Learning Performance Graphs from Demonstrations via Task-Based Evaluations

Aniruddh G. Puranic, Jyotirmoy V. Deshmukh and Stefanos Nikolaidis
Department of Computer Science
University of Southern California, USA
{puranic, jdesmuk, nikolaid}@usc.edu

Abstract: In the paradigm of robot learning-from-demonstrations (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.

Keywords: Learning from demonstrations, reinforcement learning, temporal logic

1 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 paper 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 perfor-
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 paper, 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 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 3.2.

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. 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.

### 2 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

---

1Here we assume that Likert scales are interval scales[17].
agent. Formally, a demonstration \( \xi \) 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, \( \xi \) 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 \( \varphi \). In this regard, we define a rating function via Definition 2.

**Definition 2 (Rating Function).** A rating function \( R \) is a real-valued function that maps a specification and a time-series data or trajectory to a real number, i.e., \( R : \Phi \times \Xi \to \mathbb{R} \), where, \( \Phi \) and \( \Xi \) 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 \( \varphi \) and a demonstration \( \xi \), the rating (also referred to as evaluation or score) of \( \xi \) with respect to \( \varphi \) is denoted by \( \rho = R(\varphi, \xi) \). This \( \rho \) is negative if \( \xi \) violates \( \varphi \), 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 \( x \sim y \) 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.

## 3 Problem Definition and Methodology

### 3.1 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 \( \rho = \rho_1, \ldots, \rho_n \) is obtained. We can then represent this as an \( m \times n \) matrix \( Z \) where each row \( i \) represents a demonstration and a column \( j \) represents a specification. An element \( \rho_{ij} \) indicates the rating or score of demonstration \( i \) for specification \( j \), i.e., \( \rho_{ij} = R(\varphi_j, \xi_i) \).

\[
Z = \begin{bmatrix}
\rho_{11} & \rho_{12} & \cdots & \rho_{1n} \\
\rho_{21} & \rho_{22} & \cdots & \rho_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\rho_{m1} & \rho_{m2} & \cdots & \rho_{mn}
\end{bmatrix} = \begin{bmatrix}
\rho_{1}^T \\
\rho_{2}^T \\
\vdots \\
\rho_{m}^T
\end{bmatrix}
\]

(1)

As in prior work on LfD-STL, we need to compute a cumulative score or rating \( r_\xi \) for each demonstration to collectively represent its individual specification scores, and so we have an \( m \times 1 \) vector \( r = [r_{\xi_1}, r_{\xi_2}, \ldots, r_{\xi_m}]^T \). 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 \( w = [w_1, w_2, \ldots, w_\Phi]^T \) from which we can obtain the cumulative scores as \( Z \cdot w = r \). In other words, for each demonstration \( \xi \), \( r_\xi = \rho_\xi^T \cdot w \). The objective is to compute both \( w \) and \( r \), given only \( Z \), 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 \( w \) proposed in [12, 13] required the demonstrators to specify their preferences encoded as a DAG and
the weights were computed via (2)².

\[ w(\varphi) = |\Phi| - |\text{ancestor}(\varphi)| \]  

(2)

where, \( \text{ancestor}(v) = \{ u \mid u \sim v, u \in V \} \), 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 \( \Phi \) 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 \( \Phi \), 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.

### 3.2 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.

#### 3.3 Generating local graphs

Each demonstration \( \xi \in \Xi \) is associated with a vector of ratings \( \rho_\xi = [\rho_1, ..., \rho_\Phi]^T \), and the objective is to construct a weighted DAG for \( \xi \) 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.

Consider 4 specifications \( \varphi_i; i \in \{1, 2, 3, 4\} \). Let a demonstration, say \( \xi \in \Xi \) have evaluations \( \rho_\xi = [\rho_1, \rho_2, \rho_3, \rho_4] \) 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 \( \Phi \), i.e., a node contains the index of the specification it represents. An edge between two nodes \( \varphi_i \) and \( \varphi_j \) 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 \( \epsilon \) acts as a high-pass filter and can be tuned depending on

²Before robustness values of various temporal logic specifications are combined, they are normalized using \( \tanh \) or piece-wise linear functions.
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 $G_\xi$

begin
1. $G_\xi \leftarrow 0_{n \times n}$ // zero matrix
2. $S \leftarrow \emptyset$ // Create an empty queue
3. for $j = 1$ to $n$ do
4.   Obtain the rating or score $s_j$ for specification $j$; $S.insert((j, s_j))$
5. end for
6. $S' \leftarrow S$ 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. end if
16. end for
17. return $G_\xi$

Figure 2: Example local graph for a demonstration.

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

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 $\text{Algorithm 1:}$ 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 $G_\xi$

begin
1. $G_\xi \leftarrow 0_{n \times n}$ // zero matrix
2. $S \leftarrow \emptyset$ // Create an empty queue
3. for $j = 1$ to $n$ do
4.   Obtain the rating or score $s_j$ for specification $j$; $S.insert((j, s_j))$
5. end for
6. $S' \leftarrow S$ 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. end if
16. end for
17. return $G_\xi$

Figure 2: Example local graph for a demonstration.

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

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 $\text{Algorithm 1:}$ 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 $G_\xi$

begin
1. $G_\xi \leftarrow 0_{n \times n}$ // zero matrix
2. $S \leftarrow \emptyset$ // Create an empty queue
3. for $j = 1$ to $n$ do
4.   Obtain the rating or score $s_j$ for specification $j$; $S.insert((j, s_j))$
5. end for
6. $S' \leftarrow S$ 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. end if
16. end for
17. return $G_\xi$

Figure 2: Example local graph for a demonstration.

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

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

The demonstration $\xi$ argument in $\rho$ was dropped for convenience since we are only considering 1 demonstration at a time.
linear when using adjacency lists and quadratic when using adjacency matrix to represent the graph. The total complexity is thus $O(mn^2)$ in the worst case (using matrix representation).

3.4 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. 3c. This is not necessarily acyclic since there is a cycle between the nodes of $\varphi_1$ and $\varphi_2$. In this figure, each $w'_{ij} = (w_{1ij} + w_{2ij})/2$. This intermediate graph needs to be further reduced to a weighted DAG, i.e., by eliminating any cycles/loops.

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
4 // Extracts edge-weighted DAG from the raw Performance-Graph
5 $G \leftarrow 0_{n \times n}$ // zero matrix
6 for $i = 1$ to $n$ do
7   for $j = 1$ to $n$ do
8     $G[i,j] \leftarrow \max(0, G'[i,j] - G'[j,i])$
9     if $G[i,j] < \epsilon$ then $G[i,j] \leftarrow 0$
10 return $G$

3.5 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 $\epsilon$ 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 $\xi_1$ and $\xi_2$, the partial order $\xi_1 \preceq \xi_2$ is defined when $\rho_1(i) \leq \rho_2(i), \forall i \in \{1, \cdots, n\}$. Thus, we say that $\xi_2$ is better or at least as good as $\xi_1$. Then, by making use of Lemma 3.1, we arrive at Theorem 3.1 that addresses the problem definition (all proofs are in [19]).

**Lemma 3.1.** For a DAG, the weights associated with the nodes computed via (2) are non-negative.

**Theorem 3.1.** For any two demonstrations $\xi_1$ and $\xi_2$ in an environment, the partial ordering $\xi_1 \preceq \xi_2$ is preserved by PeGLearn.

The global DAG imposes a partial order of specifications. For any 2 specifications $\varphi_i$ and $\varphi_j$, 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.

4 Experiments

4.1 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 $\varphi_1$, (ii) avoiding the hindrance/obstacle regions $\varphi_2$, (iii) always staying within the workspace region $\varphi_3$, and (iv) reaching the goal as fast as possible $\varphi_4$. 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 [20, 14].

4.2 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

---

\(^4\)This paper is accompanied by a multimedia (video) for the experiments.
Figure 5: Results for the MiR100 navigation environment. (a) Obstacles and boundary walls are shown in red point clouds; green sphere is the 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 \( \theta_t \)), linear (\( v_t \)) and angular (\( w_t \)) velocities, and 16 readings from the robot’s Lidar for detecting obstacles (\( x^i_t = 1 \)). 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 \( \delta \) 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:
   \[ \varphi_v := \mathbf{G}(v_{min} \leq v_t \leq v_{max}), \quad \text{and} \quad \varphi_w := \mathbf{G}(w_{min} \leq w_t \leq w_{max}). \]
   The limits conform to the robot’s capabilities.

3. Always avoid obstacles:
   \[ \varphi_s := \mathbf{G} \left( \bigwedge_{i=1}^{16} (x^i_t > 0) \right), \]
   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) 79% compared to 81% for expert rewards when using PPO, with 29% improvement over demonstrations, and (ii) 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.

4.3 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 meters 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 $\varphi_1$, (ii) maintaining speed limits $\varphi_2$ and (iii) maintaining safe distance to any lead vehicle $\varphi_3$; formal definitions are provided in [19]. For this scenario, we recorded 15 demonstrations from one of the authors of this paper 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. 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
The Hamming distance between two orderings is a measure of how different they are. If the orderings have the same number of elements, the distance is the number of disagreements or mismatches between them. The reason for using Hamming distance is that it provides a direct way to compare the orderings. We combine 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., {PH, KH, 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 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 $O(mknt)$ 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 $O(mknt)$. Since there are $k = 2$ components in our formulation, the $k$ is treated as a constant and the complexity is just $O(mnt)$. Our algorithm on the other hand has a complexity of $O(mn^2)$ 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.

5 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) [33, 20, 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 3.1, 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.

6 Conclusion

In this work, 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 LiD-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. The National Academies Press, 2022. ISBN 978-0-309-27017-5. doi:10.17226/26355.

[2] H. Kress-Gazit, K. Eder, G. Hoffman, H. Admoni, B. Argall, R. Ehlers, C. Heckman, N. Jansen, R. Knepper, J. Křetínský, S. Levy-Tzedek, J. Li, T. Murphey, L. Riek, and D. Sadigh. Formalizing and guaranteeing human-robot interaction. Commun. ACM, 2021.

[3] R. Paleja, M. Ghuy, N. R. Arachchige, R. Jensen, and M. Gombolay. The utility of explainable ai in ad hoc human-machine teamwork. In NeurIPS, 2021.

[4] L. Sanneman and J. A. Shah. An empirical study of reward explanations with human-robot interaction applications. RA-L, 2022.

[5] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety, 2016.

[6] D. Gundana and H. Kress-Gazit. Event-based signal temporal logic synthesis for single and multi-robot tasks. IEEE RA-L, 2021.
[7] X. Li, C. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. In *IROS*, 2017.

[8] X. Li, Y. Ma, and C. Belta. Automata guided reinforcement learning with demonstrations, 2018.

[9] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. Q-Learning for robust satisfaction of signal temporal logic specifications. In *Conference on Decision and Control (CDC)*, 2016.

[10] M. Wen, R. Ehlers, and U. Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *IROS*, 2015.

[11] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In Y. Lakhnech and S. Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30206-3.

[12] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis. Learning from demonstrations using signal temporal logic. In *CoRL*, 2020.

[13] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis. Learning from demonstrations using signal temporal logic. In *IEEE Robotics and Automation Letters (RA-L)*, 2021.

[14] B. D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, USA, 2010.

[15] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, 2016.

[16] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *ICLR*, 2018.

[17] G. Norman. Likert scales, levels of measurement and the “laws” of statistics. *Advances in health sciences education*, 2010.

[18] M. Chen, S. Nikolaidis, H. Soh, D. Hsu, and S. Srinivasa. Planning with trust for human-robot collaboration. In *HRI*, 2018.

[19] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis. Learning performance graphs from demonstrations via task-based evaluations - supplemental material, 2022. URL 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 *AAAI*, 2008.

[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. *IROS*, 2020.

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

[23] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. P. Lillicrap. Distributed distributional deterministic policy gradients. In *ICLR*, 2018.

[24] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *CoRL*, 2017.

[25] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[26] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. 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 Systems with Applications*, 2014.
[28] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Mach. Learn.*, 2002.

[29] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[30] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *ICML*, 1997.

[31] S. Schaal. Learning from demonstration. In *NIPS*, 1996.

[32] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

[33] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[34] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[35] W. Zhou and W. Li. Safety-aware apprenticeship learning. In *Computer Aided Verification CAV*, 2018.

[36] K. Cho and S. Oh. Learning-based model predictive control under signal temporal logic specifications. In *ICRA*, 2018.

[37] F. Memarian, Z. Xu, B. Wu, M. Wen, and U. Topcu. Active task-inference-guided deep inverse reinforcement learning. In *CDC*, 2020.

[38] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. L. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. In *NeurIPS*, 2021.

[39] T. Everitt, M. Hutter, R. Kumar, and V. Krakovna. Reward tampering problems and solutions in reinforcement learning: a causal influence diagram perspective. *Synthese*, May 2021.

[40] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere. Explainable reinforcement learning through a causal lens. *AAAI*, 2020.

[41] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI. A Workshop on Explainable Artificial Intelligence.*, 2019.

[42] L. El Asri, B. Piot, M. Geist, R. Laroche, and O. Pietquin. Score-based inverse reinforcement learning. In *AAMAS*, 2016.

[43] E. Bıyık, 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. *IJRR*, 2022.

[44] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42), 2009.

[45] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design*, 53(1), 2018.

[46] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010.

[47] A. Dhonthi, P. Schilling, L. D. Rozo, and D. Nardi. Study of signal temporal logic robustness metrics for robotic tasks optimization. *CoRR*, abs/2110.00339, 2021.

[48] A. Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification CAV*, 2010.

[49] D. Ničković and T. Yamaguchi. Rtamt: Online robustness monitors from stl. In *Automated Technology for Verification and Analysis*, 2020.

[50] H. Hasselt. Double q-learning. In *NIPS*, 2010.
[51] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[52] Y. Gao, S. Vedula, C. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Béjar, D. Yuh, C. C. Chen, R. Vidal, S. Khudanpur, and G. Hager. Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling. In Modeling and Monitoring of Computer Assisted Interventions (M2CAI) – MICCAI Workshop, 2014.
Appendix

A Signal Temporal Logic

Signal Temporal Logic (STL) is a real-time logic, generally interpreted over a dense-time domain for signals whose values are from a continuous metric space (such as $\mathbb{R}^n$). The basic primitive in STL is a signal predicate $\mu$ that is a formula of the form $f(x(t)) > 0$, where $x(t)$ is the tuple (state, action) of the demonstration $x$ at time $t$, and $f$ maps the signal domain $D = (S \times A)$ to $\mathbb{R}$. STL formulas are then defined recursively using boolean combinations of sub-formulas, or by applying an interval-restricted temporal operator to a sub-formula. The syntax of STL is formally defined as follows: $\varphi ::= \mu \mid \neg \varphi \mid \varphi \wedge \varphi \mid G_I \varphi \mid F_I \varphi \mid I \varphi I \varphi$. Here, $I = [a, b]$ denotes an arbitrary time-interval, where $a, b \in \mathbb{R}^{\geq 0}$. The semantics of STL are defined over a discrete-time signal $x$ defined over some time-domain $T$. The Boolean satisfaction of a signal predicate is simply True (T) if the predicate is satisfied and False (\perp) if it is not, the semantics for the propositional logic operators $\neg, \wedge$ (and thus $\lor, \rightarrow$) follow the obvious semantics. The following behaviors are represented by the temporal operators:

- At any time $t$, $G_I(\varphi)$ says that $\varphi$ must hold for all samples in $t + I$.
- At any time $t$, $F_I(\varphi)$ says that $\varphi$ must hold at least once for samples in $t + I$.
- At any time $t$, $\varphi I \psi$ says that $\psi$ must hold at some time $t'$ in $t + I$, and in $[t, t')$, $\varphi$ must hold at all times.

**Definition 4 (Quantitative Semantics for Signal Temporal Logic).** Given an algebraic structure $(\oplus, \otimes, T, \perp)$, we define the quantitative semantics for an arbitrary signal $x$ against an STL formula $\varphi$ at time $t$ as in Table 1.

| $\varphi$        | $\rho(\varphi, x, t)$ |
|------------------|------------------------|
| true | $\top$ |
| false | $\bot$ |
| $\mu$ | $f(x(t))$ |
| $\neg \varphi$ | $\neg \rho(\varphi, x, t)$ |
| $\varphi_1 \wedge \varphi_2$ | $\otimes(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t))$ |
| $\varphi_1 \lor \varphi_2$ | $\oplus(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t))$ |
| $G_I(\varphi)$ | $\otimes_{t \in T} \rho(\varphi, x, t)$ |
| $F_I(\varphi)$ | $\oplus_{t \in T} \rho(\varphi, x, t)$ |
| $\varphi I \psi$ | $\otimes_{t \in T} \rho(\psi, x, t)$ |

A signal satisfies an STL formula $\varphi$ if it is satisfied at time $t = 0$. Intuitively, the quantitative semantics of STL represent the numerical distance of “how far” a signal is away from the signal predicate. For a given requirement $\varphi$, a demonstration or policy $d$ that satisfies it is represented as $d \models \varphi$ and one that does not, is represented as $d \not\models \varphi$. In addition to the Boolean satisfaction semantics for STL, various researchers have proposed quantitative semantics for STL, [44, 45] that compute the degree of satisfaction (or robust satisfaction values) of STL properties by traces generated by a system. In this work, we use the following interpretations of the STL quantitative semantics: $\top = +\infty$, $\perp = -\infty$, and $\oplus = \max$, and $\otimes = \min$, as per the original definitions of robust satisfaction proposed in [44, 46].

Using these semantics allows a demonstration that satisfies a specification to have non-negative robustness (score) for that specification, and a demonstration that violates it will have a negative robustness (score). We use STL in our work because it offers a rich set of quantitative semantics (Definition 4) that are suitable for formal analysis and reasoning of systems. The requirements defined with STL are grounded w.r.t. the actual description of the tasks/objectives. Furthermore, STL allows designers or users to specify constraints that evolve over time and define causal dependencies among tasks. Their semantics allow for the definition of non-Markovian rewards and accurately evaluating trajectories and policies for RL.

In our setting, a task can consist of multiple specifications. However, the robustness of each specification may lie on different scales. Consider for example, a driving scenario, where one specification
concerns the speed of the vehicle, while another concerns the steering angle. Since the measurement scale of speed is significantly larger than angle (e.g., 60 mph vs 1.6°), the robustness of the corresponding specifications also differs significantly. Furthermore, if the maximum robustness a car can achieve is 60 and 1.6 for the respective specifications, then directly performing summation on them would induce bias towards the speed specification. To avoid this bias, the robustness ranges need to be normalized. Some common normalization techniques are surveyed in [47]. We use the tanh hyperbolic smoothing in our work to bound the robustness values.

B Derivations and Proofs

B.1 Space of all directed graphs

In regard to the number of different orderings in extracting local graphs, given \( n \) specifications, the number of permutations or arrangements is \( n! \). For each permutation, there is 1 operator from \( \{>, =\} \) that can be placed in between any two specifications (e.g., \( a > b \)). The number of such “places” is \( n - 1 \) and hence the number of operator arrangements for each permutation is \( 2^{n-1} \) or \( |op|^{n-1} \) in general. However, we can observe that one of the arrangements for each permutation consists of the ‘=’ operator appearing in all the “places”. For example, \( a = b = c \) is the same as the permutation \( b = a = c \) and so on. Hence, all the \( n! \) permutations share this common/redundant ordering, and so we need to remove all but one of them. Thus, the total number of unique orderings over all the permutations is \( n! \cdot |op|^{n-1} - n! + 1 = n! \cdot |op|^{n-1} - 1 \). Following the use of directed graphs to reduce this search space, we first need to derive the number of possible directed graphs. For a directed graph without self-loops, there are 3 possible edge categories between any two nodes, a cycle comprises at least 3 vertices because we allow only 1 edge. Hence, all the \( n! \) permutations share this common/redundant ordering, and so we need to remove all but one of them. Thus, the total number of unique orderings over all the permutations is \( n! \cdot |op|^{n-1} - n! + 1 = n! \cdot |op|^{n-1} - 1 \). Following the use of directed graphs to reduce this search space, we first need to derive the number of possible directed graphs. For a directed graph without self-loops, there are 3 possible edge categories between any two nodes, a cycle comprises at least 3 vertices because we allow only 1 edge to exist between any two nodes. So, adding the number of cycles, we get:

\[
\sum_{k=3}^{n} \binom{n}{k} = \sum_{k=0}^{n} \binom{n}{k} - 2 \sum_{k=0}^{n} \binom{n}{k} = 2^n - \left[ \frac{n!}{0!(n-0)!} + \frac{n!}{1!(n-1)!} + \frac{n!}{2!(n-2)!} \right] = 2^n - [1 + n + n(n-1)/2]
\]

We can then reverse the edges and obtain another \( 2^n - [1 + n + n(n-1)/2] \) cycles, therefore the total number of cycles is twice this number = \( 2^{n+1} - (n^2 + n + 1) \). Finally, the number of valid directed graphs is \( 3^{n(n-1)/2} - [2^{n+1} - (n^2 + n + 2)] = 3^{n(n-1)/2} - 2^{n+1} + n^2 + n + 2 \), which is still exponential, but has eliminated the factorial component of the search space.

B.2 Proof of Lemma and Theorem

We provide proofs for the Lemma and Theorem stated in the main paper.

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

\[
w(\varphi) = |\Phi| - |\text{ancestor}(\varphi)|
\]

Proof Sketch. From the LfD-STL framework, the weights for specifications represented by the DAG nodes are given by (2). We know that \( |\Phi| = n \) and \( |\text{ancestor}(\varphi)| \) is a set whose cardinality is non-negative. In a DAG, there are no cycles and hence \( |\text{ancestor}(\varphi)| \) is an integer in \([0, n - 1]\). By this equation, the minimum weight (i.e., worst-case) for any node representing a specification \( \varphi \) occurs when that node is a leaf and all other \( n - 1 \) nodes are its ancestors. Therefore, \( w(\varphi) = |\Phi| - |\text{ancestor}(\varphi)| \implies w(\varphi) = n - (n - 1) = 1 \geq 0 \). Similarly, the maximum value of \( w(\varphi) \) is \( n \), i.e., there are no ancestors when \( \varphi \) is one of the root nodes in the DAG. This non-negative nature
of weights also holds true when the weights are normalized via a softmax function since it is used to represent a probability distribution that lies in the interval \([0, 1]\).

Using this lemma, we derive the proof for the theorem as described below.

The theorem states: For any two demonstrations \(\xi_1\) and \(\xi_2\) in an environment, the partial ordering \(\xi_1 \preceq \xi_2\) is preserved by PeGLearn.

Proof. Recall that for any two demonstrations \(\xi_1\) and \(\xi_2\) in an environment, if \(\xi_1 \preceq \xi_2\), then the cumulative rating/scores are such that \(r_{\xi_1} \leq r_{\xi_2}\). Also recall the notation that \(\rho_{\xi_i} = [\rho_{1i}, \cdots, \rho_{ni}]^T\). Let there be \(n\) specifications for the environment, then for these two demonstrations, we have:

\[
Z = \begin{bmatrix}
\rho_{11} & \rho_{12} & \cdots & \rho_{1n} \\
\rho_{21} & \rho_{22} & \cdots & \rho_{2n}
\end{bmatrix}
\]

and \(w = [w_1, w_2, \ldots, w_n]^T\). W.l.o.g., let \(\xi_2\) be at least as good as \(\xi_1\), i.e., we have \(\rho_{1j} \leq \rho_{2j}, \forall j \in \{1, \cdots, n\}\). The cumulative scores for the demonstrations are \(r_{\xi_i} = \rho_{\xi_i}^T \cdot w\) where \(i \in \{1, 2\}\). For any constant \(w_j \geq 0\),

\[
w_j \cdot \rho_{1j} \leq w_j \cdot \rho_{2j} \\
\implies \sum_{j=1}^{n} w_j \cdot \rho_{1j} \leq \sum_{j=1}^{n} w_j \cdot \rho_{2j} \\
\implies \rho_{\xi_1}^T \cdot w \leq \rho_{\xi_2}^T \cdot w \\
\implies r_{\xi_1} \leq r_{\xi_2}
\]

This holds iff \(w_j \geq 0, \forall w_j \in w\)

Once the global DAG is learned, the weights for specifications (nodes) are computed via \((2)\). From the above Lemma, we have shown that these weights are all non-negative. Since the LfD-STL framework ranks the demonstrations by their cumulative scores, this guarantees that better demonstrations are always ranked higher than the others, i.e., a partial order is created, and also provide justification for the use of DAGs.

\[\square\]

C Additional Details on Experiments

The STL formulas in our discrete-world and 2D driving experiments were specified and evaluated using Breach \[48\], and the specifications for the MiR100 and CARLA experiments were evaluated in RTAMT-STL library \[49\]. Note that the complexity of evaluating a trajectory w.r.t. a temporal logic specification is polynomial in the length of the signal and specification \[11\]. However, tools such as Breach and RTAMT are capable of producing linear-time complexity when evaluating In the 2D driving simulator experiment, we used the same neural network architecture as in our prior work [13] that was trained using PyTorch. All experiments were performed on a desktop machine with AMD Ryzen 7 3700X 8-core CPU and Nvidia RTX 2070-Super GPU.

C.1 Discrete-World

We use a grid environment, based on the OpenAI Gym Frozenlake environment, consisting of a set of states \(S = \{\text{start}, \text{goals}, \text{obstacles}\}\) of varying grid sizes such as: 5 × 5, 8 × 8 and 15 × 15 and randomizing the obstacle locations. Stochasticity in the range \(p \in [0.1, 0.8]\) was introduced to the transition dynamics. This environment was created using PyGame library where users provided demonstrations in the PyGame GUI by clicking on their desired states with the task to reach the goal state from start without hitting any obstacles. Due to the stochasticity, unaware to the users, their clicked state may not always end up at the desired location. The user then proceeds to click from that unexpected state till they quit or reach the goal. Just as in \[12, 13\], we used Manhattan distance as the distance metric and formulated the STL specifications:

1. Avoid obstacles at all times (hard requirement): \(\varphi_1 := G_{[0,T]}(d_{\text{obs}}[t] \geq 1)\), where \(T\) is the length of a demonstration and \(d_{\text{obs}}\) is the minimum distance of robot from obstacles computed at each step \(t\).
Eventually, the robot reaches the goal state (soft requirement):  
\[ \varphi_2 := F_{[0,T]}[d_{goal}[t] < 1], \]  
where \( d_{goal} \) is the distance of robot from goal computed at each step. \( \varphi_2 \) depends on \( \varphi_1 \).

3. Reach the goal as fast as possible (soft requirement):  
\[ \varphi_3 := F_{[0,T]}(t \leq T_{goal}), \]  
where \( T_{goal} \) is the upper bound of time required to each the goal, which is computed by running breadth-first search algorithm from start to goal state, since the shortest policy must take at least \( T_{goal} \) to reach the goal. \( \varphi_3 \) depends on both \( \varphi_1 \) and \( \varphi_2 \) in the DAG.

The PeGLearn algorithm was evaluated against (i) user-defined DAGs, (ii) MaxEntropy IRL, and (iii) MaxCausalEntropy IRL. Once rewards were extracted from each algorithm for all environment settings, we used Double Q-Learning [50], as it is suited for stochastic settings, with the modifications to the algorithm at 2 steps (reward update and termination) as described by [12]. The number of episodes varied based on environment complexity such as grid size, number and locations of obstacles. The discount factor \( \gamma \) was set to 0.8 and \( \epsilon \)-greedy strategy with decaying \( \epsilon \) for actions was used. A learning rate of \( \alpha = 0.1 \) was found to work reasonably well after analyzing hyperparameters. Our evaluations over 100 trials showed that policies independently learned from PeGLearn and manually-defined DAGs were able to achieve a task success rate of 80% and 81% respectively for the environments with 0.2 stochasticity. However, the execution time of PeGLearn was within a 2-second increment over that of LfD-STL with manually-specified DAGs.

C.2 2D Driving Simulator

For this experiment, we used the same kinematic model for a car, described by the following equations:

\[
\begin{align*}
\dot{x} &= v \cdot \cos(\theta) + N(0, \sigma^2); \\
\dot{y} &= v \cdot \sin(\theta) + N(0, \sigma^2); \\
\dot{v} &= u_1 \cdot u_2; \\
\dot{\theta} &= v \cdot \tan(\psi); \\
\dot{\psi} &= u_3
\end{align*}
\]

where \( x \) and \( y \) represent the XY’ position of the car; \( \theta \) is the heading; \( v \) is the velocity; \( u_1 \) is the input acceleration; \( u_2 \) is the gear indicating forward (+1) or backward (-1); \( u_3 \) is the input to steering angle \( \psi \). The state of the car at time \( t \) is given by \( S_t = [x, y, \theta, v, \dot{x}, \dot{y}, \dot{\theta}, \dot{\psi}]^T \). Demonstrations are provided by users via an analog Logitech G29 steering with pedal controller or via keyboard inputs. For comparison with prior work, we utilized the same 8 (6 good and 2 bad) demonstrations recorded earlier (Fig. 6).

The distance metric used in this space is Euclidean. The specifications for this scenario are as follows:

1. Avoid obstacles at all times (hard requirement):  
\[ \varphi_1 := G_{[0,T]}(d_{obs}[t] \geq d_{Safe}), \]  
where \( T \) is the length of a demonstration and \( d_{obs} \) is the minimum distance of the car from \( H \) computed at each step \( t \). For our experiments, we used \( d_{Safe} = 3 \) units.

2. Always stay within the workspace/drivable region (hard requirement):  
\[ \varphi_2 := G_{[0,T]}((x, y) \in Box(30, 25)), \]  
where the workspace is defined by a rectan-
gle of dimensions $30 \times 25$ square units. The *Box* is an indicator for the real-valued data in the *OpenAIGym* library.

3. Eventually, the robot reaches the goal state (soft requirement): $\varphi_3 := F_{[0,T]}(d_{\text{goal}}[t] < \delta)$, where $d_{\text{goal}}$ is the distance between centers of car and goal computed at each step $t$ and $\delta$ is a small tolerance when the center of the car is “close enough” to the goal’s center. $\varphi_3$ depends on $\varphi_1$ and $\varphi_2$ in the DAG.

The rewards are assigned to states by modeling the states $s$ as samples of multi-variate Gaussian distribution $N(\mu, \sigma^2 I)$ where $\mu = s$ and $\sigma$ represents the deviations in noise levels, that can be tuned. Here, we use $\sigma = 0.03$. For each $s$, we generated $k = 20$ samples to represent the reachable set and assigned stochastic rewards as described in [13]. The neural network used for regressing the rewards consisted of 2 layers with 200 neurons in each layer that were activated by ReLU. The reward inference for both PeGLearn and manual-DAG baseline had execution times of less than 30 seconds.

### C.3 MiR100 Navigation Experiment

Given the 30 demonstrations, PeGLearn was able to extract the rewards within 30 seconds. The expert rewards that were used for this task comprised of the following components: (i) Euclidean distance between the robot and goal, (ii) power used by the robot motors for linear and angular velocities, (iii) distance to obstacles to detect collisions, and (iv) an optional penalty if the robot exited the environment boundary walls. The PPO agent that was trained separately on the expert and PeGLearn rewards used the default architecture and hyperparameter settings from [51]. Both training sessions were run for 3e6 steps, with each session lasting about 30 hours on our hardware. Likewise, the D4PG agent was trained independently on each reward function, under similar training conditions (hyperparameters shown in Table 2), with each training session lasting about 12 hours. Each of these RL agents were then evaluated on 100 test runs (trials) to compute the success rates.

| Hyperparameter     | Value   |
|--------------------|---------|
| Actors             | 5       |
| Learners           | 1       |
| Actor MLP          | 256 $\rightarrow$ 256 |
| Critic MLP         | 256 $\rightarrow$ 256 |
| N-Step             | 5       |
| Atoms              | 51      |
| $V_{\text{min}}$   | -10     |
| $V_{\text{max}}$   | 10      |
| Exploration Noise  | 0.3     |
| Discount Factor    | 0.99    |
| Mini-batch Size    | 256     |
| Actor Learning Rate| 5e-4    |
| Critic Learning Rate| 5e-4 |
| Memory Size        | 1e6     |
| Learning Batch     | 64      |

### C.4 CARLA-AMT Survey

The demonstrations for this experiment used the same analog hardware for the 2D car simulator (Fig. 7). The formal description of the STL task specifications for the CARLA simulator are as follows:

1. Keeping close to the center of the lane:
   $\varphi_1 := G_{[0,T]}(d_{\text{lane}}[t] \leq \delta)$, where $T$ is the length of a demonstration, $d_{\text{lane}}[t]$ is the distance of car from the center of the lane at each step $t$ and $\delta$ is a small tolerance factor. The width of a typical highway lane in the US is 12 ft (3.66 m)\(^5\) and the average width of a big vehicle

\(^5\)Based on USDOT highway and US vehicle specifications (e.g., Ford F-150).
Teleoperation demonstrations in CARLA.

(e.g., SUV or pickup truck) is 7 ft (2.13 m), which leaves about 2.5 ft (0.76 m) of room on either side of the vehicle. Hence, we chose to use 1 ft (0.3 m) as the tolerance factor to accurately track the lane center while also providing a small room for error.

2. Maintaining speed limits:

\[ \varphi_2 := G_{[0,T]}(v_{\min} \leq v_t \leq v_{\max}) \]

where \( v_t \) is speed of the ego/host car at each timestep \( t \), and \( v_{\min} \) and \( v_{\max} \) are the speed limits. Since it is a US highway scenario, the \( v_{\max} = 65 \) mph and \( v_{\min} = 0 \) mph.

3. Maintaining safe distance from any lead vehicle:

\[ \varphi_3 := G_{[0,T]}(\text{safety\_flag}_t \leq 0) \]

where \( \text{safety\_flag}_t \) is a binary signal that outputs 0 if the ego is safe (i.e., there is no vehicle directly in front of the ego in the same lane whose distance is closer than some threshold \( d_{\text{safe}} \)) and 1 otherwise. In OpenAI Gym-CARLA, the safe distance was set to 15 m.

For the online AMT survey, we initially recruited 150 human participants and took numerous measures to ensure reliability of results. We posed a control question at the end to test their attention to the task, and eliminated data associated with the wrong answer, including incomplete data, resulting in 146 samples. All participants had an approval rating over 98% and the demographics are as follows: (i) 73 males, 72 females, 1 other, (ii) participant age ranged from 22 to 79 with an average age of 40.67, and (iii) average driving experience of 22.4 years. Our survey collected the following information from each participant:

- Participant information: Number of years of driving experience, age, gender and experience with video games.
- Ratings on a scale of 1 (worst) - 5 (best) for the queries/specifications: (i) driver staying close to the lane center, (ii) driver maintaining safe distance to lead vehicle(s) and (iii) driver respecting speed limits of the highway.
Figure 9: A frame from one of the survey videos.

Figure 10: Comparison of specification orderings between humans and PeGLearn.

- Ratings on a scale of 1 (lowest) - 3 (highest) on the overall driving behavior shown in these 5 videos and also how the participants would prioritize each of the specifications if they were driving in that scenario.

The web-based questionnaire showed a batch of 5 videos to a participant, where each video was accompanied by the three questions regarding the task specifications. A still of one of the videos is shown in Fig. 9. Users were presented with a dropdown menu in which each option was a Likert rating from 1 (lowest) to 5 (highest). We then presented users with a question to rate the overall behavior of all 5 videos in the batch w.r.t. the task specifications on a scale of 1 (lowest) to 3 (highest). Finally, a control question was posed regarding the color of the car shown in the videos to test the attention of the users, since the colors of cars were same across all 15 videos. The graphs for each batch obtained via PeGLearn is shown in Fig. 8.

The overall orderings from human experts and our PeGLearn algorithm for each AMT survey batch are shown in Fig. 10. To compare the ratings, we first normalized all the human and PeGLearn ratings to be in the range [0, 3]. The user bars correspond to the human expert ratings while auto represents our algorithm rating, which is deterministic and hence there are no error bars.

**D JIGSAWS (Surgical Robot Dataset)**

To show the generalizability of our method to performance assessment metrics beyond those induced by temporal logics, we evaluated it on human ratings (e.g., Likert scale) provided for various robotic surgical tasks [52] performed on a Da Vinci Surgical robot system. As described in the dataset, an expert surgeon provided evaluations or ratings for 8 different surgeons with various expertise levels on 3 basic surgical tasks - knot-tying, needle-passing and suturing. There are 6 specifications for evaluating the performance of surgeons on the tasks and the ratings are measured on a scale from 1 (lowest) to 5 (highest) for each specification. The 6 specifications or evaluation criteria are:

1. Respect for tissue (TR) - force exerted on tissue.
2. Suture/needle handling (SNH) - control while tying knots.
3. Time and motion (TM) - fluent motions and efficient.
4. Flow of operation (FO) - planned approach with minimum interruptions between moves.
5. Overall performance (P).
6. Quality of final product (Q).

Using this rating scheme, our method was able to generate a performance-graph for each class of expertise as shown in Fig. 11 for the knot-tying task. If the expertise levels were unknown, then the generated graph would be as indicated in Fig. 11d. We obtained similar performance graphs for the other 2 surgical tasks - needle-passing and suturing. This shows how human evaluations can be used in environments such as these where it is difficult even for experts to express the tasks in a formal language. From the graphs, we can see that all 3 categories of surgeons showed maximum performance on (Q) and (TM). However, there were differences in the other specifications. For example, experts had a higher rating of (P) over (TR) compared to intermediates. One possible explanation for this is that experts typically perform multiple consecutive surgeries, and so they optimize on the (P) and (FO) aspects compared to (TR), while the intermediate-level surgeons are trainees who are still learning the nuances of surgeries and are focusing more on the qualitative aspects such as (TR) over quantity and speed. Similar reasoning can be applied to each category of surgeons using these DAGs.

Remark. We acknowledge that providing individual ratings for every demonstration-specification pair is tedious since the complexity of manually specifying the performance graph is exponential as elaborated in Appendix B. This is because one needs to take into account, not just the labels or ratings, but also the orderings (permutations) among those labels. In other words, a user needs to assign \( O(mn) \) ratings and also compare them with different permutations of those ratings, i.e., creating relative priorities to specify the graph, which is exponential in \( O(n^2) \). Thus, manually defining the graphs results in a very large complexity as shown. Our method significantly eliminates this complex manual labor by using only minimal inputs from users as it is much easier to provide individual labels than having to compare with all permutations of the labels. To even further reduce human inputs, a potential solution we will consider for future work is to use deep temporal learning methods to learn from existing labeled data and predict the labels for newer demonstrations. We argue that some form of human feedback would be necessary to provide formal guarantees in learning rewards since it provides a ground truth baseline.
Figure 11: DAGs for the Knot-Tying task. (a)-(c) DAG for each level of expertise: Experts, Intermediates and Novices respectively. (d) DAG for all surgeons, without discriminating expertise levels.