning via reward decomposition. In *Proceedings at the international joint conference on artificial intelligence*.
12. Aniek Markus, J. K., & Rijnbeek, P. (2021). The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and

evaluation strategies. *Journal of Biomedical Informatics, 113*, 103655. https://doi.org/10.1016/j.jbi.2020.103655
13. Madumal, P., Miller, T., Sonenberg, L., & Vetere, F. (2020). Explainable reinforcement learning through a causal lens. In *The thirty-fourth AAAI conference on artificial intelligence, AAAI 2020* (pp. 2493–2500). AAAI Press. https://aaai.org/ojs/index.php/AAAI/article/view/5631.
14. Liu, B., Xia, Y., & Yu, P. S. (2004). Clustering via decision tree construction. In *Foundations and advances in data mining*. Studies in fuzziness and soft computing (Vol. 180). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11362197_5.
15. Schulman, J., Wolski, F., Radford, A., Dhariwal, P., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
16. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. In arXiv preprint arXiv:1312.5602
17. Liu, Guiliang & Schulte, Oliver & Zhu, Wang & Li, Qingcan, 2018. Toward interpretable deep reinforcement learning with linear model U-trees: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II. https://doi.org/10.1007/978-3-030-10928-8_25.
18. van der Waa, J., van Diggelen, J., van den Bosch, K., & Neerincx, M. (2018). Contrastive explanations for reinforcement learning in terms of expected consequences.
19. Iyer, R. R., Li, Y., Li, H., Lewis, M., Sundar, R., & Sycara, K. P. (2018). Transparency and explanation in deep reinforcement learning neural networks. In *Proceedings of the 2018 AAAI/ACM conference on AI, ethics, and society*.
20. Yang, Z., Bai, S., Zhang, L., & Torr, P. H. S. (2019). Learn to interpret Atari agents.
21. Greydanus, S., Koul, A., Dodge, J., & Fern, A. (2018). Visualizing and understanding Atari agents. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning. Proceedings of machine learning research* (Vol. 80, pp. 1792–1801). PMLR.
22. Amir, D., & Amir, O. (2018) Highlights: Summarizing agent behavior to people. In *AAMAS*.
23. McCalmon, J., Le, T., Alqahtani, S., & Lee, D. (2022) Caps: Comprehensible abstract policy summaries for explaining reinforcement learning agents. In *Proceedings of the 21st international conference on autonomous agents and multiagent systems. AAMAS '22* (pp. 889–897). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC.
24. Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112*, 181–211.
25. Moore, A. (1995). Variable resolution reinforcement learning. CMU-RI-TR-95-19. https://apps.dtic.mil/sti/tr/pdf/ADA311507.pdf.
26. Rodgers, J., & Nicewander, A. (1988). Thirteen ways to look at the correlation coefficient. *American Statistician, 42*, 59–66. https://doi.org/10.1080/00031305.1988.10475524
27. Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell., 267*, 1–38.
28. Goyal, Y., Wu, Z., Ernst, J., Batra, D., Parikh, D., & Lee, S. (2019). Counterfactual visual explanations. In *ICML* (pp. 2376–2384). http://proceedings.mlr.press/v97/goyal19a.html.
29. Uesato, J., Kumar, A., Szepesvari, C., Erez, T., Ruderman, A., Anderson, K., Heess, N., & Kohli, P. (2018). Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. arXiv. https://doi.org/10.48550/ARXIV.1812.01647. arXiv:1812.01647.
30. Abolfathi, E. A., Luo, J., Yadmellat, P., & Rezaee, K. (2021). Coachnet: An adversarial sampling approach for reinforcement learning. arXiv preprint arXiv:2101.02649.
31. van der Waa, J., van Diggelen, J., Bosch, K. V. D., & Mark, N. (2018). Contrastive explanations for reinforcement learning in terms of expected consequences. https://doi.org/10.48550/ARXIV.1807.08706.
32. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.

Authors and Affiliations

Tongtong Liu[1] · Joe McCalmon[1] · Thai Le[2] · Md Asifur Rahman[1] · Dongwon Lee[3] · Sarra Alqahtani[1]

✉ Sarra Alqahtani
alqahtas@wfu.edu

Tongtong Liu
liut18@wfu.edu

Joe McCalmon
mccalmonjoe@gmail.com

Thai Le
thaile@olemiss.edu

Md Asifur Rahman
rahmm21@wfu.edu

Dongwon Lee
dongwon@psu.edu

[1] Computer Science Department, Wake Forest Univeristy, 1834 Wake Forest Rd, Winston-Salem, NC 27109, USA

[2] Computer and Information Science Department, University of Mississippi, 1848 University, Oxford, MS 38677, USA

[3] College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, USA