Learning Performance Graphs From Demonstrations via Task-Based Evaluations

Aniruddh G. Puranic, *Graduate Student Member, IEEE*, Jyotirmoy V. Deshmukh, and Stefanos Nikolaidis

***Abstract*—In the paradigm of robot learning-from-demonstra tions (LfD), understanding and evaluating the demonstrated behaviors plays a critical role in extracting control policies for robots. Without this knowledge, a robot may infer incorrect reward functions that lead to undesirable or unsafe control policies. Prior work has used temporal logic specifications, manually ranked by human experts based on their importance, to learn reward functions from imperfect/suboptimal demonstrations. To overcome reliance on expert rankings, we propose a novel algorithm that learns from demonstrations, a partial ordering of provided specifications in the form of a performance graph. Through various experiments, including simulation of industrial mobile robots, we show that extracting reward functions with the learned graph results in robot policies similar to those generated with the manually specified orderings. We also show in a user study that the learned orderings match the orderings or rankings by participants for demonstrations in a simulated driving domain. These results show that we can accurately evaluate demonstrations with respect to provided task specifications from a small set of imperfect data with minimal expert input.**

***Index Terms*—Formal methods in robotics and automation, learning from demonstration, reinforcement learning.**

I. Introduction

In HUMAN-ROBOT interaction, understanding the behaviors exhibited by humans and robots plays a key role in robot learning, improving task efficiency, collaboration and mutual trust [1], [2], [3], [4]. Demonstrated behaviors can be evaluated by specifying a cumulative reward function that assigns behaviors a numeric value; the implicit assumption here is that higher cumulative rewards indicate good behaviors. Such cumulative rewards are then used with reinforcement learning (RL) to learn an optimal policy. Poorly-designed reward functions can pose the serious risk of reward hacking where the agent behavior maximizing the cumulative reward is undesirable or unsafe [5]. An alternative method is the use of high-level task descriptions using logical specifications (rather than manually defined rewards), and approaches that use task descriptions expressed in temporal logic are quite popular in robotics [6], [7], [8], [9], [10]. Of particular relevance to this letter is the use of the specification language Signal Temporal Logic (STL) [11]. Recent work develops the LfD-STL framework [12], [13]; this framework uses STL specifications to infer rewards from user-demonstrations (some of which could be suboptimal or unsafe) and has shown to outperform inverse reinforcement learning (IRL) methods [14], [15], [16] in terms of number of demonstrations required and the quality of rewards learned.

In the LfD-STL framework, STL specifications describe desired objectives, and demonstrations show how to achieve these objectives. For example, in an autonomous driving scenario, an STL specification could describe the task “reach the goal while avoiding obstacles”. STL is equipped with quantitative semantics that indicate how well the demonstrations performed on the tasks, which are used to infer rewards. The inferred rewards guide the robotic agent towards policies that produce desirable behaviors. Typically, robotic systems are difficult to characterize using a single specification, and users may thus seek for policies that satisfy several task specifications. Not all specifications may be equally important; for example, a hard safety constraint is more important than a performance objective. LfD-STL permits users to thus provide several specifications, but also requires them to manually specify their preferences or priorities on specifications. These preferences are then encoded in a directed acyclic graph (DAG) that, in this letter, we call the *performance graph*. Based on a given performance graph and the quantitative semantics of each STL specification, the LfD-STL framework then defines a state-based reward function that is used with off-the-shelf RL methods to extract a policy. This framework offers few crucial advantages: (i) STL allows defining non-Markovian rewards useful for sequential, patrolling, and reactive tasks, (ii) applications to high-dimensional, continuous and stochastic spaces, and (iii) has empirically demonstrated significant reductions in sample complexity for learning.

A key limitation of the LfD-STL framework is that the onus on providing the performance graph is on the user, which becomes infeasible when there are numerous task specifications. Moreover, there could exist multiple ways of performing a task and a challenge in LfD is whom the agent should imitate, i.e., disambiguate the demonstrations. We propose solutions to these problems by using the STL specifications and quantitative semantics to not only evaluate the performance of demonstrations, but also to use the evaluations to infer the performance graph. We propose the Performance-Graph Learning (PeGLearn) algorithm that systematically constructs DAGs from demonstrations and STL evaluations, resulting in a representation that can be used to explain the performance of provided demonstrations. A

Manuscript received 10 August 2022; accepted 14 November 2022. Date of publication 1 December 2022; date of current version 8 December 2022. This letter was recommended for publication by Associate Editor Y. Zhao and Editor L. Pallottino upon evaluation of the reviewers’ comments. This work was supported in part by NSF under Grants CCF-2048094, CNS-1932620, and CCF-1837131, and in part by the Toyota R&D. *(Corresponding author: Aniruddh G. Puranic.)*

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by University of Southern California Institutional Review Board.

The authors are with the Computer Science Department, University of Southern California, Los Angeles, CA 90089 USA (e-mail: puranic@usc.edu; jdeshmuk@usc.edu; nikolaid@usc.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3226072, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3226072

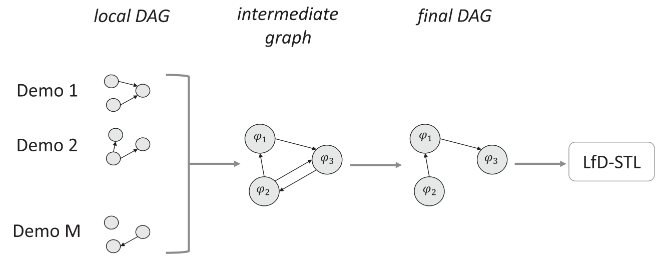


Fig. 1. Overview of PeGLearn algorithm.

high-level overview of this algorithm used in conjunction with prior LfD framework is shown in Fig. 1, which we discuss in detail in Section III-B.

In complex environments where it is non-trivial to express tasks in STL, we use human annotations (ratings or scores) of the data. Examples of complex tasks in human-robot interactions can include descriptions like "tying a knot" or having "fluency in motion" in robotic surgery, where even experts can struggle to express the task in formal logic, or specifying the task formally may not be expressible in a convenient logical formalism. In our setting, rating scales can replace temporal logics by (i) choosing queries that assess performance and (ii) treating the ratings/scores as quantitative assessments[1]. There is precedence of such quantitative assessments; for example, $Likert$ ratings from humans are used as ground-truth measurements of trust [18]. We perform several simulation experiments in autonomous driving and complex navigation of an industrial mobile robot (MiR100) to validate our approach. We also discuss the complexity of PeGLearn and draw comparisons with existing learning techniques.

II. Background

In this section, we provide various definitions and notations used in our methodology and experiments.

Definition 1 (Demonstration): A demonstration is a finite sequence of state-action pairs in an environment that is composed of a state-space S and action-space A that can be performed by the agent. Formally, a demonstration ξ of finite length $L \in \mathbb{N}$ is $\xi = \langle (s_1, a_1), (s_2, a_2), \ldots, (s_L, a_L) \rangle$, where $s_i \in S$ and $a_i \in A$. That is, ξ is an element of $(S \times A)^L$. We interchangeably refer to demonstrations as trajectories.

Each environment is governed by tasks or objectives, which we refer to as specifications or requirements, denoted by φ. In this regard, we define a rating function via Definition 2.

Definition 2 (Rating Function): A *rating function* $\mathcal{R}$ is a real-valued function that maps a specification and a time-series data or trajectory to a real number, i.e., $\mathcal{R} : \Phi \times \Xi \rightarrow \mathbb{R}$, where, Φ and Ξ are each infinite sets of specifications and demonstrations, respectively.

Intuitively, the rating function describes how "well" the specifications are met (satisfied) by a trajectory. The rating function can be obtained via the quantitative (robustness) semantics in temporal logics [19] or human ratings via surveys, annotations, etc. It indicates the score or signed distance of the time-series data to the set of temporal data satisfying a specification. For a given specification φ and a demonstration ξ, the *rating* (also referred to as *evaluation* or *score*) of ξ with respect to φ is denoted by $\rho = \mathcal{R}(\varphi, \xi)$. This ρ is negative if ξ violates φ, and non-negative otherwise.

Definition 3 (Directed Acyclic Graph): A directed acyclic graph or DAG is an ordered pair $G = (V, E)$ where V is a set of elements called vertices or nodes and E is a set of ordered pairs of vertices called edges or arcs. An edge $e = (u, v)$ is directed from a vertex u to another vertex v.

A path $x \rightsquigarrow y$ in G is a set of vertices starting from vertex x and ending at vertex y by following the directed edges from x. Each vertex $v \in V$ is associated with a real number - weight of the vertex, represented by $w(v)$. Similarly, each edge $(u, v) \in E$ is associated with a real number - weight of the edge and is represented by $w(u, v)$. Notice the difference in the number of arguments in the notations of vertex and edge weights. Similar to prior work [12], [13], we define each node $v \in V$ of some DAG, G, to represent a task specification.

III. Problem Definition and Methodology

A. Problem Formulation

To accomplish a set of tasks, we are given: (i) a finite dataset of m demonstrations $\Xi = \{\xi_1, \xi_2, \ldots, \xi_m\}$ in an environment, where each demonstration is defined as in Definition 1, (ii) a finite set of n specifications $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ to express the high-level tasks and by which a set of scores for each demonstration evaluated on each of the n specifications $\boldsymbol{\rho_\xi} = [\rho_1, \ldots, \rho_{|\Phi|}]^T$ is obtained. We can then represent this as an $m \times n$ matrix $\mathcal{Z}$ where each row i represents a demonstration and a column j represents a specification. An element ρ_{ij} indicates the rating or score of demonstration i for specification j, i.e., $\rho_{ij} = \mathcal{R}(\varphi_j, \xi_i)$.

$$\mathcal{Z} = \begin{bmatrix} \rho_{11} & \rho_{12} & \cdots & \rho_{1n} \\ \rho_{21} & \rho_{22} & \cdots & \rho_{2n} \\ \vdots & \cdots & \ddots & \vdots \\ \rho_{m1} & \rho_{m2} & \cdots & \rho_{mn} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\rho_{\xi_1}^T} \\ \boldsymbol{\rho_{\xi_2}^T} \\ \vdots \\ \boldsymbol{\rho_{\xi_m}^T} \end{bmatrix} \tag{1}$$

As in prior work on LfD-STL, we need to compute a cumulative score or rating r_ξ for each demonstration to collectively represent its individual specification scores, and so we have an $m \times 1$ vector $\mathbf{r} = [r_{\xi_1}, r_{\xi_2}, \ldots, r_{\xi_m}]^T$. To obtain the cumulative scores, we require a scalar quantity or *weight* associated with each specification indicating its (relative) priority/preference/importance over other specifications. We thus have a weight vector $\mathbf{w} = [w_1, w_2, \ldots, w_{|\Phi|}]^T$ from which we can obtain the cumulative scores as $\mathcal{Z} \cdot \mathbf{w} = \mathbf{r}$. In other words, for each demonstration ξ, $r_\xi = \boldsymbol{\rho_\xi}^T \cdot \mathbf{w}$. The objective is to compute both $\mathbf{w}$ and $\mathbf{r}$, given only $\mathcal{Z}$, such that the "better" demonstrations have higher cumulative scores than others and ranked appropriately by the LfD-STL framework, i.e., generate *partial order*; this is an unsupervised learning problem. One of the approaches to computing $\mathbf{w}$ proposed in [12], [13] required the demonstrators to specify their preferences encoded as a DAG and the weights were computed via (2)[2].

$$w(\varphi) = |\Phi| - |ancestor(\varphi)| \tag{2}$$

[1] Here we assume that $Likert$ scales are interval scales [17].

[2] Before robustness values of various temporal logic specifications are combined, they are normalized using tanh or piece-wise linear functions.

where, $ancestor(v) = \{u \mid u \rightsquigarrow v, u \in V\}$, i.e., the ancestors of a vertex v is the set of all vertices in G that have a path to v. However, this is not data-driven as it requires human inputs to define the weights and is only feasible if the number of specifications is small. To overcome this, we can rely on data-driven approaches such as unsupervised learning. In our experiments, we show how existing methods are inefficient, and we thus propose a new approach by learning a DAG directly from demonstrations (i.e., without human inputs) and using (2) to compute weights for the LfD-STL framework to extract rewards for RL. The DAG contains the elements of Φ as its vertices, and the relative differences in performance between specifications as edges. We refer to this as the Performance-Graph since it captures the performance of the demonstrations w.r.t. the task specifications. This final graph is required to be acyclic so that topological sorting can be performed on the graph to obtain an ordering of the nodes and hence specifications, i.e., topological ordering does not apply when there are cycles in the graph.

An assumption that we make is that, at least 1 demonstration satisfies all the specifications of Φ, but does not have to be optimal (i.e., having the highest rating) w.r.t. those specifications; we argue that this is a reasonable assumption to make compared to related LfD works like IRL that require a large sample size of nearly-optimal (i.e., close to the highest rating) demonstrations and also to show that the task(s) can be realized, even if suboptimally, under the given specifications. The last assumption is that the demonstrators' intentions are accurately reflected (in terms of performance) in the demonstrations provided, therefore we consider each demonstration equally important when inferring graphs.

B. Rating-Based Graph

In this section, we describe the procedure to create the Performance-Graph from ratings or scores obtained either automatically by temporal logics or provided by human annotators. This process involves 3 main steps:

1) Constructing a *local* weighted-DAG for each demonstration based on its individual specification scores.
2) Combining the *local* graphs into a single weighted directed graph, which is not necessarily acyclic as it can contain bidirectional edges between nodes.
3) Converting the resultant graph into a weighted DAG.

The framework in Fig. 1 depicts the 3 steps described above and the final stage where the inferred DAG is fed to the LfD-STL framework to learn rewards and perform RL.

C. Generating Local Graphs

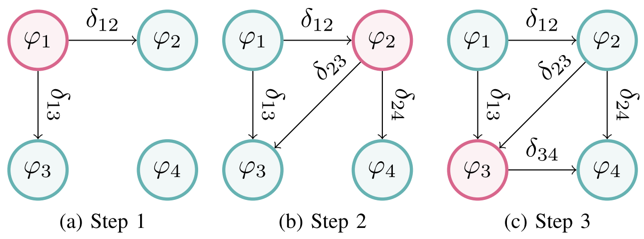
Each demonstration $\xi \in \Xi$ is associated with a vector of ratings $\boldsymbol{\rho_\xi} = [\rho_1, \ldots, \rho_{|\Phi|}]^T$, and the objective is to construct a weighted DAG for ξ from these evaluations. We propose the novel Algorithm 1, where, initially, the evaluations are sorted in non-increasing order with ties broken arbitrarily (lines 3–5). This creates a partial ordering based on the performance of the demonstrations regarding each specification, and is represented by a DAG to capture the partial ordering. Though DAGs can be represented by either adjacency lists or adjacency matrices, in this work, we represent them using adjacency matrices for notational convenience.

Algorithm 1: Algorithm to Compute Local DAG for a Single Demonstration.

```
Input: $\xi$ := a demonstration of any length $L$; $\Phi$ := set
       of $n$ specifications; $\epsilon$: threshold (tunable)
Result: Constructs local Performance-Graph $G_\xi$
1  begin
2      $G_\xi \leftarrow \mathbf{0_{n \times n}}$                  // zero matrix
3      $\mathcal{S} \leftarrow \emptyset$          // Create an empty queue
4      for $j = 1$ to $n$ do
5          Obtain the rating or score $s_j$ for specification
            $j$; $\mathcal{S}.insert(\langle j, s_j \rangle)$
       // Resulting $S$ is an $n \times 2$ matrix
          where each row is $\langle index, score \rangle$
6      $S' \leftarrow$ sort $\mathcal{S}$ in non-increasing order of scores
        // original indices are recorded
7      for $k = 1$ to $n - 1$ do // no self-loops
8          $\varphi \leftarrow S'[k, 1]$                  // get index
9          $v \leftarrow S'[k, 2]$                  // get score
10         for $j = k + 1$ to $n$ do
11             $\varphi' \leftarrow S'[j, 1]$
12             $v' \leftarrow S'[j, 2]$
13             if $(v - v') \geq \epsilon$ then
14                 $G_\xi[\varphi, \varphi'] \leftarrow G_\xi[\varphi, \varphi'] + v - v'$
15     return $G_\xi$
```

Consider 4 specifications $\varphi_i; i \in \{1, 2, 3, 4\}$. Let a demonstration, say $\xi \in \Xi$ have evaluations $\boldsymbol{\rho_\xi} = [\rho_1, \rho_2, \rho_3, \rho_4]$ and without loss of generality, let them already be sorted in non-increasing values, i.e., $\rho_i \geq \rho_j; \forall i < j$. This sorting is performed in the first `for` loop of Algorithm 1. Recall that each node of the DAG represents a specification of Φ, i.e., a node contains the index of the specification it represents. An edge between two nodes φ_i and φ_j is created when the difference between their corresponding evaluations is greater than a small threshold value (lines 6–14). This edge represents the relative rating or performance difference between the specifications and creates a partial order indicating this difference. The threshold ϵ acts as a high-pass filter and can be tuned depending on the normalization of ratings. The intuition is that demonstrations having similar states or features will have similar evaluations for the specifications, and should produce the same partial ordering of specifications. That is, an edge is created if the evaluations differ greatly, e.g., two specifications producing ratings say, 1.0 and 0.99, are numerically different, but have similar performance, so they should be equally ranked (have no edge between them). Without this filter, an edge with a very small weight would be created between them, thereby inadvertently distinguishing similar performances. Formally, $e(\varphi_i, \varphi_j)$ is added when $\delta_{ij} = (\rho(\varphi_i) - \rho(\varphi_j)) \geq \epsilon$.[3] We repeat this process for each node in the DAG (Fig. 2) and the resultant DAG will have at most $n(n-1)/2$ edges, where $n = |\Phi|$. The *local* graph is acyclic, because the nodes are sorted by their respective evaluations in a non-increasing order and hence edges with negative weights will not be added thereby eliminating any bidirectional edges. The DAG for a demonstration imposes a partial order over all

[3] The demonstration ξ argument in ρ was dropped for convenience since we are only considering 1 demonstration at a time.



(a) Step 1 (b) Step 2 (c) Step 3

Fig. 2. Example *local* graph for a demonstration.

specifications. For any 2 specifications φ_i and φ_j, $\varphi_i \succeq \varphi_j$ if $\rho(\varphi_i) \geq \rho(\varphi_j)$ and so, an edge is created from φ_i to φ_j with weight $\rho(\varphi_i) - \rho(\varphi_j)$ subject to the threshold ϵ.

Complexity Analysis: In general, given n specifications and a set of algebraic operators (e.g., $op = \{>, =\}$), the number of different orderings is: $n! \cdot [|op|^{n-1} - 1] + 1$. In our case, $|op| = 2$ since the operator $<$ in an ordering is equivalent to a permutation of the ordering using $>$, i.e., $a < b \equiv b > a$. By making use of directed graphs, we can eliminate the factorial component (refer to our supplemental document [19] for proofs), but this still results in an exponential-time search algorithm. To overcome this, in our algorithm, we eliminate cycles by building a DAG for each of the m demonstrations. Depending on the data structure used, the complexity of building a DAG is linear when using adjacency lists and quadratic when using adjacency matrix to represent the graph. The total complexity is thus $\mathcal{O}(mn^2)$ in the worst case (using matrix representation).

D. Aggregation of Local Graphs

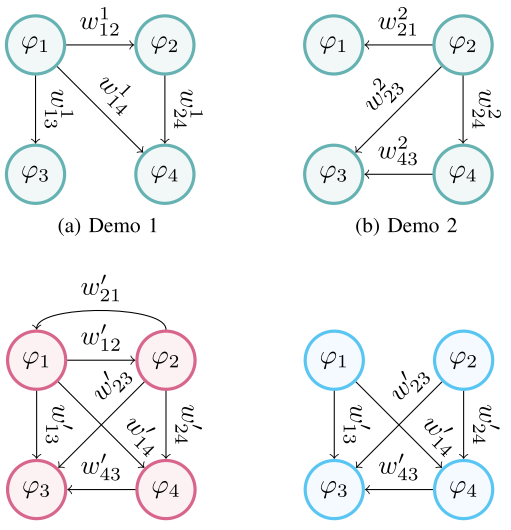
Once the *local* graphs for each demonstration have been generated, they need to be combined into a single DAG to be used directly in the LfD-STL framework [12], [13]. We now propose Algorithm 2 to aggregate all *local* graphs into a single DAG. Line 2 generates the *local* graphs via Algorithm 1 and stores them in a dataset $\mathcal{G}$. For every directed edge between any pairs of vertices u and v, the mean of the weights on corresponding edges across all graphs in $\mathcal{G}$ is computed (line 3 of Algorithm 2). For example, consider the *local* graphs of 2 sample demonstrations shown in Fig. 3. By averaging the edge weights of the graphs of the 2 demonstrations, we get the intermediate weighted directed graph shown in Fig. 3(b). This is not necessarily acyclic since there is a cycle between the nodes of φ_1 and φ_2. In this figure, each $w'_{ij} = (w^1_{ij} + w^2_{ij})/2$. This intermediate graph needs to be further reduced to a weighted DAG, i.e., by eliminating any cycles/loops.

E. Conversion/Reduction to Weighted DAG

Note that there can only be at most 2 edges between any pair of vertices since the outgoing (and similarly, incoming) edges are averaged into a single edge. In order to reduce this graph to a global DAG, we systematically eliminate edges by first computing the difference between the outgoing and incoming edge and then checking if it is above a certain threshold to add an edge in the direction of positive difference (note that if the difference is negative, the edge can be simply reversed). In other words, for any 2 nodes, u and v, if $(w(u,v) - w(v,u)) \geq \epsilon$, then $e(u,v)$ is retained with new weight $w(u,v) - w(v,u)$, while $e(v,u)$ is removed or discarded since it gets absorbed by the retained edge. The threshold ϵ again acts as a high-pass filter. As we can observe in the case of bidirectional edges, one of the edges will be "consumed" by the other or both will be discarded if they are similar. This conversion procedure is shown by lines 5–8 in Algorithm 2. Thus, all cycles/loops are eliminated, resulting in a weighted DAG that can be directly used to rank the demonstrations and compute rewards for RL tasks as performed in the LfD-STL framework. To show that our DAG-learning method indeed preserves the performance ranking over demonstrations, we first define a partial ordering over demonstrations: for any 2 demonstrations ξ_1 and ξ_2, the partial order $\xi_1 \preceq \xi_2$ is defined when $\rho_{1i} \leq \rho_{2i}, \forall i \in \{1, \ldots, n\}$. Thus, we say that ξ_2 is better or at least as good as ξ_1. Then, by making use of Lemma 1, we arrive at Theorem 1 that addresses the problem definition (all proofs are in [19]).

Lemma 1: For a DAG, the weights associated with the nodes computed via (2) are non-negative.

Theorem 1: For any two demonstrations ξ_1 and ξ_2 in an environment, the partial ordering $\xi_1 \preceq \xi_2$ is preserved by PeGLearn.



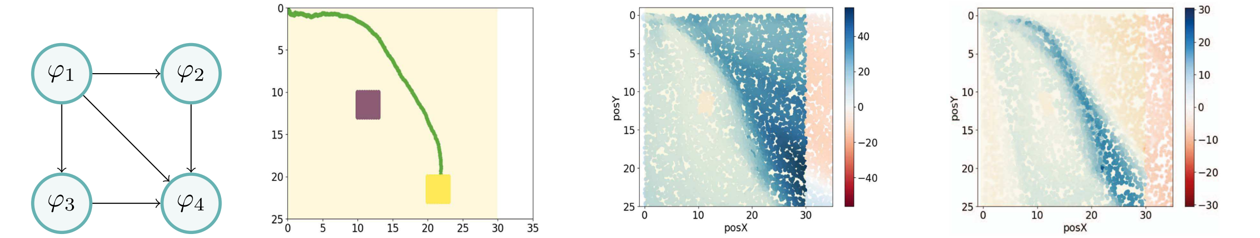
(a) Demo 1 (b) Demo 2

(c) Interim Graph (d) Final DAG

Fig. 3. Example *global* graph from 2 demonstrations.

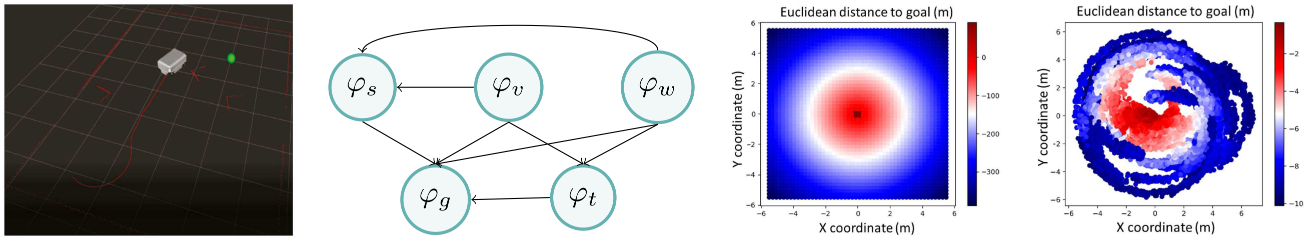
Algorithm 2: PeGLearn: Generating the Global DAG From All Demonstrations.

```
  Input: $D$ := set of $m$ demonstrations; $\Phi$ := set of $n$
         specifications; $\epsilon$: threshold (tunable)
  Result: Constructs global Performance-Graph
1 begin
2     $\mathcal{G} \leftarrow \bigcup_{i=1}^{m} G_i$            // via algorithm 1
3     $G' \leftarrow \frac{1}{|\mathcal{G}|}\sum_{G \in \mathcal{G}} G$        // Edge-wise mean
      // Extracts edge-weighted DAG from
         the raw Performance-Graph
4     $\mathbf{G} \leftarrow \mathbf{0_{n \times n}}$                  // zero matrix
5     for $i = 1$ to $n$ do
6         for $j = 1$ to $n$ do
7             $\mathbf{G}[i,j] \leftarrow max(0, G'[i,j] - G'[j,i])$
8             if $\mathbf{G}[i,j] < \epsilon$ then $\mathbf{G}[i,j] \leftarrow 0$
9     return G
```



(a) Learned DAG (b) Driving layout (c) Baseline rewards (d) PeGLearn rewards

Fig. 4. Results for the 2-D autonomous driving simulator. Baseline rewards are from user-defined DAG.



(a) Environment setup (b) Learned DAG (c) Expert rewards (d) PeGLearn rewards

Fig. 5. Results for the MiR100 navigation environment.

The global DAG imposes a partial order of specifications. For any 2 specifications φ_i and φ_j, the partial order $\varphi_i \succeq \varphi_j$ is defined when $\bar{\rho}(\varphi_i) \geq \bar{\rho}(\varphi_j)$, where $\bar{\rho}(\varphi)$ is the mean of $\rho(\varphi)$ across all demonstrations. The *global* graph thus uses a holistic approach to explain the overall performance of demonstrations and could provide an intuitive representation for non-expert users to teach agents to do tasks as well as understand the policies the agent is learning.

IV. Experiments

A. Comparison With Baseline

For comparison with user-defined DAG in the LfD-STL baseline [12], [13], we evaluated our method on the same discrete-world and 2D autonomous driving domains using the same demonstrations ($m = 8$) and specifications. We show the results for the 2D driving scenario in Fig. 4. The STL specifications correspond to (i) reaching the goal φ_1, (ii) avoiding the hindrance/obstacle regions φ_2, (iii) always staying within the workspace region φ_3, and (iv) reaching the goal as fast as possible φ_4. This resembles a real-world scenario wherein, one of the challenging problems in autonomous driving is overtaking moving or stationary/parked vehicles on road-sides (e.g., urban and residential driving). The scenario presented here is a high-level abstraction where the purple square is a parked car and the yellow square is the goal state of the ego car after overtaking the parked car. The light-yellow shaded region are the dimensions of the road/lane and the task for the ego car is to navigate around the parked car to the goal state without exiting the lane. From these figures, we can observe that the rewards using the inferred DAG are consistent with the specifications, i.e., rewards are aligned with entity locations. In the discrete-world settings, we were able to learn similar graphs and rewards, and hence *same policies* as the baseline. This shows that our proposed method is a significant improvement over the prior work since it eliminates the burden of the user to define graphs, while also using at least *4 times* fewer demonstrations than IRL-based methods [14], [20].

B. Industrial Mobile Robot Navigation

In this setup, we consider a high-fidelity simulator [21] based on the mobile robot *MiR100* that is widely used in today's industries[4]. In this environment, the robot is tasked with navigating to a goal location while avoiding 3 obstacles (Fig. 5). However, the locations of the robot, goal and 3 obstacles are randomly initialized for every episode. This presents a major challenge for LfD algorithms since the demonstrations collected on this environment are unique to a particular configuration of the entity locations (i.e., no two demonstrations are the same). The 20-dimensional state-space of the robot, indexed by timestep t consists of the position of the goal in the robot's frame in polar coordinates (radial distance p_t and orientation θ_t), linear (v_t) and angular (w_t) velocities, and 16 readings from the robot's Lidar for detecting obstacles ($x_t^i, i \in [1, 16]$). We obtained 30 demonstrations, of which 15 were incomplete (i.e., collided with obstacle or failed to reach the goal in time) by training an RL agent on an expert reward function and recording trajectories at different training intervals. The specifications governing this environment are:

1) *Eventually reach the goal:* $\varphi_g := \mathbf{F}(p_t \leq \delta)$, where δ is a small threshold set by the environment to determine if the robot is sufficiently close to the goal to terminate the episode.
2) *Always maintain linear and angular velocity limits:* $\varphi_v := \mathbf{G}(v_{\min} \leq v_t \leq v_{\max})$, and $\varphi_w := \mathbf{G}(w_{\min} \leq w_t \leq w_{\max})$. The limits conform to the robot's capabilities.

[4] This letter is accompanied by a multimedia (video) for the experiments.

3) *Always avoid obstacles:* $\varphi_s := \mathbf{G}(\bigwedge_{i=1}^{16}(x_t^i > 0))$, where x_i^t is the distance from an obstacle as measured by Lidar i.
4) *Reach the goal in a reasonable time:* $\varphi_t := \mathbf{F}(T_{\min} \leq t \leq T_{\max})$, where the time limits are obtained from the average lengths of *good* demonstrations observed in this environment.

Once the graph is extracted via PeGLearn, the rewards are propagated to the observed states and modeled with a neural network via the method described in [13]. We used a 2-hidden layer neural network with 512 nodes in each layer for reward approximation. The DAG and rewards inferred are shown in Fig. 5. We can observe that the rewards are semantically consistent with the specifications, in that, rewards increase as the robot moves towards the goal (center of figure) in a radial manner. To compare the policies learned with our method and expert-designed dense rewards, we independently trained two RL agents via: (i) PPO (on-policy, stochastic actions) [22] and (ii) D4PG (off-policy, deterministic actions) [23], on each of the reward functions and evaluated the policies over 100 trials.

Despite being presented with only 50% successful demonstrations, our method achieved a success rate of (i) $\mathbf{79}\%$ compared to 81% for expert rewards when using PPO, with *29%* improvement over demonstrations, and (ii) $\mathbf{90}\%$ compared to 93% for expert rewards when using D4PG, with *40%* improvement over demonstrations. This indicates that our method is capable of producing expert-like behaviors. Furthermore, this particular environment [21] has shown to be readily transferable to real-robots without any modifications (i.e., the sim2real transfer gap is almost nil), which also indicates the real-world applicability of our framework.

C. CARLA Driving Simulator

We evaluated our method on a realistic driving simulator, CARLA [24], on highway and urban driving scenarios. A demonstrator controls the (ego) car via an analog controller. The states of the car provided by the environment are: lateral distance and heading error between the ego vehicle to the target lane center line (in `meter` and `rad`), ego vehicle's speed (in `meters per second`), and (`Boolean`) indicator of whether there is a front vehicle within a safety margin. Based on this information, we formulated 3 STL specifications that are similar to the 2D driving and mobile navigation experiments, which informally, correspond to (i) keeping close to the lane center φ_1, (ii) maintaining speed limits φ_2 and (iii) maintaining safe distance to any lead vehicle φ_3; formal definitions are provided in [19]. For this scenario, we recorded 15 demonstrations from one of the authors of this letter via an analog steering and pedal controller. These 15 videos were split uniformly across 3 batches where, the 5 videos in each batch showed a common behavior, but the behaviors were different across the batches. All videos were exclusive to their respective batches, i.e., no video was used in more than 1 batch, and each video was 30 seconds long on average.

User Study: The recorded driving videos were used to perform a user study to determine if users would rate the driving behavior similarly, thereby providing evidence that the graphs generated using PeGLearn produce accurate ranking of specifications[4]. Using the Amazon Mechanical Turk (AMT) platform, we created a survey of the 3 batches representing the split of the 15 videos. Each participant of a batch was shown the corresponding 5 driving videos and was tasked with providing $Likert$ ratings for: (i) performance of demonstrator on each of the 3 task specifications described in natural language and (ii) ranking of specifications based on overall driving behaviors. We recruited 146 human participants via AMT service, each with an approval rating over 98%. The average driving experience of the participants was 22.4 years. This user study was approved by the *University of Southern California Institution Review Board* on *January 10, 2022*. Additional information about this survey and participant demographics are provided in the supplemental document [19]. The goal of this study was to first ensure that the PeGLearn orderings were similar to expert orderings and not a random coincidence. The total number of possible orderings for the 3 specifications is 27 $(= 3^3)$, so for each video and participant, we also generated an ordering randomly and uniformly chosen from the space of 27 orderings. Based on this, we formulate the hypothesis as:

Hypothesis 1: The similarity between human expert and PeGLearn orderings is significantly higher than that of a random ordering.

Secondly, we wanted to investigate the similarity between existing clustering techniques such as K-Means [25], [26] and our algorithm, with expert rankings. The problem of finding weights for specifications resembles anomaly detection where the *bad* demonstrations are outliers and methods such as clustering, classification or combination of both can be employed [27]. Additionally, the weights for specifications that we seek to learn, indicate the importance of the specifications, which is analogous to importance/rank of features in classification tasks [28]. Hence, we use K-Means combined with SVM [25], [26] for comparison purposes, which we now refer to as Km+SVM. Clustering is first employed to extract the clusters of demonstrations based on their corresponding ratings. Note that the set of demonstrations can contain all *good* (ones with positive ratings for all specifications), all *bad* (ones with negative ratings for any specification) or a mixture of both these types. Based on the inertia of the k-means clustering for this data, we found that $k = 2$ was optimal for each batch and provided the best fit. Then, SVM was used to classify the cluster centroids and extract the weights. The magnitude of the weights indicate the relative importance/ranking of each feature (specification) [28]. So we ranked the weights to compare with PeGLearn rankings. Therefore, we formulate our second hypothesis as follows:

Hypothesis 2: The similarity between human expert and PeGLearn orderings is significantly higher than that of K-means+SVM.

Analysis: We first obtain the ratings and hence specification-orderings from all sources: participants, PeGLearn, Km+SVM and uniform-random algorithm. We then compute the *Hamming* distance [29] between the human expert orderings and orderings from (i) PeGLearn, (ii) Km+SVM, and (iii) uniform-random. The reason being that the *Hamming* distance between any two sequences of equal lengths measures the number of element-wise disagreements or mismatches, and hence gives an estimate of how close any two orderings are. The distance is a value in $[0, 1]$ with 0 representing same sequences and 1 indicating completely different sequences. To perform statistical analysis, we introduce a few notations for convenience, as follows: (i) PH for PeGLearn–human *Hamming* distance or error, (ii) KH for Km+SVM–human *Hamming* error, and (iii) RH for uniform-random–human *Hamming* error. We concatenate these errors under the name "Score" for analysis purposes. Note that lower the *Hamming* distance or "Score," the more similar are the two orderings. A two-way ANOVA was conducted to examine the effects of agent type (i.e., $\{\text{PH}, \text{KH}, \text{RH}\}$) and batch number (i.e., $\{1, 2, 3\}$) on the "Score". There was no statistically significant interaction between agent type and batch number for "Score," $F(4, 429) = 1.605, p = .172$, partial $\eta^2 = .015$. Therefore, an analysis of the main effect of agent type was performed, which indicated there was a statistically significant main effect of agent type, $F(2, 429) = 34.558, p < .001$, partial $\eta^2 = .139$. All pairwise comparisons were run, where reported 95% confidence intervals and p-values are Bonferroni-adjusted. The marginal means for "Score" were. 475 (SE =. 022) for PH,. 724 (SE =. 022) for KH and. 660 (SE =. 022) for RH. The "Score" means between RH and PH were found to be statistically significant $p < .001$, showing support for **H1**. Similarly, the "Score" means for PH and KH were statistically significant $p < .001$ and the mean for KH was higher than PH, supporting **H2**. Lastly, there was no statistically significant main effect of batch number on "Score," $F(2, 429) = .172, p = .842$, partial $\eta^2 = .001$.

Comparison with Km+SVM: K-means typically has a complexity of $\mathcal{O}(mknt)$ where m is the number of data points (i.e., demonstrations), k is the number of components/clusters, n is the dimension of data (i.e., number of specifications) and t is the number of iterations. Linear SVM follows linear complexity in m and so the combination of Km+SVM is still $\mathcal{O}(mknt)$. Since there are $k = 2$ components in our formulation, the k is treated as a constant and the complexity is just $\mathcal{O}(mnt)$. Our algorithm on the other hand has a complexity of $\mathcal{O}(mn^2)$ when using matrices to represent graphs. This shows that our algorithm not only performs better than clustering methods, but is also more efficient because generally, the number of specifications is much smaller than the number of iterations to converge ($n \ll t$). *All the experiments and results show that our method can not only learn accurate rewards similar to the way humans perceive them, but it does so with a limited number of even imperfect data*. Similarly, we also performed experiments on a *real-world* surgical robot dataset, JIGSAWS, to demonstrate how human *Likert* ratings can be used to learn DAGs [19]. Additionally, together with the LfD-STL framework, we are able to learn temporal-based rewards, even in continuous and high-dimensional spaces with just a handful of demonstrations.

V. Related Work

Learning-from-demonstrations (LfD) has been extensively explored to obtain control policies for robots [30], [31] using methods such as behavioral cloning [32] and inverse reinforcement learning (IRL) [20], [33], [34].

Aside from our prior works, there have been many other methods involving temporal logics with demonstrations. A counterexample-guided approach using probabilistic computation tree logics for safety-aware apprenticeship learning is proposed in [35], which makes use of DAGs to search for counterexamples. The authors in [36] utilize model-predictive control to design a controller that imitates demonstrators while deciding the trade-offs among STL rules. Similar to our prior works, they assume that the priorities or ordering among the STL specifications are provided beforehand. An active learning approach has been explored in [37], in which the rewards are learned by augmenting the state space of a Markov Decision Process (MDP) with an automaton, represented by directed graphs. An alternative approach to characterize the expressivity of Markovian rewards has been proposed in [38], which could provide interpretations for rewards in terms of task descriptions. However, our LfD-STL framework offers several critical advantages over IRL and the methods of [38], such as: employing non-Markovian rewards via task-based temporal logics and empirically having a significant reduction in sample and computation complexity. Additionally, the methods in [38] have been explored for deterministic systems, while LfD-STL can also generalize to stochastic dynamics and continuous spaces as seen in our experiments.

Causal influence diagrams of MDPs via DAGs for explainable online-RL have been recently investigated [39]. Another related work by the authors of [40] make use of causal models to extract causal explanations of the behavior of model-free RL agents. They represent action influence models via DAGs in which the nodes are random variables representing the agent's world and edges correspond to the actions. These works are mainly focused on explainability in forward RL (i.e., when rewards are already known), while our method is mainly focused on generating intuitive representations of behaviors and rewards to be used later in forward RL.

In the area of reward explanations for RL tasks, the method proposed in [41] decomposes rewards into several components, based on which, the RL agent's action preferences can be explained and can help in finding bugs in rewards. Another work pertaining to IRL [42] uses expert-scored trajectories to learn a reward function. This work, which builds on standard IRL, typically relies on a large dataset containing several hundreds of nearly-optimal demonstrations and hence generating scores for each of them. By Theorem 1, our method can also overcome any rank-conflicts arising out of myopic trajectory preferences [43]. The authors in [4] have investigated the reward-explanation problem in the context of human-robot teams wherein the robot, via interactions, learns the reward that is known to the human. They propose 2 categories of reward explanations: (i) feature-space: where rewards are explained through individual features comprising the reward function and their relative weights, and (ii) policy-space: where demonstrations of actions under a reward function are used to explain the rewards. Our work can be regarded as a combination of these categories since it uses specifications as features along with inferred weights, and demonstrations.

VI. Conclusion

In this letter, we proposed a novel algorithm, *PeGLearn*, to capture the performance and provide intuitive holistic representations of demonstrations in the form of graphs. We showed, through challenging experiments in robotic domains, that the inferred graphs could be directly applied to the existing LfD-STL framework to extract rewards and robust control policies via RL, with a limited number of even imperfect demonstrations. The user study conducted showed that our graph-based method produced more accurate results than (un)supervised algorithms in terms of similarities with human ratings. We believe our work is a step in the direction of developing interpretable and explainable learning systems with formal guarantees, which is one of the prominent challenges today [1], [2], [38]. Using intuitive structures such as DAGs to represent rewards and trajectories would provide insights into the learning aspects of RL agents, as to the quality of behaviors they are learning and can be used alongside/integrated with works in explainable AI [3], [4]. In the future, we will also investigate the use of such graphs during RL as feedback to improve policies and provide performance guarantees.

REFERENCES

[1] National Academies of Sciences, Engineering, and Medicine, *Human-AI Teaming: State-of-the-Art and Research Needs*. Washington, DC, USA: The Nat. Academies Press, 2022. [Online]. Available: https://nap.nationalacademies.org/catalog/26355/human-ai-teaming-state-of-the-art-and-research-needs

[2] H. Kress-Gazit et al., "Formalizing and guaranteeing human-robot interaction," *Commun. ACM*, vol. 64, no. 9, pp. 78–84, 2021.

[3] R. Paleja, M. Ghuy, N. R. Arachchige, R. Jensen, and M. Gombolay, "The utility of explainable AI in Ad Hoc human-machine teaming," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 610–623.

[4] L. Sanneman and J. A. Shah, "An empirical study of reward explanations with human-robot interaction applications," *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, pp. 8956–8963, Oct. 2022.

[5] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," 2016, doi: 10.48550/ARXIV.1606.06565.

[6] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic synthesis for single and multi-robot tasks," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 3687–3694, Apr. 2021.

[7] X. Li, C. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3834–3839.

[8] X. Li, Y. Ma, and C. Belta, "Automata guided reinforcement learning with demonstrations," 2018, doi: 10.48550/ARXIV.1809.06305.

[9] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. Conf. Decis. Control*, 2016, pp. 6565–6570.

[10] M. Wen, R. Ehlers, and U. Topcu, "Correct-by-synthesis reinforcement learning with temporal logic constraints," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 4983–4990.

[11] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. Int. Symp. Formal Techn.. Real-Time. Fault-Tolerant Syst.* 2004, pp. 152–166.

[12] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Learning from demonstrations using signal temporal logic," in *Proc. Conf. Robot Learn.*, 2020, pp. 2228–2242.

[13] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Learning from demonstrations using signal temporal logic," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 6250–6257, Oct. 2021.

[14] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2010.

[15] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova, "Learning from demonstration for shaping through inverse reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2016, pp. 429–437.

[16] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2018.

[17] G. Norman, "Likert scales, levels of measurement and the 'laws' of statistics," *Adv. Health Sci. Educ.*, vol. 15, no. 5, pp. 625–632, 2010.

[18] M. Chen, S. Nikolaidis, H. Soh, D. Hsu, and S. Srinivasa, "Planning with trust for human-robot collaboration," in *Proc. IEEE/ACM Int. Conf. Human-Robot Interact.*, 2018, pp. 307–315.

[19] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Learning performance graphs from demonstrations via task-based evaluations - supplemental material," 2022. [Online]. Available: https://aniruddh-puranic.info/files/performance_graph_supplemental.pdf

[20] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2008, pp. 1433–1438.

[21] M. Lucchi, F. Zindler, S. Mühlbacher-Karrer, and H. Pichler, "robo-gym– An open source toolkit for distributed deep reinforcement learning on real and simulated robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5364–5371.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, doi: 10.48550/ARXIV.1707.06347.

[23] G. Barth-Maron et al., "Distributed distributional deterministic policy gradients," in *Proc. Int. Conf. Learn. Representations*, 2018.

[24] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. Conf. Robot Learn.*, 2017, pp. 1–17.

[25] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006, pp. 1–16.

[26] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: The MIT Press, 2012.

[27] B. Zheng, S. W. Yoon, and S. S. Lam, "Breast cancer diagnosis based on feature extraction using a hybrid of K-means and support vector machine algorithms," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1476–1482, 2014.

[28] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, no. 1, pp. 389–422, 2002.

[29] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Tech. J.*, vol. 29, no. 2, pp. 147–160, 1950.

[30] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc. Int. Conf. Mach. Learn.*, 1997, pp. 12–20.

[31] S. Schaal, "Learning from demonstration," in *Proc. Neural Inf. Process. Syst.*, 1996, pp. 1040–1046.

[32] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 4950–4957.

[33] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2000, pp. 663–670.

[34] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2004, p. 1.

[35] W. Zhou and W. Li, "Safety-aware apprenticeship learning," in *Proc. Int. Conf. Comput. Aided Verification*, 2018, pp. 662–680.

[36] K. Cho and S. Oh, "Learning-based model predictive control under signal temporal logic specifications," in *Proc. Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 7322–7329.

[37] F. Memarian, Z. Xu, B. Wu, M. Wen, and U. Topcu, "Active task-inference-guided deep inverse reinforcement learning," in *Proc. Conf. Decis. Control*, 2020, pp. 1932–1938.

[38] D. Abel et al., "On the expressivity of Markov reward," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 7799–7812.

[39] T. Everitt, M. Hutter, R. Kumar, and V. Krakovna, "Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective," *Synthese*, vol. 198, no. 27, pp. 6435–6467, May 2021.

[40] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Explainable reinforcement learning through a causal lens," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 2493–2500.

[41] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable reinforcement learning via reward decomposition," in *Proc. IJCAI/ECAI Workshop Explainable Artif. Intell.*, 2019.

[42] L. El Asri, B. Piot, M. Geist, R. Laroche, and O. Pietquin, "Score-based inverse reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2016, pp. 457–465.

[43] E. Biyik, D. P. Losey, M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh, "Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences," *Int. J. Robot. Res.*, vol. 41, 2022, pp. 45–67.