2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Signal Temporal Logic-Guided Apprenticeship Learning

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

Abstract— **Apprenticeship learning crucially depends on effectively learning rewards, and hence control policies from user demonstrations. Of particular difficulty is the setting where the desired task consists of a number of sub-goals with temporal dependencies. The quality of inferred rewards and hence policies are typically limited by the quality of demonstrations, and poor inference of these can lead to undesirable outcomes. In this paper, we show how temporal logic specifications that describe high level task objectives, are encoded in a graph to define a temporal-based metric that reasons about behaviors of demonstrators and the learner agent to improve the quality of inferred rewards and policies. Through experiments on a diverse set of robot manipulator simulations, we show how our framework overcomes the drawbacks of prior literature by drastically improving the number of demonstrations required to learn a control policy.**

I. INTRODUCTION

Recent advances in robotics have led to the development of algorithms that extract control policies for autonomous agents from human demonstrations via the paradigm of learning-from-demonstrations (LfD). An interesting sub-area of LfD is the use of demonstrations alongside reinforcement learning (RL) to either (i) initialize policies for the RL agent [1] via behavior cloning (BC) [2] or (ii) infer rewards using inverse RL (IRL) [3] for tasks from which policies can be extracted - apprenticeship learning via IRL [4]. However, designing rewards for Markov Decision Processes (MDPs) [5] is non-trivial and typically requires expert knowledge in designing reward functions that can ensure safety and efficiency in the extracted RL policies. More importantly, for robots to be robust to perturbations in the environment, it is crucial to capture the overall goals/intentions of demonstrators, i.e., via IRL, rather than merely mimicking them [4]. Our work draws inspiration from Apprenticeship Learning (AL) [4] to learn both rewards and policies.

A drawback of AL is that it relies on demonstrations being optimal, which is seldom the case in real-world scenarios. More recent IRL and BC-based methods that learn from suboptimal demonstrations [6]–[10] measure optimality or performance based on statistical noise deviation from the true/optimal demonstrations. However, such noise-to-performance measures are extracted *empirically* and hence lack formal reasoning that can explain the quality of behaviors. Furthermore, as the core reward-inference algorithm in AL uses IRL, the rewards are inherently Markovian, and they do not account for temporal dependencies among subgoals in demonstrations. Research in reward design [11], [12] discusses the need for non-Markovian reward representations, especially in time-dependent multi-goal RL settings. Such non-Markovian rewards are typically designed using spilt-MDPs [13] and reward machines [14], [15], which require increasing the state and/or action spaces of the MDPs significantly thereby increasing the space and computational complexities for the underlying RL algorithms.

To address these limitations, our prior work [16], [17] has proposed to use Signal Temporal Logic (STL) to define high-level tasks, and evaluate and rank demonstrations to infer rewards. The semantics of STL measure the quality/fitness, which is the degree of task satisfaction by demonstrations. This facilitates holistic temporal-based ranking of demonstrations and agent behaviors to formulate non-Markovian rewards. Our LfD-STL framework can learn from only a handful of even imperfect/suboptimal demonstrations, *without the need to augment the MDP spaces*. It has shown to significantly outperform state-of-the-art IRL methods in terms of reward quality, number of demonstrations required and safety of the learned policy. It can also be applied to stochastic and continuous spaces to extract rewards and behaviors consistent with the task specifications. Our recent work proposed PeGLearn [18] to automatically infer non-Markovian rewards for tasks comprising multiple STL objectives, addressing the representation issues discussed in [12]. PeGLearn uses directed graphs to create a partial ordering of specifications to produce a single graph - *performance graph* - that holistically captures the demonstrated behaviors.

While the LfD-STL framework with PeGLearn can offer assurances in safety of the learned rewards and policy, it does not explicitly reason about optimality of inferred rewards and performance of the learned RL policy. The reason being that LfD-STL is an open-loop framework where the inferred rewards are fixed and are not guaranteed to be optimal without any exploration. Without feedback from agent exploration, it may be impossible to discover better behaviors. We aim to address this issue by using the performance graph as a metric, which we refer to as the *performance-graph advantage (PGA)* to guide the RL process. We propose the AL-STL framework that addresses this issue with LfD-STL, by integrating closed-loop learning wherein, both the reward function and policy are updated iteratively. PGA can be interpreted as the quantification of the areas for improvement of the policy, and is optimized alongside appropriate existing

The authors are with the Department of Computer Science, University of Southern California, USA. Email: {puranic, jdeshmuk, nikolaid}@usc.edu.

The authors gratefully acknowledge the support from the National Science Foundation (NSF) under the following grants: CCF-2048094, CNS-2039087, CNS-1932620, and CCF-1837131, support from Toyota R&D and Siemens R&D through the USC Center for Autonomy and AI, support from the USC Airbus Institute for Engineering Research and partial support from the Agilent Early Career Professor Award.

RL algorithms. This enables reasoning about possibly new behaviors that were not demonstrated before, but still satisfy the task specifications. The key insight of our work is that *a cumulative/collective measure of (multiple) task objectives along with exploration in the neighborhood of observed behaviors guides the refinement of rewards and policies that can extrapolate beyond demonstrated behaviors*. Our contributions are summarized as follows:

1) We propose AL-STL, a novel successor to the LfD-STL framework, introducing closed-loop learning by improving both the reward function and policy significantly.
2) We quantify STL-based performance graphs learned via PeGLearn in terms of an advantage function to guide the RL training process, and formally reason about policy improvements when demonstrations are suboptimal.
3) We evaluate our approach on a variety of robotic manipulation tasks and discuss how our framework outperforms state-of-the-art literature.

II. Related Works

Learning-from-demonstrations (LfD) to extract control policies can be broadly classified into two main categories based on the underlying intentions: (i) imitation learning (IL), such as behavior cloning (BC) via supervised learning [2], where the objective is to directly mimic the actions of the demonstrators, and (ii) inverse reinforcement learning (IRL) [3], [4], [6], where the objective is to characterize the overall goal or intent of the demonstrators via cost/reward functions.

Learning rewards via entropy-enabled IRL [6], [7], [19], [20] regard suboptimal demonstrations as noisy deviations from the optimal statistical model, and hence require access to many demonstrations. Learning better policies from suboptimal demonstrations has been explored in [10]. This method injects noise into trajectories to infer a ranking, however it synthetically generates trajectories via BC which has issues with covariate shift and induces undesirable bias. This is addressed in [9] by defining a relation between injected noise and performance. However, this noise-performance relationship is empirically derived and lacks formal reasoning. Score-based IRL [8] uses expert-scored trajectories to learn a reward function, relying on a large set of nearly-optimal demonstrations and hence generating scores for each of them. Additionally, rewards learned via IRL-based methods are Markovian by nature and typically suited to single-goal tasks, as discussed in prior work [16], [17].

In the area of LfD with temporal logics, the closest to our work is a counterexample-guided approach using probabilistic computation tree logics (PCTL) for safety-aware AL [21]. Our work differs from it in two significant ways: (i) we use STL which is applicable to continuous spaces and offers timed-interval semantics, which are lacking in PCTL, and (ii) the reward inference algorithm in [21] relies on IRL, while ours is based on LfD-STL [18], which greatly improves sample complexity, accuracy and inference speed. Trade-offs for multi-objective RL have been explored in [22] by explicitly defining specification priorities beforehand. Alternate approaches convert specifications to their equivalent automaton and augment it to the MDP states [23]–[25]. In our work, we do not alter the MDP structure, thereby avoiding the drawbacks of increased space and computational complexities of augmented MDPs.

III. Preliminaries

A. Mathematical Notations

The interactions between the agent (robot) and the environment are modeled with a Markov Decision Process.

Definition 3.1 (Markov Decision Process (MDP)): An MDP is given by a tuple $M = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where $\mathcal{S} \subset \mathbb{R}^k$ is the state space and $\mathcal{A} \subset \mathbb{R}^l$ is the action space of the system; T is the transition function, where $T(s, a, s') = Pr(s' \mid s, a)$; R is a reward function that typically maps either some $s \in \mathcal{S}$, state-action pair $\mathcal{S} \times \mathcal{A}$ or some transition $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ to $\mathbb{R}$.

The goal of RL is to find a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximizes the total (discounted) reward from performing actions on an MDP, i.e., the objective is to compute $\max \sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the output of the reward function R at time t and γ is the discount factor. In this paper, we assume full observation of the state space for MDPs.

Definition 3.2 (Trajectory or Episode Rollout): A trajectory τ in an MDP is a sequence of state-action pairs of finite length $L \in \mathbb{N}$ by following some policy π from an initial state s_0, i.e., $\tau = \langle s_0, a_0, \cdots, s_L \rangle$, $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}$.

In our LfD setting, the demonstrations are collected on the robot itself (e.g., via teleoperation or kinesthetic teaching), so the observations are elements of the MDP state and action spaces. Hence, we interchangeably refer to trajectories or rollouts as demonstrations. For intuition, we use demonstrations to refer to rollouts provided to the RL agent as inputs, and represent a demonstration by ξ.

Prior work in LfD [16]–[18] uses Signal Temporal Logic (STL) [26], [27] to define high-level tasks.

Signal Temporal Logic (STL): 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 (s, a) of the trajectory $\mathbf{x}$ at time t, and f maps the signal domain $\mathcal{D} = (\mathcal{S} \times \mathcal{A})$ 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.

The quantitative (robustness) semantics of STL, defined in [27], [28], capture the performance of trajectories. Directed acyclic graphs are used to encode the preferences or performance of the demonstrators. Such graphs provide a convenient way to interpret reward functions for RL tasks.

Definition 3.3 (Directed Acyclic Graph (DAG)): A directed graph is an ordered pair $G = (V, E)$ where V is a set of elements called nodes and E is a set of ordered pairs of nodes called edges, which are directed from one node to another. An edge $e = (u, v)$ is directed from node u to node v. A *DAG* is a directed graph that has no directed cycles, i.e., it can be topologically ordered.

A path $x \rightsquigarrow y$ in G is a set of nodes starting from x and ending at y by following the directed edges from x. The ancestors of a node v is the set of all nodes in G that have a path to v. Formally, $ancestor(v) = \{u \mid u \rightsquigarrow v, u \in V\}$. In our setting, we use a weighted DAG, where each node $v \in V$ is associated with a pair of real numbers - value and weight of the node, represented by $\nu(v)$ and $w(v)$ respectively. Each edge $(u, v) \in E$ is associated with a real number - weight of the edge, represented by $w(u, v)$. Note the difference in number of arguments in these notations.

B. Reward Inference from Demonstrations and STL

In LfD-STL, the reward function R of the MDP is unknown, instead, it is presented with a finite set of high-level task descriptions in STL $\Phi = \{\varphi_1, \varphi_2, \cdots, \varphi_n\}$ and a finite set of demonstrations $\Xi = \{\xi_1, \xi_2, \cdots, \xi_m\}$, from which the reward function and policy must be inferred.

LfD-STL Framework: For a specification $\varphi \in \Phi$ and a demonstration $\xi \in \Xi$ defined as in Def. 3.2, the value $\rho(\varphi, \xi, t)$ represents how well the demonstration satisfied the given specification from time t, which is the quality of the demonstration. To evaluate the entire trajectory, the robustness is defined at $t = 0$, i.e. $\rho(\varphi, \xi, 0)$ and is implicitly denoted by $\rho(\varphi, \xi)$. For a demonstration ξ, we have an array of evaluations over Φ, given by $\hat{\rho}_\xi = [\rho(\varphi_1, \xi), \cdots, \rho(\varphi_n, \xi)]^T$.

Then, for each $\xi \in \Xi$, a local DAG G_ξ is initially constructed via the PeGLearn algorithm [18], wherein, (i) each task specification $\varphi \in \Phi$ is represented by a node, with the value of the node indicating the fitness of ξ for φ, i.e., $\nu(\varphi) = \rho(\varphi, \xi)$, and (ii) the edges, along with their corresponding weights encode information about the preferences or performance between every pair of specifications as exhibited by the behavior. For any edge $e(\varphi_i, \varphi_j)$, its weight, defined by $\nu(\varphi_i) - \nu(\varphi_j)$, indicates a measure by which the value of φ_j must be increased to match the value of φ_i. As an edge in G_ξ is always directed from a higher-valued node to a lower-valued node, the edge weight is always positive. Absence of an edge between a pair of nodes indicates a zero-weighted edge. *Note that this local DAG is applicable to all trajectories that conform to Def. 3.2.* Thus, PeGLearn maps a trajectory τ and Φ to a DAG G_τ. In our work, since specifications can be of different scales (e.g., a specification that monitors acceleration, while another monitors distance), we assume that the robustness bounds are known apriori and we normalize/scale the robustness values to be bounded to some $[-\Delta, \Delta]$. Scaling of robustness can be achieved with piece-wise linear functions or smooth semantics [29]. In addition to extracting a DAG for each trajectory, PeGLearn also captures the holistic behavior of a set of trajectories by aggregating their corresponding local DAGs into a global DAG $\mathcal{G}$. The nodes in $\mathcal{G}$ are weighted to capture the relative pair-wise priorities of specifications based on the node ancestors or dependencies via $w(\varphi) = |\Phi| - ancestor(\varphi)$, illustrated with an example in Fig. 1.

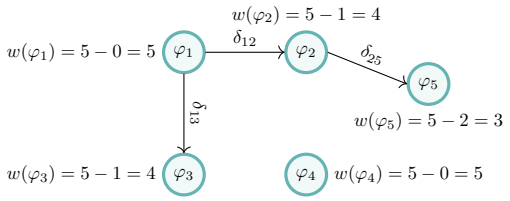


Fig. 1. Weights on nodes (specifications) in a DAG.

The node weights are used to induce bias towards specifications during inference of the reward function and hence the RL policy. Prior literature in behavior modeling with reward functions [6], [7], [9] has shown that the performance variations in trajectories obey an exponential form. So, the weights of the specifications from the DAG are normalized with `softmax` to ensure $\sum_{i=1}^{n} w(\varphi_i) = 1$. We now have a weight vector $w_\Phi = [w(\varphi_1), w(\varphi_2), \cdots, w(\varphi_n)]^T$. Each demonstration is then assigned a cumulative robustness/fitness value based on these weights, given by $r_\xi = \hat{\rho}_\xi{}^T \cdot w_\Phi$. To generalize, all trajectories are associated with a corresponding performance DAG and cumulative fitness. Once the cumulative fitness is assigned to each demonstration, the demonstrations are ranked based on their r_ξ and the rank-scaled rewards are propagated to the observations via the reward inference method described in [17]. In short, the method assigns monotonically increasing rewards (i.e., partial cumulative fitness) to the observed states and/or actions in demonstrations that satisfy the specification, while negative rewards are assigned to states in demonstrations that violate the task specifications.

IV. Methodology

A. Problem Formulation

For an MDP\R, we are given: (i) a finite dataset of demonstrations $\Xi = \{\xi_1, \xi_2, ..., \xi_m\}$ and (ii) a set of specifications $\Phi = \{\varphi_1, \varphi_2, ..., \varphi_n\}$ unambiguously expressing the tasks to be performed. The objective is to infer rewards and extract a behavior or control policy for an agent such that its behavior is at least as good or better than the demonstrations, and maximizes the satisfaction of the task specifications. The satisfaction of task specifications is conveyed through the learned reward function that the RL agent seeks to maximize.

More formally, consider a policy π under the reward function R that captures the degree of satisfaction of Φ. Let τ indicate a trajectory obtained by a rollout of π in an RL episode. Then, our objective is to find

$$\pi^*, R^* = \underset{\pi,R}{\operatorname{argmax}} \mathbb{E}_{\tau\sim\pi} \sum_{i=1}^{n} \rho(\varphi_i, \tau)$$

Since every trajectory τ is characterized by its associated performance DAG G_τ, where the value of a node indicates the robustness for the specification it represents (Sec. III-B), the summation term is the sum of all nodes. We thus define $\texttt{VS}_\tau \doteq \sum_{i=1}^{n} \nu(\varphi_i) = \sum_{i=1}^{n} \rho(\varphi_i, \tau)$. Then the objective is:

$$\pi^*, R^* = \underset{\pi,R}{\operatorname{argmax}} \mathbb{E}_{\tau\sim\pi}[\texttt{VS}_\tau]$$

An issue with this formulation occurs when there are multiple task specifications ($n > 1$). This results in multi-objective learning, which can introduce conflicting specifications and hence requires optimal trade-offs. For example, in autonomous driving or robot manipulation, consider the task of reaching a goal location as quickly as possible while avoiding obstacles. Depending on the obstacle locations, performing highly safe behaviors (i.e., staying as far away from obstacles as possible) might affect the time to reach the goal. Similarly, a behavior that aims to reach the goal in the least time will likely need to compromise on its safety robustness. We thus need to find the behaviors that not only maximize the total robustness, but are also maximally robust to each task specification. We illustrate this with Example 1.

Example 1: Consider a task with three specifications $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$, and consider two trajectories τ_1 and τ_2 with robustness vectors $[3, 0, 1]$ and $[2, 1, 1]$, respectively. The reward function inferred with τ_1 will have the weight for φ_1 dominate φ_2 due to the exponential `softmax` component (Sec. III-B), while the reward function for τ_2 will have more uniform weights over all specifications, albeit with a little bias towards φ_1 versus others. Thus, while both have the same $\texttt{VS}$ ($= 4$), τ_2 is holistically more robust w.r.t. all the task specifications due to better trade-offs.

By this reasoning, it is more desirable to not only maximize the overall sum, but also maximally satisfy the individual specifications with trade-offs. So, how do we ensure that optimal trade-offs are achieved while maximizing the main objective? By observation, it is straight-forward to deduce that the sum of *absolute* pair-wise differences in robustness of specifications must be minimized. This sum is indeed exactly encoded by the edges of our trajectory DAG (performance graph) formulation, which is a unique characteristic. Recall that the edges between two nodes (specifications) indicate the difference in their robustness values (performance). We thus capture the optimal trade-offs for a trajectory τ with the sum of all edges in its corresponding DAG G_τ, which is given by $\texttt{ES}_\tau = \sum_{e\in G_\tau} e$; each edge is created as in Sec. III-B. Both $\texttt{VS}$ and $\texttt{ES}$ can be computed in *linear time* using the same DAG, without additional computational overhead. One might wonder if merely minimizing $\texttt{ES}$ is sufficient for finding the optimal trade-offs. We provide a counterargument in Example 2.

Example 2: Consider the same task from Example 1, but with two different trajectories τ_3 and τ_4 with robustness vectors $[1, 1, 1]$ and $[-1, -1, -1]$, respectively. Since all the specifications are equally weighted, the $\texttt{ES}$ for both trajectories are the same ($= 0$). But clearly, τ_3 is more robust than τ_4 due to the higher $\texttt{VS}$. Furthermore, consider another trajectory τ_5 with vectors $[-1, 2, -1]$, whose $\texttt{ES}$ is 6 (i.e., an edge weight is the absolute pair-wise difference). Between τ_4 and τ_5, the RL agent will prefer τ_4 due to the lower $\texttt{ES}$, which is undesirable.

From both examples, we conclude that the objective is to maximize $\texttt{VS}$ while minimizing $\texttt{ES}$. Our new formulation is,

$$\pi^*, R^* = \underset{\pi,R}{\operatorname{argmax}} \mathbb{E}_{\tau\sim\pi}(\texttt{VS}_\tau - \texttt{ES}_\tau)$$

As both $\texttt{VS}$ and $\texttt{ES}$ are dependent on each other, this optimization trade-off can be written as:

$$\pi^*, R^* = \underset{\pi,R}{\operatorname{argmax}} \mathbb{E}_{\tau\sim\pi}(\texttt{VS}_\tau - \lambda \cdot \texttt{ES}_\tau) \tag{1}$$

The constant $\lambda \in [0, 1)$ acts as a regularizer to penalize behaviors with dominant specifications as in Example 1, and is a tunable hyperparameter. The formulation is very intuitive because we want to extract the optimal DAG which has no edges. Recall that edges are added only if there is a difference between the node values (i.e., robustness). Ideally, if the policy is optimal, then every rollout has the same maximum robustness for all Φ and so no edges are created. This representation offers the unique ability of providing an intuitive graphical representation of behaviors for interpretability [18], and formulating an optimization problem. As the robustness of each specification is bounded in $[-\Delta, \Delta]$, the $\texttt{VS}$ for any trajectory is bounded to $[-n\Delta, n\Delta]$. At either limit, the $\texttt{ES}$ is 0, indicating that all nodes in the resulting global DAG $\mathcal{G}$ have equal weights ($= 1/n$) at the extrema. We will refer to the term $(\texttt{VS}_\tau - \lambda \cdot \texttt{ES}_\tau)$ as *performance graph advantage (PGA)*. PGA indicates the scope for improvement (extra possible rewards) under the current reward function and policy. During RL, this PGA can be either used as a bonus term alongside the episode returns or augmented in the gradient ascent formulation.

B. AL-STL Framework

We now describe our proposed framework, shown in Fig. 2, that closes the RL training loop to extract both the reward function and policy that optimally satisfy Φ, resembling apprenticeship learning. The corresponding pseudocode is given in Alg. 1. Analogous to the replay buffer in RL, we introduce storage buffers for the reward model: (i) *frontier* $\mathcal{F}$ containing the best episode rollouts of the agent so far and (ii) *candidate* $\mathcal{C}$ containing the rollouts under the current reward and policy with PGAs. Initially, the frontier is populated with demonstrations (line 2) from which the global DAG $\mathcal{G}$ and hence the reward function are extracted via PeGLearn [18] (line 5). RL is performed with the learned rewards and each rollout is associated with its PGA, that is optimized

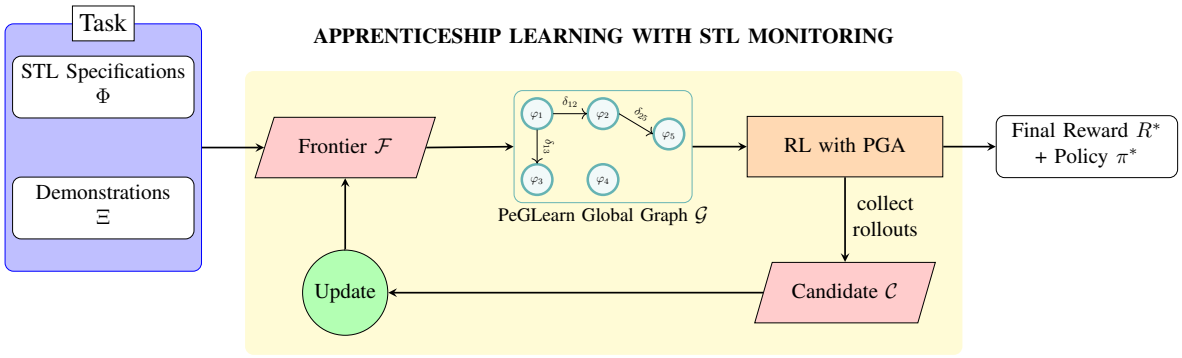


Fig. 2. AL-STL Framework with Performance-Graph Advantage.

either in the episode returns or in the gradient ascent. Upon updating the policy, multiple rollouts are collected in the candidate buffer (loop on line 8), and the frontier is updated by comparing the overall PGAs of both the frontier and candidate based on a strategy (line 11) that we describe in Sec. IV-B.1. This loop, shown by the yellow background in Fig. 2, continues for a finite number of cycles or until the frontier can no longer be updated. At this stage, the reward and policy representing the frontier optimally satisfy Φ, which we discuss in Sec. IV-B.2.

Algorithm 1: STL-Guided Apprenticeship Learning

Input: $\Xi :=$ demonstrations; $\Phi :=$ specifications
Result: $R^* :=$ reward function; $\pi^* :=$ a policy

1 **begin**
2 $\quad \mathcal{F} \leftarrow \Xi$ `// Initialize frontier`
3 $\quad converged = \bot$
4 $\quad$ **while** $\neg converged$ **do**
5 $\quad\quad R \leftarrow \texttt{PeGLearn}(\mathcal{F}, \Phi)$ `// reward function from rollouts in` $\mathcal{F}$
6 $\quad\quad \mathcal{C} \leftarrow \emptyset$ `// Initialize candidate`
7 $\quad\quad \pi \leftarrow$ perform RL with `PGA`
$\quad\quad$ `// Rollout` k `trajectories from` π `and add them to` $\mathcal{C}$
8 $\quad\quad$ **for** $i \leftarrow 1$ **to** k **do**
9 $\quad\quad\quad \tau_i \leftarrow \langle (s_t, a_t \sim \pi(s_t)) \rangle_{t=0}^{T}$
10 $\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \tau_i$
11 $\quad\quad converged \leftarrow \texttt{Update}(\mathcal{C}, \mathcal{F})$
12 $\quad$ **return** $R^* = R, \pi^* = \pi$

1) Frontier Update Strategies: $\mathcal{F}$ and $\mathcal{C}$ contain rollouts that are associated with their PGAs. We define an operator $\odot \in \{\texttt{min}, \texttt{max}, \texttt{mean}\}$, and therefore, the metrics $\widehat{\mathcal{F}} \doteq \odot\{\texttt{PGA}(\tau) | \tau \in \mathcal{F}\}$ and $\widehat{\mathcal{C}} \doteq \odot\{\texttt{PGA}(\tau) | \tau \in \mathcal{C}\}$. To update the frontier, we propose the *strategic merge* operation as:

(a) We first compare whether $\widehat{\mathcal{C}} > \widehat{\mathcal{F}}$, i.e., the trajectories with the newly-explored PGA are better than the current best trajectories in $\mathcal{F}$. The operator $\odot$ acts as the criterion for filtering bad-performing trajectories.
(b) If so, we retain the trajectories in $\mathcal{F} \cup \mathcal{C}$ whose PGAs are greater than $\widehat{\mathcal{F}}$ and discard the others; resulting trajectories form the new $\mathcal{F}$. Formally, this is given by $\mathcal{F} \leftarrow \{\tau | \texttt{PGA}(\tau) > \widehat{\mathcal{F}}, \tau \in \mathcal{F} \cup \mathcal{C}\}$. That is, quality of the worst $\odot$ criteria-based rollouts in $\mathcal{F}$ is improved.
(c) Otherwise, $\mathcal{F}$ already has the best trajectories so far and is left unaltered. If the statistic $\odot$ for $\mathcal{F}$ and $\mathcal{C}$ are similar (i.e., their difference is below some threshold) upon sufficient exploration, then convergence is achieved.

In theory, with unbounded memory, the frontier would be able to keep all the best-performing trajectories. For practical implementations, both buffers are bounded (say p), so we keep the top-p trajectories in the frontier in our experiments. The *strategic merge* is not the only way to maintain the buffer, however, it offers some performance guarantees as we show in Sec. IV-B.2. One could consider a naïve approach of simply merging all the trajectories in both buffers without any filtering criteria. Alternately, one could also replace all the trajectories in $\mathcal{F}$ with those in $\mathcal{C}$, which also exhibits monotonic improvement in the RL policy.

2) Policy Improvement Analysis: In order to analyze Alg. 1 and show policy improvement, we make certain assumptions about the task and RL models:

(a) The specifications accurately represent the task.
(b) The task can be completed, regardless of optimal behavior, with the given MDP configurations and task specifications. Our algorithm requires *at least one* demonstration that can satisfy all specifications, but is not required to be optimal.
(c) The RL agent always has an active exploration component (stochastic policy or an exploration rate) to cover the MDP spaces. This not only helps in discovering new policies, but also helps learn more accurate reward models. Theoretically, with infinite timesteps, the RL agent will have fully explored the environment spaces to find the optimal policy [5]. In practice, the timesteps are set to a large finite value for majority coverage of the spaces.

Here, we describe how the *strategic merge* functionality exhibits policy improvement. From Sec. IV-B.1, the new $\mathcal{F}$ contains the set of trajectories given by $\mathcal{F} = \{\tau | \texttt{PGA}(\tau) > \widehat{\mathcal{F}}, \tau \in \mathcal{F} \cup \mathcal{C}\}$. For the purpose of this proof, we will consider $\odot$ to be the `mean`. Then, $\widehat{\mathcal{F}} = \frac{\sum_{\tau \in \mathcal{F}} \texttt{PGA}(\tau)}{|\mathcal{F}|}$ and $\widehat{\mathcal{C}} = \frac{\sum_{\tau \in \mathcal{C}} \texttt{PGA}(\tau)}{|\mathcal{C}|}$. We know that the $\mathcal{F}$ is updated in the `Update` function when $\widehat{\mathcal{C}} > \widehat{\mathcal{F}}$. Let $\widehat{\mathcal{F}}'$ be the mean of the

intermediate set $\mathcal{F}' = \mathcal{F} \cup \mathcal{C}$. Then,

$$\widehat{\mathcal{F}}' = \frac{\sum_{\tau \in \mathcal{F}'} \texttt{PGA}(\tau)}{|\mathcal{F}'|} = \frac{\sum_{\tau \in \mathcal{F}} \texttt{PGA}(\tau) + \sum_{\tau \in \mathcal{C}} \texttt{PGA}(\tau)}{|\mathcal{F}| + |\mathcal{C}|}$$
$$= \frac{|\mathcal{F}|\,\widehat{\mathcal{F}} + |\mathcal{C}|\,\widehat{\mathcal{C}}}{|\mathcal{F}| + |\mathcal{C}|} = \widehat{\mathcal{F}} + \frac{|\mathcal{C}|\,k}{|\mathcal{F}| + |\mathcal{C}|} \tag{2}$$

since $\widehat{\mathcal{C}} > \widehat{\mathcal{F}}$, we can write this as $\widehat{\mathcal{C}} = \widehat{\mathcal{F}} + k$, where $k > 0$.

Now, let $\widehat{\mathcal{F}}''$ be the new mean after filtering $l < (|\mathcal{F}| + |\mathcal{C}|)$ rollouts whose $\texttt{PGA} \leq \widehat{\mathcal{F}}$ in the merged set $\mathcal{F}'$.

$$\widehat{\mathcal{F}}'' = \frac{|\mathcal{F}'|\,\widehat{\mathcal{F}}' - \Sigma\{\texttt{PGA}(\tau) | \tau \in \mathcal{F}', \texttt{PGA}(\tau) \leq \widehat{\mathcal{F}}\}}{|\mathcal{F}'| - l}$$

In the worst case, all l trajectories have PGAs at most $\widehat{\mathcal{F}}$.

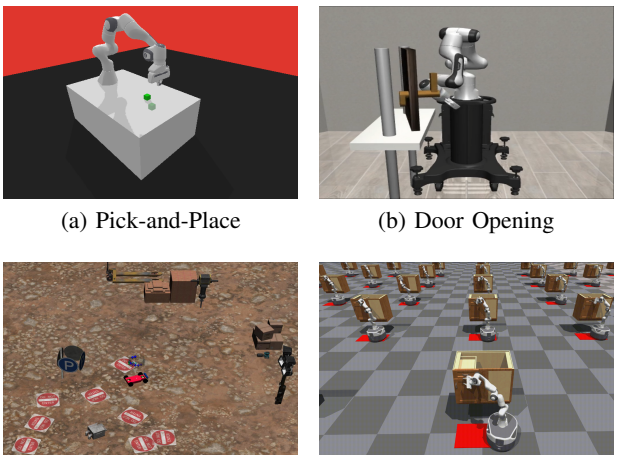
$$\widehat{\mathcal{F}}'' \geq \frac{|\mathcal{F}'|\,\widehat{\mathcal{F}}' - l\widehat{\mathcal{F}}}{|\mathcal{F}'| - l} = \frac{(|\mathcal{F}| + |\mathcal{C}|)\widehat{\mathcal{F}}' - l\widehat{\mathcal{F}}}{|\mathcal{F}| + |\mathcal{C}| - l}$$
$$\widehat{\mathcal{F}}'' \geq \widehat{\mathcal{F}} + \frac{|\mathcal{C}|\,k}{|\mathcal{F}| + |\mathcal{C}| - l} \quad \text{(substituting from (2))} \tag{3}$$

As the cardinalities of both buffers $\mathcal{F}$ and $\mathcal{C}$ are non-zero, the denominator $(|\mathcal{F}| + |\mathcal{C}| - l) > 0$. Thus, in (3), the second term is always positive, which proves that our algorithm improves the policy and reward in each cycle, under the exploration assumption. A special case of (3) is when $\mathcal{F}$ is completely replaced by $\mathcal{C}$, i.e., when all l trajectories belong to $\mathcal{F}$, then $l = |\mathcal{F}|$ and so, $\mathcal{F}$ inherits the higher mean from $\mathcal{C}$. The frontier remains unchanged when either the demonstrations or the rollouts in $\mathcal{F}$ at the end of each training cycle are optimal. We can apply similar reasoning to the other operators for $\odot$. In the case of `max`, the frontier's maximum value will always inherit the maximum (i.e., the best rollouts) from the candidate. For `min`, only the least-performance trajectories are discarded and the second-to-least ones are updated to be the new minimum in $\mathcal{F}$. Since the upper-bound of $\mathcal{F}$ is $n\Delta$, our method keeps improving the policy towards this maximum. *However, this does not guarantee that the maximum value can always be achieved due to several factors: conflicting specifications causing trade-offs, environment configuration, solvability of the MDP under the given specifications, etc.*

3) Effect of Affine Transformations to Rewards: In practice RL is sensitive to hyperparameter settings, environment stochasticity, scales of rewards and observations, and other algorithmic variances [30]. Hence, in our experiments, we normalize observations and rewards using affine transforms. However, applying affine transformations to the reward function does not alter the optimal policy [31]. We also prove this for basic scaling and shifting of the rewards by a constant amount in the supplemental document [32].

V. Experiments

Our proposed framework is evaluated on a diverse set of robotic simulation tasks (Fig. 3): (i) placing an object at a desired location, (ii) opening doors, (iii) safety-aware mobile navigation and (iv) closing cabinets with a mobile manipulator. In all experiments, the task specifications only monitor the observed states and so, the rewards are a function of just the states. The STL specifications are evaluated using `RTAMT` [33]. The reward function is modeled by regression with either fully connected neural networks or Gaussian processes. All experiments are performed on an Ubuntu desktop with an *Intel®Xeon 8-core CPU* and *Nvidia Quadro RTX 5000 GPU*. For each environment, $m = 5$ demonstrations are generated by training an appropriate RL agent under an expert dense reward function. *In these domains, every RL episode features a unique/randomized target and hence the collected demonstrations are unique (i.e., the states do not overlap.) Additionally, these simulations implicitly model noise in the environment which make it challenging to provide optimal trajectories.* Due to space restrictions, we provide details of all hyperparameters in the supplemental document [32]. In all tasks, unless explicitly stated, the frontier is updated by completely replacing its contents with the candidate (i.e., special case of (3)) and we set $|\mathcal{F}| = |\mathcal{C}| = 5$. Furthermore, in all tasks, the trained policy is evaluated on 5 random seeds, drawn from the baselines for comparisons. For each seed, 20 trials are performed, thus totalling 100 test scenarios; the mean success rates are then reported.



(a) Pick-and-Place (b) Door Opening

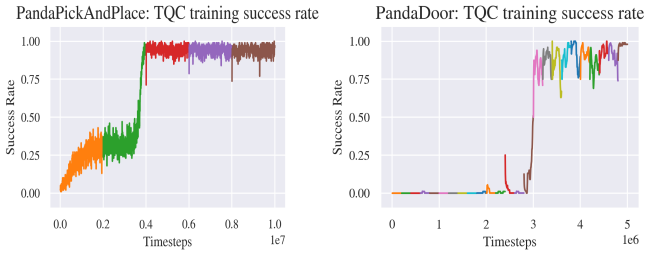
(c) Reach-Avoid (d) Cabinet Closing

Fig. 3. Overview of the robot simulation environments. The task in (d) uses the Nvidia Isaac simulator.

a) Task - Placing Cube: A Franka Panda robot is required to pick up a cube on a table (Fig. 3a) and place it at the desired location [34]. Only 4 of the 5 demonstrations were successful. The specifications are: $\varphi_g := \mathbf{F}(d < \delta)$ and $\varphi_t := \mathbf{G}(t < T)$, where d is the distance between the cube and target poses, δ is a small threshold to determine success, and T is the task-specific time in which the target must be achieved. The specifications indicate that the distance between the cube and desired pose is below a threshold and the robot must do so as quickly as possible. The RL agent used TQC [35] with HER [36] and achieved a training success rate of 98% (Fig. 4a), and converges to a high success rate after just 3 cycles. The resulting policy achieved

a success rate of $\mathbf{96}\%$ in the test trials. The task specification, although minimalistic, is significantly challenging because it only describes that the cube be placed at the desired pose. In other words, the RL agent must learn the sequence of elementary behaviors: reach, grasp and move to the desired location while holding the cube, just from the 5 demonstrations. Another remarkable finding in our work (shown in the supplemental video), is that the policy learns to (i) correctly *pick* the cube and place it at the target whenever the target height is above the table and (ii) *push/drag* the cube when the target is on the same table surface. This shows that our algorithm combines RL exploration and graph advantage to possibly learn *specification-satisfying* behaviors that were not observed before. Under identical training conditions, with the exclusion of reward model-specific hyperparameters, the number of demonstrations used for this task in the baselines that achieved comparable success rates, are: 100 for MCAC [37], between 4 and 16 for OPRIL [38], 20 for goalGAIL [39] and 50 for ROT [40].

b) Task - Opening Door: A Panda robot, mounted on a pedestal (Fig. 3b), is required to open a door [41]. Only 3 of the 5 demonstrations were successful. The task is successful if the door hinge is rotated beyond $\theta = 0.3$rad. The task specifications consist of (i) reaching the door handle, (ii) rotating the hinge beyond θ and (iii) completing the task within T steps. The elementary behaviors to be learned are: reaching the door handle, turning the handle to unlock the door and pulling to open the door. This is a non-trivial task for expert reward design as it must capture all these elementary behaviors and compose them sequentially. Since this is a more challenging task, the frontier was updated with *strategic merge*, and the size of reward buffers were set to 20 to collect more rollouts. The RL agent used TQC and was trained for 25 cycles to achieve a success rate of $\mathbf{98}\%$ (Fig. 4b). In the evaluations, the resulting policy achieved a success rate of $\mathbf{100}\%$.

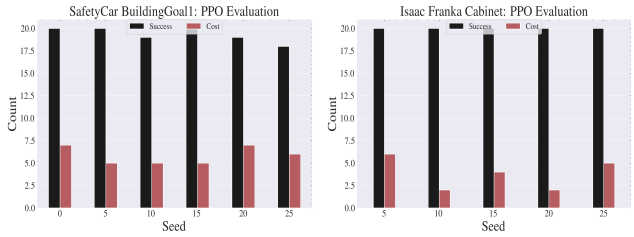


(a) Cube-Placing (b) Door Opening

Fig. 4. Summary of RL training for the object manipulation tasks.

We compare our work with two state-of-the-art baselines MCAC [37] and OPIRL [38], which has shown to outperform maximum entropy and adversarial IRL-based methods. Under identical training conditions, while both these methods successfully complete this task, MCAC used 100 demonstrations, while OPIRL used between 4 and 16. OPIRL had significantly more variance (i.e., unstable learning) with 4 demonstrations compared to using 16. Furthermore, in OPRIL, the method uses a substantially large reward buffer size of $2 \cdot 10^6$ to compensate for the limited demonstrations, while ours uses $2 \cdot 10^4$ (i.e., $|\mathcal{F}| = |\mathcal{C}| = 20$, each trajectory of length 500), using **100x** less memory. This indeed shows our method is more efficient compared to IL and IRL.

c) Task - Safe Mobile Navigation: In this task (Fig. 3c) [42], a mobile robot navigates to the goal while avoiding hazards (red markers) as much as possible. A cost is incurred for traversing a hazard, and the objective is to minimize this cost. The distance to the goal and hazards are provided by Lidar measurements and the observation space had 56 dimensions. The task specifications are: (i) $\varphi_g := \mathbf{F}(\bigvee_{i=1}^{16}(d_g^i < 0.1))$, where d_g^i is the Lidar's i-th distance measurement to the goal, (ii) $\varphi_s := \mathbf{G}(cost < 1)$, where $cost$ is the value incurred when the risk-area Lidar detects that the robot is too close to a hazard, and (iii) $\varphi_t := \mathbf{G}(t < T)$, where T is the maximum episode time. The RL agent was trained using PPO [43] for $5 \cdot 10^6$ steps over 25 cycles and the training time was about 20 hours. The evaluations (Fig. 5a) showed 98% task success rate with 28% mean cost. Compared to expert reward functions [42] and state-of-the-art IL method SIM [44], our method was able to achieve identical task success and cost rates, with **5x** fewer demonstrations and $\mathbf{50}\%$ fewer training steps. Furthermore, both specifications φ_g and φ_s have a length of 16, indicating that our method is able to effectively accommodate lengthy specifications.



(a) Safety-Car (b) Isaac Drawer

Fig. 5. Evaluation results for safety-aware tasks.

d) Task - Cabinet Closing with Mobile-Manipulator: In this task (Fig. 3d), a mobile-manipulator consisting of a Panda arm mounted on a Fetch Robotics Freight mobile robot platform, must close the cabinet drawer while minimizing traversing an unsafe (red) zone. The observation space consists of 57 dimensions, posing a challenge for reward models. This simulation is built on the RL adaptation of *Nvidia Isaac Sim* [45], which enables parallel (vectorized) training environments. The specifications are similar to the safe-navigation task, but with $\varphi_g := \mathbf{F}(drawer_y < 0.2)$, i.e., the drawer must be closed (y-axis) within a 0.2 unit tolerance. Only 4 of the 5 demonstrations succeeded the task. The RL agent was trained using PPO on 400 parallel instances for 10^7 steps over 10 cycles. Due to the highly vectorized implementation, the training was completed within 1.5 hours. It had a 100% success rate and 19% mean cost on the test trials (Fig. 5b), similar to the policies from expert-designed complex dense reward functions.

VI. CONCLUSIONS

We developed AL-STL, a novel LfD framework, that utilizes apprenticeship learning and STL task objectives to infer rewards and policies simultaneously. AL-STL is a significant advancement over prior LfD-STL by introducing closed-loop learning that iteratively improves the quality of rewards and policies. We proposed a graph-based optimization formalism, *performance graph advantage*, which (i) provides a succinct representation of multiple non-Markovian (temporal) task specifications for quantitative and interpretable assessments of agent behaviors, and (ii) guides the agent's learning process to maximally satisfy the task specifications and perform optimal trade-offs. Through realistic simulation experiments on mobile and manipulation robotic tasks, we have discussed how our approach outperforms several state-of-the-art methods in terms of sample and space efficiency. For future, we propose to investigate diversity in demonstrations, vision-based observations, prioritization of specifications in trade-offs and, task and sim2real transfer-learning.

REFERENCES

[1] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *ICRA*, 2018, pp. 6292–6299.

[2] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *IJCAI*, 2018, pp. 4950–4957.

[3] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000, pp. 663–670.

[4] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, vol. 69. ACM, 2004.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[6] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI*, 2008.

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

[8] L. E. Asri, B. Piot, M. Geist, R. Laroche, and O. Pietquin, "Score-based inverse reinforcement learning," in *AAMAS*. ACM, 2016, pp. 457–465.

[9] L. Chen, R. R. Paleja, and M. C. Gombolay, "Learning from suboptimal demonstration via self-supervised reward regression," in *CoRL*, 2020.

[10] D. S. Brown, W. Goo, and S. Niekum, "Better-than-demonstrator imitation learning via automatically-ranked demonstrations," in *CoRL*. PMLR, 2020.

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

[12] S. Pitis, D. Bailey, and J. Ba, "Rational multi-objective agents must admit non-markov reward representations," in *NeurIPS ML Safety Workshop*, 2022.

[13] D. Abe, A. Barreto, M. Bowling, W. Dabney, S. Hansen, A. Harutyunyan, M. K. Ho, R. Kumar, M. L. Littman, D. Precup, and S. Singh, "Expressing non-markov reward to a markov agent," in *RLDM*, 2022.

[14] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Ltl and beyond: Formal languages for reward function specification in reinforcement learning," in *IJCAI*, 2019.

[15] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *J. Artif. Int. Res.*, vol. 73, 2022.

[16] A. Puranic, J. Deshmukh, and S. Nikolaidis, "Learning from demonstrations using signal temporal logic," in *CoRL*, 2021.

[17] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Learning from demonstrations using signal temporal logic in stochastic and continuous domains," *RA-L*, 2021.

[18] ——, "Learning performance graphs from demonstrations via task-based evaluations," *RA-L*, 2023.

[19] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," in *ICLR*, 2018.

[20] F. Torabi, G. Warnell, and P. Stone, "Generative adversarial imitation from observation," 2019.

[21] W. Zhou and W. Li, "Safety-aware apprenticeship learning," in *CAV*. Springer, 2018.

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

[23] X. Li, Y. Ma, and C. Belta, "Automata guided reinforcement learning with demonstrations," *CoRR*, vol. abs/1809.06305, 2018.

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

[25] M. Wen, I. Papusha, and U. Topcu, "Learning from demonstrations with high-level side information," in *IJCAI*, 2017.

[26] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *FORMATS*. Springer, 2004.

[27] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *FORMATS*, 2010.

[28] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, 2009.

[29] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta, "Control from signal temporal logic specifications with smooth cumulative quantitative semantics," in *CDC*, 2019.

[30] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," in *Reproducibility in Machine Learning Workshop (ICML)*, 2017.

[31] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*. Morgan Kaufmann, 1999, pp. 278–287.

[32] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Signal temporal logic-guided apprenticeship learning - supplemental document," https://aniruddh-puranic.info/assets/pdf/alstl_supp.pdf, 2024.

[33] D. Nickovic and T. Yamaguchi, "RTAMT: online robustness monitors from STL," in *ATVA*, 2020.

[34] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, "panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning," *NeurIPS Workshop*, 2021.

[35] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," in *ICML*, 2020.

[36] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *NeurIPS*, vol. 30, 2017.

[37] A. Wilcox, A. Balakrishna, J. Dedieu, W. Benslimane, D. Brown, and K. Goldberg, "Monte carlo augmented actor-critic for sparse reward deep reinforcement learning from suboptimal demonstrations," *NeurIPS*, 2022.

[38] H. Hoshino, K. Ota, A. Kanezaki, and R. Yokota, "Opirl: Sample efficient off-policy inverse reinforcement learning via distribution matching," in *ICRA*, 2022.

[39] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, "Goal-conditioned imitation learning," in *NeurIPS*, 2019.

[40] S. Haldar, V. Mathur, D. Yarats, and L. Pinto, "Watch and match: Supercharging imitation with regularized optimal transport," *CoRL*, 2022.

[41] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[42] J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang, "Safety gymnasium: A unified safe reinforcement learning benchmark," in *NeurIPS Datasets and Benchmarks Track*, 2023.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[44] H. Hoang, T. Mai, and P. Varakantham, "Imitate the good and avoid the bad: An incremental approach to safe reinforcement learning," in *AAAI*, 2024.

[45] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.