Learning Performance Graphs from Demonstrations via Task-Based Evaluations
Supplemental Material

Aniruddh G. Puranic, Jyotirmoy V. Deshmukh, and Stefanos Nikolaidis

I. 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* μ that is a formula of the form $f(\mathbf{x}(t)) > 0$, where $\mathbf{x}(t)$ is the tuple $(state, action)$ of the demonstration $\mathbf{x}$ at time t, and f maps the signal domain $\mathcal{D} = (S \times A)$ 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: $\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{G}_I\varphi \mid \mathbf{F}_I\varphi \mid \varphi\mathbf{U}_I\varphi$. 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 $\mathbf{x}$ defined over some time-domain $\mathbb{T}$. The Boolean satisfaction of a signal predicate is simply *True* ($\top$) if the predicate is satisfied and *False* ($\bot$) if it is not, the semantics for the propositional logic operators $\neg, \wedge$ (and thus $\vee, \rightarrow$) follow the obvious semantics. The following behaviors are represented by the temporal operators:

- At any time t, $\mathbf{G}_I(\varphi)$ says that φ must hold for all samples in $t + I$.
- At any time t, $\mathbf{F}_I(\varphi)$ says that φ must hold *at least once* for samples in $t + I$.
- At any time t, $\varphi\mathbf{U}_I\psi$ says that ψ must hold at some time t' in $t + I$, and in $[t, t')$, φ must hold at all times.

Definition 1 (Quantitative Semantics for Signal Temporal Logic)**.** *Given an algebraic structure* $(\oplus, \otimes, \top, \bot)$*, we define the quantitative semantics for an arbitrary signal* $\mathbf{x}$ *against an STL formula* φ *at time* t *as in Table I.*

A signal satisfies an STL formula φ 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 φ, 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, [1], [2] 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: $\top = +\infty$, $\bot = -\infty$, and $\oplus = \max$, and $\otimes = \min$, as per the original definitions of robust satisfaction proposed in [1], [3].

TABLE I
Quantitative Semantics of STL

φ	$\rho(\varphi, \mathbf{x}, t)$
true/false	$\top/\bot$
μ	$f(\mathbf{x}(t))$
$\neg\varphi$	$-\rho(\varphi, \mathbf{x}, t)$
$\varphi_1 \wedge \varphi_2$	$\otimes(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t))$
$\varphi_1 \vee \varphi_2$	$\oplus(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t))$
$\mathbf{G}_I(\varphi)$	$\otimes_{\tau \in t+I}(\rho(\varphi, \mathbf{x}, \tau))$
$\mathbf{F}_I(\varphi)$	$\oplus_{\tau \in t+I}(\rho(\varphi, \mathbf{x}, \tau))$
$\varphi\mathbf{U}_I\psi$	$\oplus_{\tau_1 \in t+I}(\otimes(\rho(\psi, \mathbf{x}, \tau_1), \otimes_{\tau_2 \in [t, \tau_1)}(\rho(\varphi, \mathbf{x}, \tau_2))))$

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 1 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 [4]. We use the $tanh$ hyperbolic smoothing in our work to bound the robustness values.

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, USA {puranic, jdeshmuk, nikolaid}@usc.edu

Digital Object Identifier (DOI): 10.1109/LRA.2022.3226072

II. Derivations and Proofs

A. 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 $op = \{>, =\}$ 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 $n!\cdot|op|^{n-1} - n! + 1 = n!\cdot[|op|^{n-1} - 1] + 1$. 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 - no edge, incoming (outgoing from other) and outgoing (incoming to other). In the worst case, the maximum number of edges in a DAG is $n(n-1)/2$ edges and so the total number of possible directed graphs is $3^{n(n-1)/2}$. This includes all the cycles formed in the directed graph, so we then need to compute and subtract the number of cycles to obtain the actual space of DAGs. For a directed graph with n vertices, 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:

$$\begin{aligned}\sum_{k=3}^{n}\binom{n}{k} &= \sum_{k=0}^{n}\binom{n}{k} - \sum_{k=0}^{2}\binom{n}{k} \\ &= 2^n - [\frac{n!}{0!(n-0)!} + \frac{n!}{1!(n-1)!} + \frac{n!}{2!(n-2)!}] \\ &= 2^n - [1 + n + n(n-1)/2]\end{aligned}$$

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 $= 2^{n+1} - (n^2 + n + 2)$. 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. Proof of Lemma and Theorem

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

The lemma states: *For a DAG, the weights associated with the nodes computed via* (1)*, are non-negative.*

$$w(\varphi) = |\Phi| - |ancestor(\varphi)| \tag{1}$$

Proof Sketch. From the LfD-STL framework, the weights for specifications represented by the DAG nodes are given by (1). We know that $|\Phi| = n$ and $ancestor(\varphi)$ is a set whose cardinality is non-negative. In a DAG, there are no cycles and hence $|ancestor(\varphi)|$ is an integer in $[0, n-1]$. By this equation, the minimum weight (i.e., worst-case) for any node representing a specification φ occurs when that node is a leaf and all other $n-1$ nodes are its ancestors. Therefore, $w(\varphi) = |\Phi| - ancestor(\varphi) \implies w(\varphi) = n-(n-1) = 1 \geq 0$. Similarly, the maximum value of $w(\varphi)$ is n, i.e., there are no ancestors when φ 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 ξ_1 and ξ_2 in an environment, the partial ordering $\xi_1 \preceq \xi_2$ is preserved by PeGLearn.*

Proof. Recall that for any two demonstrations ξ_1 and ξ_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 $\boldsymbol{\rho_{\xi_i}} = [\rho_{i1}, \cdots, \rho_{in}]^T$. Let there be n specifications for the environment, then for these two demonstrations, we have:

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

and $\mathbf{w} = [w_1, w_2, ..., w_n]^T$. W.l.o.g., let ξ_2 be at least as good as ξ_1, i.e., we have $\rho_{1j} \leq \rho_{2j}, \forall j \in \{1, \cdots, n\}$. The cumulative scores for the demonstrations are $r_{\xi_i} = \boldsymbol{\rho_{\xi_i}}^T \cdot \mathbf{w}$ where $i \in \{1, 2\}$. For any constant $w_j \geq 0$,

$$\begin{aligned}&w_j \cdot \rho_{1j} \leq w_j \cdot \rho_{2j} \\ &\implies \sum_{j=1}^{n} w_j \cdot \rho_{1j} \leq \sum_{j=1}^{n} w_j \cdot \rho_{2j} \\ &\implies \boldsymbol{\rho_{\xi_1}}^T \cdot \mathbf{w} \leq \boldsymbol{\rho_{\xi_2}}^T \cdot \mathbf{w} \\ &\implies r_{\xi_1} \leq r_{\xi_2} \\ &\text{This holds iff } w_j \geq 0, \forall w_j \in \mathbf{w}\end{aligned}$$

Once the global DAG is learned, the weights for specifications (nodes) are computed via (1). 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. □

III. Additional Details on Experiments

The STL formulas in our discrete-world and 2D driving experiments were specified and evaluated using Breach [5], and the specifications for the MiR100 and CARLA experiments were evaluated in RTAMT-STL library [6]. 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 [7]. 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 [8] 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.

A. Discrete-World

We use a grid environment, based on the OpenAI Gym $Frozenlake$ environment, consisting of a set of states $S = \{start, goals, obstacles\}$ 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 [8], [9], we used *Manhattan* distance as the distance metric and formulated the STL specifications:

1) Avoid obstacles at all times (hard requirement): $\varphi_{\mathbf{1}} := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_{obs}}[\mathbf{t}] \geq \mathbf{1})$, where T is the length of a demonstration and d_{obs} is the minimum distance of robot from obstacles computed at each step t.
2) Eventually, the robot reaches the goal state (soft requirement): $\varphi_{\mathbf{2}} := \mathbf{F}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_{goal}}[\mathbf{t}] < \mathbf{1})$, where d_{goal} is the distance of robot from goal computed at each step. φ_2 depends on φ_1.
3) Reach the goal as fast as possible (soft requirement): $\varphi_{\mathbf{3}} := \mathbf{F}_{[\mathbf{0},\mathbf{T}]}(\mathbf{t} \leq \mathbf{T_{goal}})$, 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. φ_3 depends on both φ_1 and φ_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 [10], as it is suited for stochastic settings, with the modifications to the algorithm at 2 steps (reward update and termination) as described by [9]. The number of episodes varied based on environment complexity such as grid size, number and locations of obstacles. The discount factor γ was set to 0.8 and ϵ-greedy strategy with decaying ϵ 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.

B. 2D Driving Simulator

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

$$\dot{x} = v \cdot cos(\theta) + \mathcal{N}(0, \sigma^2);\ \dot{y} = v \cdot sin(\theta) + \mathcal{N}(0, \sigma^2)$$
$$\dot{v} = u_1 \cdot u_2;\ \dot{\theta} = v \cdot tan(\psi);\ \dot{\psi} = u_3$$

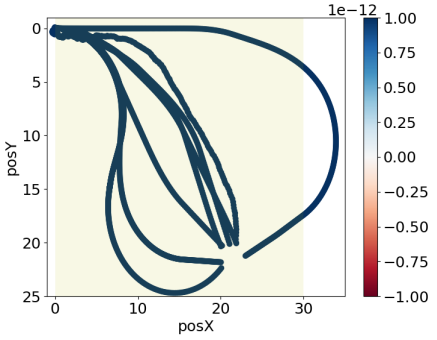


Fig. 1. Demonstrations collected for the 2D car simulator.

where x and y represent the XY position of the car; θ 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 ψ. The state of the car at time t is given by $S_t = [x, y, \theta, v, \dot{x}, \dot{y}, \dot{\theta}, \dot{v}]^T$. 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. 1).

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): $\varphi_{\mathbf{1}} := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_{obs}}[\mathbf{t}] \geq \mathbf{dSafe})$, where T is the length of a demonstration and d_{obs} is the minimum distance of the car from $\mathcal{H}$ computed at each step t. For our experiments, we used $\texttt{dSafe} = 3$ units.
2) Always stay within the workspace/drivable region (hard requirement): $\varphi_{\mathbf{2}} := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}((\mathbf{x}, \mathbf{y}) \in \mathbf{Box}(\mathbf{30}, \mathbf{25}))$, where the workspace is defined by a rectangle of dimensions 30×25 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): $\varphi_{\mathbf{3}} := \mathbf{F}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_{goal}}[\mathbf{t}] < \delta)$, where d_{goal} is the distance between centers of car and goal computed at each step t and δ is a small tolerance when the center of the car is "close enough" to the goal's center. φ_3 depends on φ_1 and φ_2 in the DAG.

The rewards are assigned to states by modeling the states s as samples of multi-variate Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$ where $\mu = s$ and σ 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 [8]. 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. 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

Fig. 2. Teleoperation demonstrations in CARLA.

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 [11]. 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 II), 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.

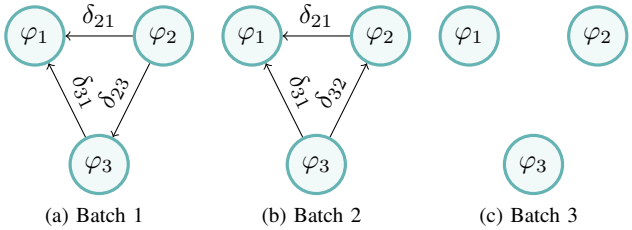
TABLE II
D4PG HYPERPARAMETERS.

Hyperparameter	Value
Actors	5
Learners	1
Actor MLP	256 $\rightarrow$ 256
Critic MLP	256 $\rightarrow$ 256
N-Step	5
Atoms	51
V_{min}	-10
V_{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

D. CARLA-AMT Survey

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

1) Keeping close to the center of the lane:
$\varphi_{\mathbf{1}} := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}(\mathbf{d_t^{lane}} \leq \delta)$, where T is the length of a demonstration, d_t^{lane} is the distance of car from the center of the lane at each step t and δ is a small tolerance factor. The width of a typical highway lane in the US is 12 ft (3.66 m) [1] and the average width of a big vehicle (e.g., SUV or pickup truck) is 7 ft (2.13 m) [1], 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:
$\varphi_{\mathbf{2}} := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}(\mathbf{v_{min}} \leq \mathbf{v_t} \leq \mathbf{v_{max}})$, 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:
$\varphi_{\mathbf{3}} := \mathbf{G}_{[\mathbf{0},\mathbf{T}]}(\mathbf{safety\_flag_t} \leq \mathbf{0})$,
where $\texttt{safety\_flag}_\texttt{t}$ 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_{safe}) and 1 otherwise. In OpenAI Gym-CARLA, the safe distance was set to `15 m`.



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

Fig. 3. DAGs for the CARLA simulator experiment.

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.
- 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. 4. Users were presented with a dropdown menu in which each option was a *Likert* rating

[1]Based on USDOT highway and US vehicle specifications (e.g., Ford F-150).

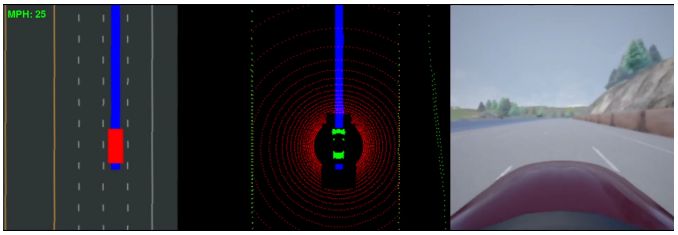


Fig. 4. A frame from one of the survey videos.

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. 3.

The overall orderings from human experts and our PeGLearn algorithm for each AMT survey batch are shown in Fig. 5. 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.

IV. 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 [12] 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:

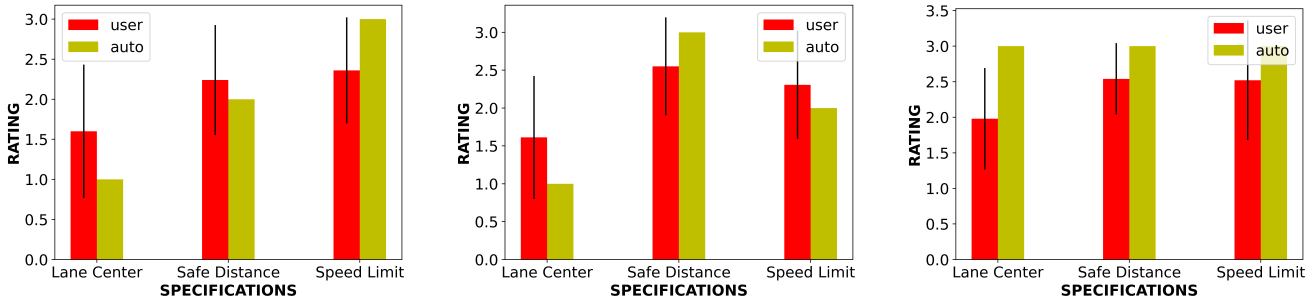
1) Respect for tissue (TR) - force exerted on tissue.
2) Suture/needle handling (SNH) - control while tying knots.
3) Time and motion (TM) - fluent motions and efficient.
4) Flow of operation (FO) - planned approach with minimum interruptions between moves.
5) Overall performance (P).
6) Quality of final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

Using this rating scheme, our method was able to generate a performance-graph for each class of expertise as shown in Fig. 6 for the *knot-tying* task. If the expertise levels were unknown, then the generated graph would be as indicated in Fig. 6d. 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 II. 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 $\mathcal{O}(mn)$ 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.*

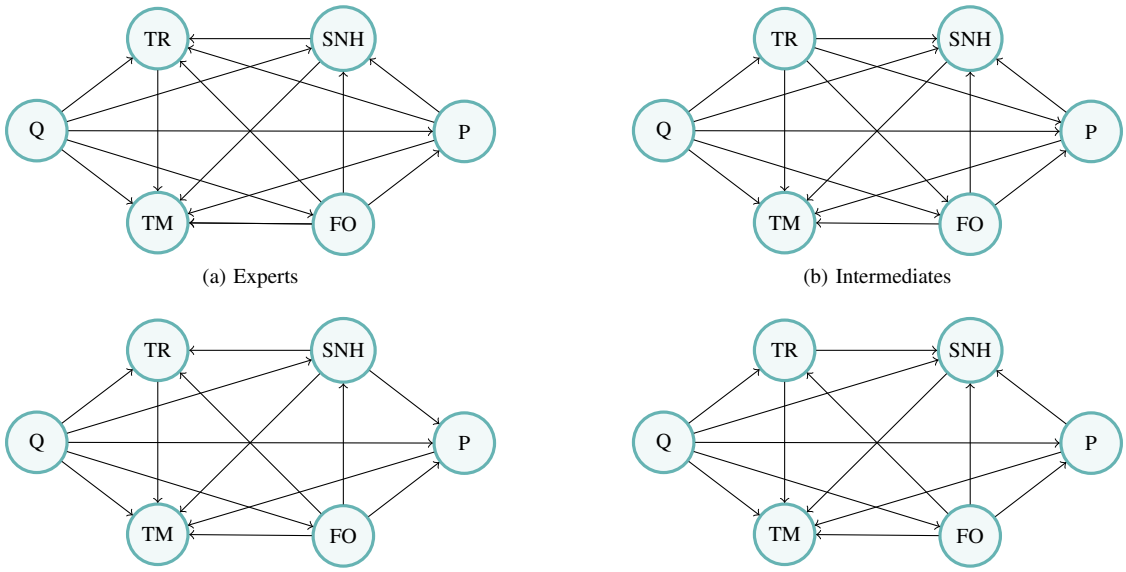
REFERENCES

[1] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, 2009.
[2] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković, "Quantitative monitoring of STL with edit distance," *Formal Methods in System Design*, vol. 53, no. 1, 2018.
[3] 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.
[4] A. Dhonthi, P. Schillinger, L. D. Rozo, and D. Nardi, "Study of signal temporal logic robustness metrics for robotic tasks optimization," *CoRR*, vol. abs/2110.00339, 2021.
[5] A. Donzé, "Breach, A toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification CAV*, 2010.
[6] D. Ničković and T. Yamaguchi, "Rtamt: Online robustness monitors from stl," in *Automated Technology for Verification and Analysis*, 2020.
[7] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166.
[8] 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.
[9] ——, "Learning from demonstrations using signal temporal logic," in *CoRL*, 2020.
[10] H. Hasselt, "Double q-learning," in *NIPS*, 2010.
[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html
[12] 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



(a) Batch A (b) Batch B (c) Batch C

Fig. 5. Comparison of specification orderings between humans and PeGLearn.



(a) Experts (b) Intermediates

(c) Novices (d) Cumulative

Fig. 6. 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.

Modeling and Monitoring of Computer Assisted Interventions (M2CAI) – MICCAI Workshop, 2014.