Interpretable Imitation Learning via Generative Adversarial STL Inference and Control

Wenliang Liu* WLIU97@BU.EDU
Danyang Li * DANYANGL@BU.EDU
Department of Mechanical Engineering, Boston University, Boston, MA, USA

Erfan Aasi EAASI@MIT.EDU
Daniela Rus RUS@MIT.EDU
Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Roberto Tron TRON@BU.EDU
Department of Mechanical Engineering, Boston University, Boston, MA, USA

Calin Belta CALIN@UMD.EDU
Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA

Editors: G. Pappas, P. Ravikumar, S. A. Seshia

Abstract

Imitation learning methods have demonstrated considerable success in teaching autonomous systems complex tasks through expert demonstrations. However, a limitation of these methods is their lack of interpretability, particularly in understanding the specific task the learning agent aims to accomplish. In this paper, we propose a novel imitation learning method that combines Signal Temporal Logic (STL) inference and control synthesis, enabling the explicit representation of the task as an STL formula. This approach not only provides a clear understanding of the task but also supports the integration of human knowledge and allows for adaptation to out-of-distribution scenarios by manually adjusting the STL formulas and fine-tuning the policy. We employ a Generative Adversarial Network (GAN)-inspired approach to train both the inference and policy networks, effectively narrowing the gap between expert and learned policies. The efficiency of our algorithm is demonstrated through simulations, showcasing its practical applicability and adaptability.

Keywords: Temporal logic, Control synthesis, Imitation learning, Generative adversarial network

1. INTRODUCTION

Imitation learning is a machine learning technique that enables autonomous systems to learn tasks by mimicking expert behavior and is increasingly popular for teaching complex tasks to robotic systems efficiently. This paper focuses on imitation learning using offline data, i.e., no interaction with the expert is required during the learning process. In this setting, imitation learning can be categorized into two main types: behavioral cloning Pomerleau (1991), which utilizes supervised learning to learn a policy from state-action pairs, and Inverse Reinforcement Learning (IRL) Ng et al. (2000), which infers the reward function that the expert optimizes and applies Reinforcement Learning (RL) to find a control policy. Although behavioral cloning is conceptually simple, it mimics the expert without uncovering the underlying task. Hence, it lacks adaptability to situations not covered in the training data. IRL can recover a reward function, but it is computationally expensive, prone to

* These authors contributed equally.

overfitting, and difficult to interpret. Therefore, it is challenging to assess the correctness of the reward function and manipulate it to incorporate human knowledge or adapt to new scenarios.

Temporal logics, including Linear Temporal Logic (LTL) Baier and Katoen (2008) and Signal Temporal Logic (STL) Maler and Nickovic (2004), are widely used for control system specifications due to their expressiveness. STL inference is the process of deriving formal descriptions of system behaviors from observed data in the form of STL formulas Baharisangari et al. (2021). The STL inference approach classifies system behaviors as desired or undesired based on whether they satisfy the inferred formula Bombara and Belta (2021); Aasi et al. (2022). In this paper, we propose a method that combines STL inference with control synthesis to enhance the interpretability of imitation learning. Our approach infers an STL formula that describes the task an expert aims to accomplish, and learns a control policy to satisfy this formula in a dynamic environment, where agent dynamics can be either known or learned, and environment dynamics are unknown. Since STL formulas closely resemble natural language, our approach provides some understanding of the objective of the policy. Moreover, they can be manually adjusted to align with expert knowledge or adapt to new conditions. The policy can also be fine-tuned in new scenarios using the learned or modified STL formulas, enabling out-of-distribution generalization without new expert data.

One of the main challenges in combining the STL inference and control tasks is the need to have both positive (expert demonstrations) and negative (incorrect behaviors) examples in order to infer an STL formula classifier. The goal is to push the decision boundary (in the form of an STL formula) as close to the positive data as possible to align the synthesized control policy with the expert's policy. However, negative examples are often unavailable or insufficient. Although we can record negative data or manually generate them using a simulator, they can hardly be comprehensive enough. To address this concern, we incorporate Generative Adversarial Networks (GANs) Goodfellow et al. (2014) into our framework. We consider the policy network as the generator and the inference network as the discriminator. The policy generates fake (negative) data, and the STL formula distinguishes them from positive data. We show that by iteratively training the two networks, the learned policy gradually approaches the expert's policy.

The contributions of this paper are as follows: (1) We develop an interpretable imitation learning approach for dynamic environments by integrating STL inference with control synthesis, where the tasks that the expert aims to accomplish are explicitly learned as STL formulas. (2) We incorporate GANs into our framework to gradually bridge the gap between expert and learned policies. (3) We illustrate the efficacy of our algorithm through three case studies. We show that the inferred STL formula can be adjusted to incorporate rules from human knowledge, and the policy can be retrained and adapted to unseen scenarios.

2. Related Work

STL inference and control synthesis, as separate areas, have received significant attention recently. Early efforts in STL inference focused on mining optimal parameters for predefined formula templates Asarin et al. (2012); Jin et al. (2013); Jha et al. (2017); Hoxha et al. (2018). Recent studies have proposed general learning frameworks to infer both formula structures and their parameters, using techniques such as lattice search Kong et al. (2016), decision trees Bombara and Belta (2021); Aasi et al. (2022), enumeration-based methods Mohammadinejad et al. (2020), and neural networks Chen et al. (2022); Li et al. (2023). Other method involves using "landmarks" to build a policy summary Sreedharan et al. (2020). On the other hand, control synthesis from STL formulas

can be solved using mixed integer programming (MIP) Raman et al. (2014); Sadraddini and Belta (2015), or gradient-based optimization Pant et al. (2017); Haghighi et al. (2019); Gilpin et al. (2020). Recently, learning-based control methods under STL specifications have been proposed, including Q-learning Aksaray et al. (2016) and model-based methods Yaghoubi and Fainekos (2019); Liu et al. (2021); Leung and Pavone (2022); Liu et al. (2023). The latter considered static environments, while in this paper we consider a learning agent in a dynamic environment.

Integrating STL inference with control synthesis has gained limited attention in existing literature. One related study Xu et al. (2018) learns advice in the form of STL formulas from successful and failed trajectories, and design an advisory controller to satisfy the inferred STL formulas. However, this approach is restricted to a specific template of STL formulas and requires expert intervention during the learning process. In their method, decision trees are used for inference, and the controller is synthesized using a MIP solver. In contrast, our approach enables the template-free learning of STL formulas, and both inference and control synthesis are based on neural networks, leading to a simpler training process and more efficient online execution using offline data only.

The combination of GANs and imitation learning was first proposed as Generative Adversarial Imitation Learning (GAIL) in Ho and Ermon (2016) and has been studied extensively in the literature, e.g., Baram et al. (2017); Wang et al. (2017). Similar to the existing works in imitation learning, these approaches lack interpretability. To the best of our knowledge, this paper is the first to use GANs to integrate temporal logic inference and control synthesis.

3. Problem Statement and Approach

We consider a discrete-time system consisting of an agent and its environment. The state of the system at time $t \in [0, T] \cap \mathbb{Z}$ is denoted as $x(t) = \big(x_{ag}(t), x_{env}(t)\big)$, where $x_{ag}(t) \in \mathbb{R}^{n_a}$ is the state of the agent, $x_{env}(t) \in \mathbb{R}^{n_e}$ is the state of the environment, and $T \in \mathbb{Z}^{>0}$ is the time horizon that we are interested in. Let the dynamics of the agent (independent from the environment) be:

$$x_{ag}(t+1) = f\big(x_{ag}(t), u(t)\big), \tag{1}$$

where $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ is the control input at time t, $\mathcal{U}$ is a set capturing the control constraints. We assume $\mathcal{U}$ is a box constraint and f is a differentiable function. Let $P_0 : \mathcal{X}_0 \to \mathbb{R}^{\geq 0}$ be a known probability distribution of the initial state for the agent over a set $\mathcal{X}_0$ and $P : (\mathbb{R}^{n_e})^{T+1} \to \mathbb{R}^{\geq 0}$ be an unknown distribution of the environment trajectory. Let $\mathbf{x}^{t_1:t_2} = [x(t_1), \ldots, x(t_2)]$ denote a sequence of system states from time t_1 to t_2 and $\mathbf{x} = \mathbf{x}^{0:T}$ denote the whole trajectory. We assume the state x can be fully observed by the agent at all times. The agent model (1) can be either known or separately learned using the method in Liu et al. (2023). We consider both cases in Sec 6.

Assume we have a dataset $D = \{(\mathbf{x}^i, l^i)\}_{i=1}^N$ of system trajectories, all of length T, where $l^i \in \{1, -1\}$ is the label with 1 indicating expert demonstration and -1 indicating undesired behaviors. We allow the dataset to contain only positive samples. Our goal is to generate an interpretable description of the expert's objective and a control policy that resembles the expert's strategy. This description is captured by an STL formula ϕ interpreted over $\mathbf{y} = [y(0), \ldots, y(T)]$, where $y(t) = h\big(x(t)\big)$ is the feature extracted from the system state and $h : \mathbb{R}^{n_a+n_e} \to \mathbb{R}^{n_y}$ is a known differentiable function. In this paper, we consider a fragment of STL with the syntax:

$$\phi ::= \top \mid \mu \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Diamond_{[t_1,t_2]}\phi \mid \Box_{[t_1,t_2]}\phi, \tag{2}$$

where μ is a predicate $\mu := a^\top y(t) \geq b$, $a \in \mathbb{R}^{n_y}$, $b \in \mathbb{R}$, and ϕ, ϕ_1, ϕ_2 are STL formulas. The Boolean operators $\neg, \wedge, \vee$ are *negation*, *conjunction* and *disjunction*, and the temporal operators $\Diamond$

and $\square$ represent *eventually* and *always*, respectively. $\Diamond_{[t_1,t_2]}\phi$ is true if ϕ is satisfied for at least one time point $t \in [t_1, t_2] \cap \mathbb{Z}$, while $\square_{[t_1,t_2]}\phi$ is true if ϕ is satisfied for all time points $t \in [t_1, t_2] \cap \mathbb{Z}$.

The quantitative semantics Donzé and Maler (2010), also known as robustness, of an STL formula ϕ, denoted as $r(\mathbf{y}, \phi)$, is a scalar that measures how strongly the formula is satisfied by a signal $\mathbf{y}$. The robustness is sound, which means that $r(\mathbf{y}, \phi) \geq 0$ if and only if ϕ is satisfied by $\mathbf{y}$.

Consider a history-dependent policy $u(t) = \pi(\mathbf{x}^{0:t})$. We assume the agent's policy depends on the environment but does not influence it. Our goal can be formulated as:

Problem 1 *Find an STL formula ϕ that classifies the positive and negative data in a set D, and find a policy $u(t) = \pi(\mathbf{x}^{0:t})$ that maximizes $r(\mathbf{y}, \phi)$, i.e., the robustness of ϕ, where $y(t) = h\big(x(t)\big)$.*

We parameterize both the STL robustness function and the policy using neural networks (detailed in Sec. 4), referred to as an inference network $r(\mathbf{y}, \phi_{\theta_I}) = \mathcal{I}(\mathbf{y}; \theta_I)$ and a policy network $u(t) = \pi(\mathbf{x}^{0:t}; \theta_P)$, where θ_I and θ_P are the neural network parameters, and ϕ_{θ_I} is an STL formula which can be extracted from the inference network parameters θ_I. Then we divide Prob. 1 into two subproblems, the inference problem and the control problem, and formulate them as:

Problem 2 *[Inference] Given a dataset D, find the optimal parameters θ_I^* for the inference network $\mathcal{I}(\mathbf{y}; \theta_I)$ such that the inferred STL formula ϕ_{θ_I} minimizes the misclassification rate (MCR): $\theta_I^* = \arg\min_{\theta_I} MCR(\phi_{\theta_I}, D)$.*

Problem 3 *[Control] Given system (1) and the inference network $\mathcal{I}(\mathbf{y}; \theta_I)$, find the optimal parameters θ_P^* for the policy network $\pi(\mathbf{x}^{0:t}; \theta_P)$ that maximize the expected robustness of ϕ_{θ_I}:*

$$\begin{aligned} \theta_P^* = \arg\max_{\theta_P} \quad & \mathbb{E}_{x_{ag}(0)\sim P_0, \mathbf{x}_{env}\sim P}\, \mathcal{I}(\mathbf{y}; \theta_I) \\ \textit{s.t.} \quad & x_{ag}(t+1) = f\big(x_{ag}(t), \pi(\mathbf{x}^{0:t}; \theta_P)\big),\ y(t) = h(x(t)). \end{aligned} \tag{3}$$

Solving Prob. 2 and Prob. 3 in one shot may not lead to an accurate STL formula describing the expert's behavior and a policy close enough to the expert's policy. The STL formula ϕ_{θ_I} can be considered as the decision boundary of a binary classifier. In order to accurately explain the underlying task, this decision boundary is expected to be as close as possible to the positive data. However, the recorded or manually designed negative data can hardly be comprehensive enough to push the decision boundary toward the positive data, because there exists many patterns of undesired behaviors of the system. For instance, it is possible to generate negative data for specific incorrect behaviors, such as a vehicle running a red light, while there are many other undesired behaviors—such as stopping in the middle of the road or frequent acceleration and deceleration. To address this challenge, we propose a generative adversarial training approach. We frame Prob. 1 as a game between an inference network and a policy network. At each iteration, the policy network generates new trajectories, which are then added to the dataset as negative samples, and the inference network is retrained with this updated data. The policy network aims to generate expert-like trajectories that are difficult for the inference network to classify, while the inference network learns to differentiate between expert and non-expert behaviors. This setup mirrors the roles of the generator and discriminator in a Generative Adversarial Network (GAN), as illustrated in Fig. 1.

4. Neural Network Architectures

4.1. STL Inference Based on Neural Networks

In this paper, we apply the TLINet Li et al. (2023, 2024) as the inference network, which is a template-free model that provides flexibility in learning various structures of STL formulas. A

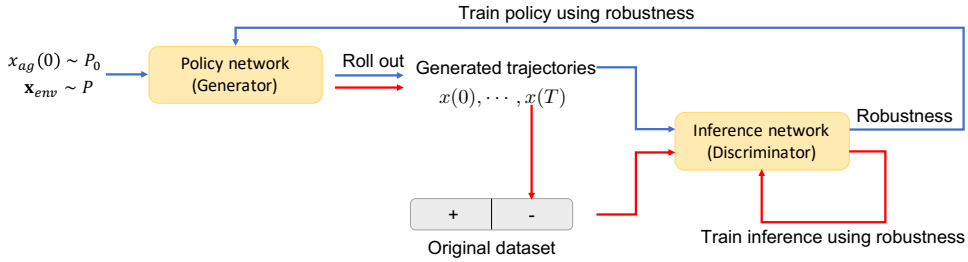


Figure 1: An overview of the Generative Adversarial STL Inference and Control: red arrows represent inference network training, while blue arrows indicate policy network training.

Figure 2: An example of a 5-layer inference network.

TLINet consists of multiple layers, each containing modules corresponding to specific types of operators. These modules are categorized into three types: predicate, temporal and Boolean. The predicate module learns the predicate type and spatial parameters; the temporal module learns the type of temporal operator and temporal parameters; and the Boolean module learns the type of Boolean operator and the structure of the formula. This capability provides flexibility in generating STL formulas, as it is not constrained by a fixed network structure. TLINet allows for flexible combinations of layers, enabling it to describe diverse behaviors. Fig. 2 shows an example of the TLINet structure.

A TLINet can be transferred to an STL formula ϕ_{θ_I} by decoding its parameters θ_I and translating modules to compositions of ϕ_{θ_I}. A detailed decoding and translation example is provided in Appendix A. With a given signal $\mathbf{y}$ as input, the output of a TLINet is an estimated robustness value $\tilde{r}(\mathbf{y}, \phi_{\theta_I}) = \mathcal{I}(\mathbf{y}; \theta_I)$, where $\tilde{r}(\mathbf{y}, \phi_{\theta_I})$ is a smooth approximation of the true robustness. We use Sparse Softmax Li et al. (2024) and Averaged Max Li et al. (2024) to approximate the robustness, which allows to incorporate gradient-based methods and can guarantee soundness. Thus, $\tilde{r}(\mathbf{y}, \phi_{\theta_I})$ is able to represent a rigorous classifier, i.e., $\tilde{r}(\mathbf{y}, \phi_{\theta_I})$ is positive if and only if the input signal belongs to the positive class and satisfies the STL formula ϕ_{θ_I}.

4.2. Recurrent Neural Network Control Policy

In general, to satisfy an STL specification, the control policy needs not only the current state but also past states Liu et al. (2021). For example, if a robot is required to move back and forth between two regions, then in the middle of the two regions the robot will need the history information of which region it has just visited to decide where to go. Hence, we apply a Recurrent Neural Network (RNN) policy $\pi(\mathbf{x}^{0:t}; \theta_P)$, where θ_P is the RNN parameters. Similar to Yaghoubi and Fainekos (2019), we apply a hyperbolic tangent function on the output of the RNN to satisfy the control constraint $\mathcal{U}$.

5. Training of Inference and Policy Networks

Now we elaborate the training processes of the networks, i.e., the solutions to Prob. 2 and Prob. 3. Then we detail the joint training of the two networks using the generative adversarial approach.

5.1. Training of Inference Network

Following Li et al. (2024), we solve Prob. 2 by minimizing the following loss function:

$$\theta_I^*, \epsilon^* = \arg\min_{\theta_I, \epsilon} \frac{1}{N} \sum_{i=1}^{N} ReLU\big(\epsilon - l^i \cdot \mathcal{I}(\mathbf{y}^i, \theta_I)\big) - \beta_1 \epsilon + \beta_2 Reg(\theta_I), \tag{4}$$

where $\epsilon > 0$ is the margin, Reg is a regularizer to adjust formula complexity, which is a combination of a bi-modal regularizer Murray and Ng (2010) as well the traditional $\ell 2$ regularizer. $\beta_1 > 0, \beta_2 \in \mathbb{R}$ are two hyperparameters to control the compromise among formula complexity, maximizing the margin ϵ, and minimizing the MCR. The loss function (4) satisfies that it is small when the inferred formula is satisfied by the positive data and violated by the negative data, and it is large otherwise. The margin ϵ is a quantitative measurement of the separation of signals by ϕ_{θ_I}. The loss function encourages a large margin to better distinguish the positive and negative data.

5.2. Training of Policy Network

Model-based training of an RNN control policy in a static environment was introduced in Liu et al. (2023). In this paper, we extend this method to dynamic environments. To solve Prob. 3, we need to estimate the expectation in (3). We first sample a set of M initial agent states $\{\bar{x}_{ag}^j(0)\}_{j=1}^M$ from the known distribution P_0. Next, since the distribution P is unknown, we sample M environment trajectories $\{\bar{\mathbf{x}}_{env}^j\}_{j=1}^M$ from the dataset D. We use the agent dynamics (1), policy network π, and the sampled environment trajectories to roll out the system trajectories from the sampled initial agent states. Then we use the mean value to estimate the expectation, attaining:

$$\begin{aligned} \theta_P^* = \arg\max_{\theta_P} \quad & \frac{1}{M} \sum_{j=1}^{M} \mathcal{I}\big(\bar{\mathbf{y}}^i; \theta_I\big) \\ \text{s.t.} \quad & \bar{x}_{ag}^j(t+1) = f\big(\bar{x}_{ag}^j(t), \pi(\bar{\mathbf{x}}^{0:t,j}; \theta_P)\big),\ \bar{y}^j(t) = h\big(\bar{x}^j(t)\big), \end{aligned} \tag{5}$$

where the closed-loop dynamics can be substituted into the objective to form an unconstrained optimization problem. Since both inference $\mathcal{I}$ and policy π are based on neural networks, and the dynamics (1) are differentiable, the gradient of the objective can be backpropagated to the policy parameters. At each optimization step, we resample the initial states and environment trajectories to get an unbiased estimation of the objective and optimize it using Adam Kingma and Ba (2014).

5.3. Generative Adversarial Training

In this section, we describe the generative adversarial training approach used to iteratively refine both the inference and policy networks to improve the overall solution. We begin by training the inference network, followed by training the policy network. Next, we sample a set of initial states and environment trajectories, apply the learned policy, and roll out system trajectories, which are assigned negative labels and added to the dataset. This process (called an iteration) is repeated until convergence. If the original dataset contains only positive data, we first generate negative-labeled trajectories using a random policy and follow the same procedure. To improve the stability of training, we balance the positive and negative samples by randomly selecting an equal number of negative samples as expert demonstrations from the pool of collected negative data at each iteration.

During training, as the policy network improves, trajectories resembling the expert demonstrations are added to the negative dataset. This can cause an increase in the MCR of the inferred STL formula, which may, in turn, degrade the performance of the policy network. To mitigate this issue, we introduce a performance score based on Dynamic Time Warping (DTW) Sakoe and Chiba (1978) to assess the control policy. Specifically, we apply the learned policy π to the same initial conditions and environment trajectories as those used in the expert demonstrations and compute the similarities between the generated trajectories $\mathbf{x}^{i,\pi}$ and the expert demonstrations $\mathbf{x}^i$, which gives us a performance score $r(\pi)$ for the policy π:

$$\begin{aligned} r(\pi) &= -\sum_{\{i|l_i=1\}} DTW(\mathbf{x}^{i,\pi}, \mathbf{x}^i), \text{ where} \\ \mathbf{x}^{i,\pi}(0) &= \mathbf{x}^i(0),\ x^i_{env}(t) = x^{i,\pi}_{env}(t),\ x^{i,\pi}_{ag}(t+1) = f\big(x^{i,\pi}_{ag}(t), \pi(\mathbf{x}^{i,\pi}_{0:t}; \theta_P)\big). \end{aligned} \tag{6}$$

We select the policy with the highest performance score as the final policy, with the corresponding STL formula as its interpretation.

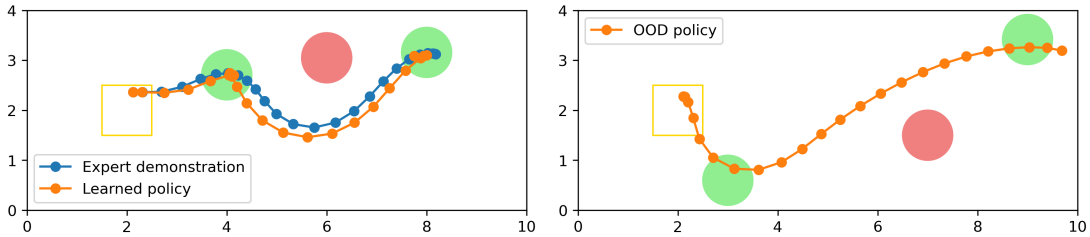
6. Experiments

In this section, we evaluate our algorithm through three case studies. The first one involves a navigation task with random goal and obstacles. We compare our algorithm with Behavioral Cloning (BC) and show out-of-distribution generalization of our approach. In the second, we apply our approach to tasks in the MuJoCo environments with unknown dynamics, and compare its performance with GAIL. The third one includes a self-driving vehicle in a dynamic environment. We show how user-generated formulas can be integrated into the policy to incorporate human knowledge and how the policy can be efficiently adapted to previously unseen scenarios through formula modification. Some detailed setup and results of experiments are omitted here and given in Appendix B. Videos of the experiments can be found at https://youtu.be/peQSizAwNAo. The implementation is available at: https://github.com/danyangl6/IGAIL.

6.1. Random Navigation

Consider a unicycle robot moving in a 2D environment with known discrete-time dynamics as shown in Fig 3. The state of the robot $x_{ag} = (p_x, p_y, \theta)$ contains its 2D position p_x, p_y and orientation θ, and the control $u = (v, \omega)$ includes the forward velocity v and angular velocity ω. The initial state is uniformly sampled from $[1.5, 2.5]^2 \times [-0.01, 0.01]$. The environment contains two goal regions and one obstacle region, each with a radius of 0.5. They are centered at random y-coordinates uniformly sampled from $[0.5, 3.5]$ (with fixed x-coordinates). The robot must first visit the closer goal region, then the other one, and always avoid the obstacle, with $p_y \in [0, 4]$. This behavior is specified by an STL formula (unknown to the learner), which is provided in Appendix B. The environment state includes the static locations of the goal and obstacle regions, i.e., $x_{env}(t) = x_{env}(0)$, $\forall t \in [0, T]$. We collect a dataset of 1000 positive robot trajectories with a time horizon $T = 24$ using an expert policy trained through model-based policy search Liu et al. (2023).

We evaluate our algorithm from 6 random initializations of both networks. The learned STL formula (given in Appendix B) closely reproduces the specification the expert aims to satisfy. Fig. 3(*a*) shows a sampled trajectory generated using the learned policy starting from the same initial state as the expert demonstration. Table. 1 lists the robustness of the learned policy with respect to the



(*a*) (*b*)

Figure 3: (a) Expert demonstration and sampled trajectory using the learned policy. (b) Sampled trajectory using the learned policy after fine-tuning in an OOD environment.

	Ours	BC	Expert	Ours (OOD)	BC (OOD)
Robustness	0.19 ± 0.13	0.31 ± 0.08	0.33 ± 0.06	0.12 ± 0.16	-0.12 ± 0.13
Suc. rate	$85.4 \pm 14.5\%$	97.5%	99.2%	$72.7\% \pm 22.7\%$	17.1%

Table 1: Robustness and success rate (mean$\pm$std) of the learned policy w.r.t. the true STL formula

ground-truth STL formula and the success rate of satisfying it, evaluated over 1000 random initial states for each network initialization. We compare the learned policy with the expert policy and BC policy trained using the same set of expert demonstrations. Our policy achieves relatively high robustness and success rate. Although its performance does not match that of BC, it provides a reasonable description of the task using the STL formula. This interpretability is absent in BC.

To demonstrate the out-of-distribution (OOD) generalization of our approach, we modify the environment by changing the x coordinates of the 3 regions (y coordinates are still random) as shown in Fig. 3(*b*). Without any new expert data, we fine-tune the policy network in this new environment with the same inferred STL formula. We compare the performance with the BC policy, which cannot be retrained due to the lack of available data. The statistics in Table. 1 show that after fine-tuning, our policy maintains relatively high robustness and success rate, while the performance of the BC policy dropped significantly.

6.2. MuJoCo Tasks

We evaluate our algorithm against baselines on 2 control tasks, the Inverted Pendulum with $x \in \mathbb{R}^4$ and the Reacher with $x \in \mathbb{R}^{11}$, as shown in Fig. 4 (detailed in Appendix B). All environments are simulated with MuJoCo Todorov et al. (2012) and the system dynamics are unknown and learned together with the policy as in Liu et al. (2023). Each task comes with a true reward function, defined in the OpenAI Gym Brockman (2016). As modern RL approaches, such as PPO Schulman et al. (2017), can generate high-quality policies for these tasks, we first run PPO on each task to create 1000 expert trajectories, each trajectory is truncated or padded to 50 steps. We test our algorithm on this dataset and compare it with the standard GAIL Ho and Ermon (2016). Learned STL formulas (given in Appendix B) effectively describe the tasks in an interpretable manner. The rewards gained by our approach, GAIL, and the expert policy are given in Table 2. The statistics of our algorithm are computed over 5 network initializations, with 1000 trajectories generated for each initialization.

For the Inverted Pendulum task, both our approach and GAIL reach the maximum reward achievable in 50 steps. For the Reacher task, our approach slightly outperforms GAIL. However,

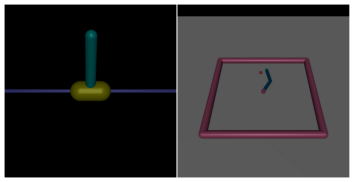
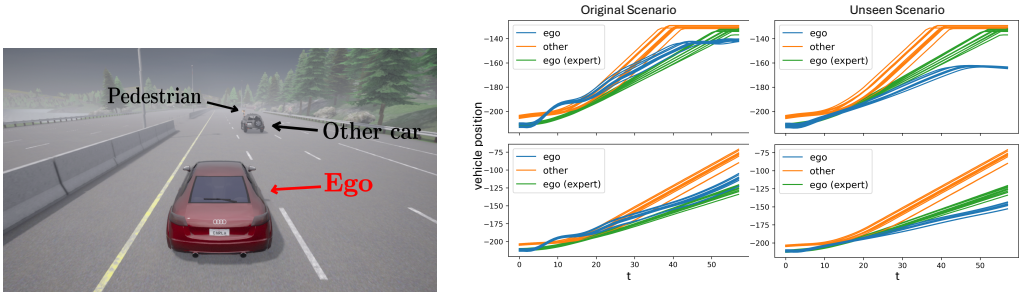


Figure 4: Inverted Pendulum (left) and Reacher (right).

Task	Ours	GAIL	Expert
Pendulum	50.0 ± 0.0	50.0 ± 0.0	50.0 ± 0.0
Reacher	-4.76 ± 1.94	-5.04 ± 2.22	-4.31 ± 1.65

Table 2: Rewards (mean±std) obtained by different polices.



(*a*) (*b*)

Figure 5: (a) The autonomous driving scenario simulated in Carla. (b) Sampled trajectories of ego and other vehicles' positions after training in the original and unseen scenarios. The top figures show the situations when there are pedestrians, and the bottom figures show the situations when there is no pedestrian.

for MuJoCo tasks such as Swimmer and Walker, which are hard to express by the STL fragment in (2), our approach does not reach good reward levels. In future work, we plan to investigate ways to enhance the expressivity of the inference network to address this limitation.

6.3. A Self-Driving Scenario

Inspired by Aasi et al. (2022), we consider the following scenario: a self-driving vehicle (ego) drives in an urban environment alongside another vehicle driven by a "reasonable" human in the adjacent lane. The initial position of the other vehicle is always ahead of the ego, and both are headed toward an unmarked crosswalk. If a pedestrian crosses, the other vehicle brakes to stop; otherwise, it continues moving without deceleration. Due to foggy weather and the other vehicle obstructing the view, ego needs to infer the presence of a pedestrian by observing the behavior of other vehicle (see Fig. 5(*a*)). Under the assumption that ego only moves forward, we simplify its dynamics as a double integrator (assumed to be known). The system state $x(t) \in \mathbb{R}^4$ is the concatenation of the agent state $x_{ag}(t) = \big(p_{eg}(t), v_{eg}(t)\big)$ and the environment state $x_{env}(t) = \big(p_{ot}(t), v_{ot}(t)\big)$, where $p_{eg}(t)$ and $v_{eg}(t)$ are the position and velocity of ego; $p_{ot}(t)$ and $v_{ot}(t)$ are the position and velocity of the other vehicle. The control $u(t)$ is the same as ego's acceleration $a_{eg}(t) \in \mathcal{U} \subset \mathbb{R}$. We use the autonomous driving simulator Carla Dosovitskiy et al. (2017) to generate a dataset of 800 trajectories, containing both positive and negative labels, and they both include situations with and without pedestrians (200 data for each situation). The time horizon $T = 57$.

Incorporate known critical rules: Some critical rules from human knowledge can be manually incorporated as supplementary formulas into the inferred STL formula to make the policy satisfy them. Here, we formulate the speed limit rule as $\phi_{sl} := \square_{[0,57]}\big((v_{eg} \leq 10) \wedge (v_{eg} > -1)\big)$. We encode ϕ_{sl} into the inference network as a conjunction to the inferred formula: $\tilde{\phi}_{\theta_I} = \phi_{\theta_I} \wedge \phi_{sl}$. We

test the learned policy in Carla and track the output of the policy. Sampled trajectories are shown in Fig. 5(*b*) and the inferred formula is provided in Appendix B, which clearly reveals the logic the ego should obey. Since we apply a more complex inference network structure in this case, we use dual annealing Xiang et al. (1997), a global optimizer to find one good initialization of θ_I and then run our algorithm. In this case, we do not have a ground-truth STL formula or reward function, so we do not compute statistics of the learned policy. But as shown in Fig. 5(*b*), the ego vehicle successfully stops in front of the pedestrian when it detects the deceleration of the other vehicle. Otherwise, it continues driving without slowing down.

Unseen scenarios: We can adapt the policy to similar but unseen scenarios by tuning the inferred formula. Assume that the speed limit is restricted to 4 due to road construction. This is an unseen scenario and both the other and the expert-controlled ego vehicles violate the new rule. We modify the formula ϕ_{sl} by replacing $v_{eg} \leq 10$ with $v_{eg} \leq 4$ and retrain the policy network. Sampled trajectories, shown in Fig. 5(*b*), indicate that the new policy successfully adheres to the updated speed limit while maintaining compliance with previous rules.

7. Discussion

As demonstrated, our method can generate a control policy while simultaneously providing a task description using STL formulas. However, this interpretability comes with trade-offs in performance for several reasons. First, the inference network is not a universal classifier, making it less suitable for tasks that are difficult to describe with an STL formula, such as the Swimmer and Walker tasks in MuJoCo. Our approach is better suited for imitating expert demonstrations under spatial-temporal requirements. Second, the inference network requires more expert demonstrations compared to traditional imitation learning. Small dataset can result in a conservative, under-approximated STL formula. Finally, the policy is trained based on the information provided by the inference network. When negative data become more and more similar to expert demonstrations, the inferred formula may degrade, which negatively impacts the policy's effectiveness. Consequently, a performance score is needed to select the most suitable policy, though our observations suggest that the policy chosen by this score is not always the best one in the training process. We believe that performance could be further improved with a more robust selection criterion.

Our approach prioritizes interpretability, offering insight into the objectives of the expert rather than simply mimicking its behavior. This interpretability enables the integration of safety constraints as supplementary STL formulas. Additionally, it offers flexibility and adaptability; once the task is described, the policy can be fine-tuned in a new but similar environment with the learned or manually adjusted STL formula. This is highly sample-efficient, as there is no need to generate new expert demonstrations, which is typically a challenging aspect of imitation learning.

8. Conclusion and Future Work

We proposed an interpretable imitation learning method that combines STL inference and control synthesis in a generative adversarial manner. We use three case studies to demonstrate that our method can uncover the underlying rules from the expert demonstrations in the form of an STL formula, and learn the policy to satisfy these rules. We also showed that we can manually add and adjust rules to adapt the policy for unseen scenarios. Future work includes improving the computational efficiency and increasing the expressiveness of the inference network.

Acknowledgments

Partial support for this work was provided by the NSF under grant 2422282 and by the AFOSR under grant FA9550-23-1-0529. This work was also supported by Toyota Research Institute (TRI). It, however, reflects solely the opinions and conclusions of its authors, and not TRI or any other Toyota entity. This work was also partly supported by the USARMY Integrated Fire Control Technology program under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and don't necessarily reflect the views of the sponsors.

References

Erfan Aasi, Cristian Ioan Vasile, Mahroo Bahreinian, and Calin Belta. Classification of time-series data using boosted decision trees. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1263–1268. IEEE, 2022.

Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6565–6570. IEEE, 2016.

Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers 2*, pages 147–160. Springer, 2012.

Nasim Baharisangari, Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu, and Zhe Xu. Uncertainty-aware signal temporal logic inference. In *International Workshop on Numerical Software Verification*, pages 61–85. Springer, 2021.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

Nir Baram, Oron Anschel, Itai Caspi, and Shie Mannor. End-to-end differentiable adversarial imitation learning. In *International Conference on Machine Learning*, pages 390–399. PMLR, 2017.

Giuseppe Bombara and Calin Belta. Offline and online learning of signal temporal logic formulae using decision trees. *ACM Transactions on Cyber-Physical Systems*, 5(3):1–23, 2021.

G Brockman. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Gang Chen, Yu Lu, Rong Su, and Zhaodan Kong. Interpretable fault diagnosis of rolling element bearings with temporal logic neural network, 2022.

Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

Yann Gilpin, Vince Kurtz, and Hai Lin. A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters*, 5(1):241–246, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Iman Haghighi, Noushin Mehdipour, Ezio Bartocci, and Calin Belta. Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4361–4366. IEEE, 2019.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20:79–93, 2018.

Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. Telex: Passive stl learning using only positive examples. In *International Conference on Runtime Verification*, pages 208–224. Springer, 2017.

Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 43–52, 2013.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Zhaodan Kong, Austin Jones, and Calin Belta. Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3):1210–1222, 2016.

Karen Leung and Marco Pavone. Semi-supervised trajectory-feedback controller synthesis for signal temporal logic specifications. In *2022 American Control Conference (ACC)*, pages 178–185. IEEE, 2022.

Danyang Li, Mingyu Cai, Cristian-Ioan Vasile, and Roberto Tron. Learning signal temporal logic through neural network for interpretable classification. In *2023 American Control Conference (ACC)*, pages 1907–1914. IEEE, 2023.

Danyang Li, Mingyu Cai, Cristian-Ioan Vasile, and Roberto Tron. Tlinet: Differentiable neural network temporal logic inference. *arXiv preprint arXiv:2405.06670*, 2024.

Wenliang Liu, Noushin Mehdipour, and Calin Belta. Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints. *IEEE Control Systems Letters*, 6:91–96, 2021.

Wenliang Liu, Mirai Nishioka, and Calin Belta. Safe model-based control from signal temporal logic specifications using recurrent neural networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12416–12422. IEEE, 2023.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

Sara Mohammadinejad, Jyotirmoy V Deshmukh, Aniruddh G Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. Interpretable classification of time-series data using efficient enumerative techniques. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2020.

Walter Murray and Kien Ming Ng. An algorithm for nonlinear optimization problems with binary variables. *Computational Optimization and Applications*, 47:257–288, 2010. URL https://api.semanticscholar.org/CorpusID:28009541.

Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1235–1240. IEEE, 2017.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87. IEEE, 2014.

Sadra Sadraddini and Calin Belta. Robust temporal logic model predictive control. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 772–779. IEEE, 2015.

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sarath Sreedharan, Siddharth Srivastava, and Subbarao Kambhampati. Tldr: Policy summarization for factored ssp problems using temporal abstractions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 272–280, 2020.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. *Advances in Neural Information Processing Systems*, 30, 2017.

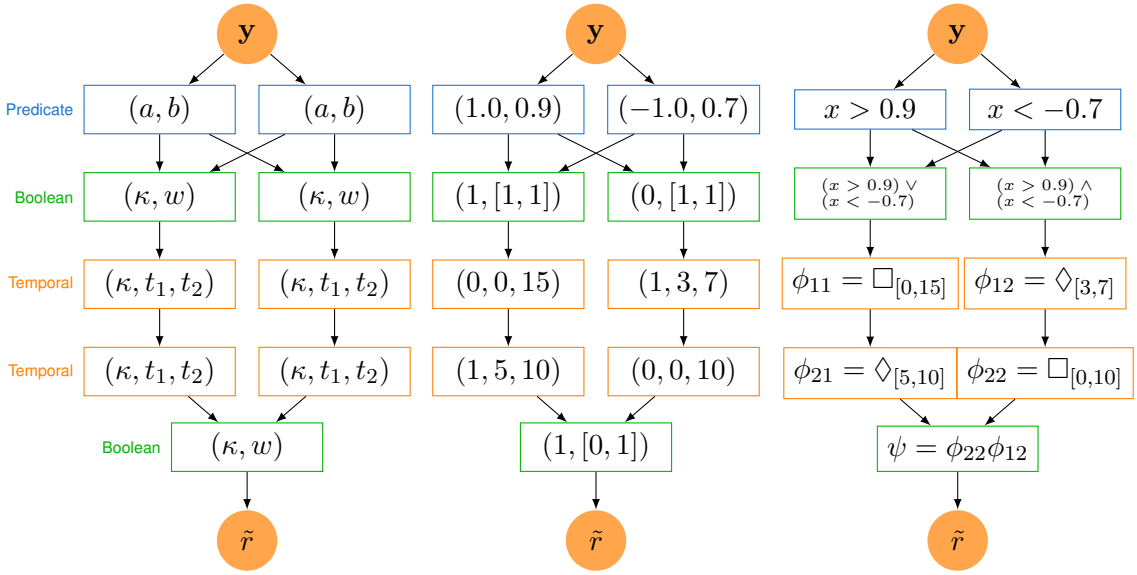
Yang Xiang, DY Sun, W Fan, and XG Gong. Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220, 1997.

Zhe Xu, Sayan Saha, Botao Hu, Sandipan Mishra, and A Agung Julius. Advisory temporal logic inference and controller design for semiautonomous robots. *IEEE Transactions on Automation Science and Engineering*, 16(1):459–477, 2018.

Shakiba Yaghoubi and Georgios Fainekos. Worst-case satisfaction of stl specifications using feed-forward neural network controllers: a lagrange multipliers approach. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–20, 2019.

Appendix A. Decoding and Translating TLINet

Consider a TLINet with five layers, as illustrated in Figure 6. In this example, the network begins with a predicate layer with two predicate modules, followed by a Boolean layer with two modules, then two temporal layers each with two modules, and finally a Boolean layer containing a single module. The parameters include spatial parameters a and b, a selector κ for the type of operator, temporal parameters t_1 and t_2, and Boolean parameter w that determines module activation. For instance, in the final layer, $w = [0, 1]$ indicates that the first module is deactivated and the second module is activated in this Boolean operation. The structure of the STL formula is determined by the activation of modules, guided by the learned Boolean parameters. Such a network can be succinctly translated into an STL formula $\psi = \square_{[0,10]}\lozenge_{[3,7]}((x > 0.9) \wedge (x < -0.7))$.



(*a*) Initialize TLINet structure. (*b*) Learned TLINet parameters. (*c*) TLINet modules to STL subformulas.

Figure 6: An example of a 5-layer TLINet and how it can be transferred to an STL formula from learning parameters.

Appendix B. Experiments Setup and Detailed Results

The inference network used in the Sec. 6.1 and Sec. 6.2 consists of three layers: a predicate layer with $2n_f$ modules, a temporal layer with $2n_f$ modules, and a Boolean layer with 1 modules, where n_f is the dimension of the features. In the self-driving case in Sec. 6.3, we observe improved performance with the addition of a second Boolean layer, so we use a similar structure with an extra Boolean layer. The policy network uses an LSTM Hochreiter (1997) with 3 hidden layers, each containing 64 nodes.

The STL formulas learned from different initializations are slightly different. In the following, we show one of them for each scenario. We also demonstrate some negative trajectories generated by the policies during the training to visualize the training processes.

B.1. Random Navigation

In Sec. 6.1, the features on which the STL formula is defined are the position of the robot and the distances from the robot to the centers of the regions, i.e., $y = h(x) = [p_x, p_y, d_{g_1}, d_{g_2}, d_o]$. The ground-truth STL formula that the expert policy is trained on is:

$$\begin{aligned} &\square_{[0,24]}(p_y < 4) \wedge \square_{[0,24]}(p_y > 0) \\ \wedge &\lozenge_{[0,8]}(d_{g_1} < 0.5) \wedge \lozenge_{[8,24]}(d_{g_2} < 0.5) \wedge \square_{[0,24]}(d_o > 0.5) \end{aligned} \tag{7}$$

A learned STL formula from one of the random initializations is given as below:

$$\begin{aligned} &\square_{[0,24]}(p_y < 3.7) \wedge \square_{[1,17]}(p_y > 0.41) \wedge \lozenge_{[4,9]}(d_{g_1} < 0.41) \\ \wedge &\lozenge_{[0,10]}(d_{g_1} < 2.71) \wedge \lozenge_{[8,24]}(d_{g_2} < 0.48) \wedge \square_{[8,20]}(d_o > 0.78), \end{aligned} \tag{8}$$

which closely reproduces the specification the expert aims to satisfy.

Some negative trajectories generated by the policy during training under same initial and environment states are shown in Fig. 7. At iteration 0, random trajectories are sampled and labeled as negative examples. These samples are insufficient to push the decision boundary near the expert demonstrations, resulting in a control policy that fails to replicate expert behavior. As training progresses and more informative negative examples are added, the policy-generated trajectories gradually converge toward the expert's behavior.

B.2. MuJoCo Tasks

For the Inverted Pendulum in Sec. 6.2, the features y and system state x are identical, both of them are same as the observation provided in OpenAI Gym Brockman (2016), i.e., $y = x = h(x) = [p, \theta, v, \omega]$, where p and v are the position and velocity of the cart along the linear surface, θ and ω are the angle and angular velocity of the pendulum. One of the learned STL formula is given as:

$$\begin{aligned} &\square_{[21,45]}(\theta < 0.07) \wedge \square_{[23,32]}(v > -0.17) \\ \wedge &\square_{[14,50]}(\omega > -0.79) \wedge \lozenge_{[19,37]}(\omega > -0.16) \end{aligned} \tag{9}$$

The first subformula $\square_{[21,45]}(\theta < 0.07)$ requires that the pendulum must not tilt in the positive direction, while the rest of the formula specifies additional constraints for the system.

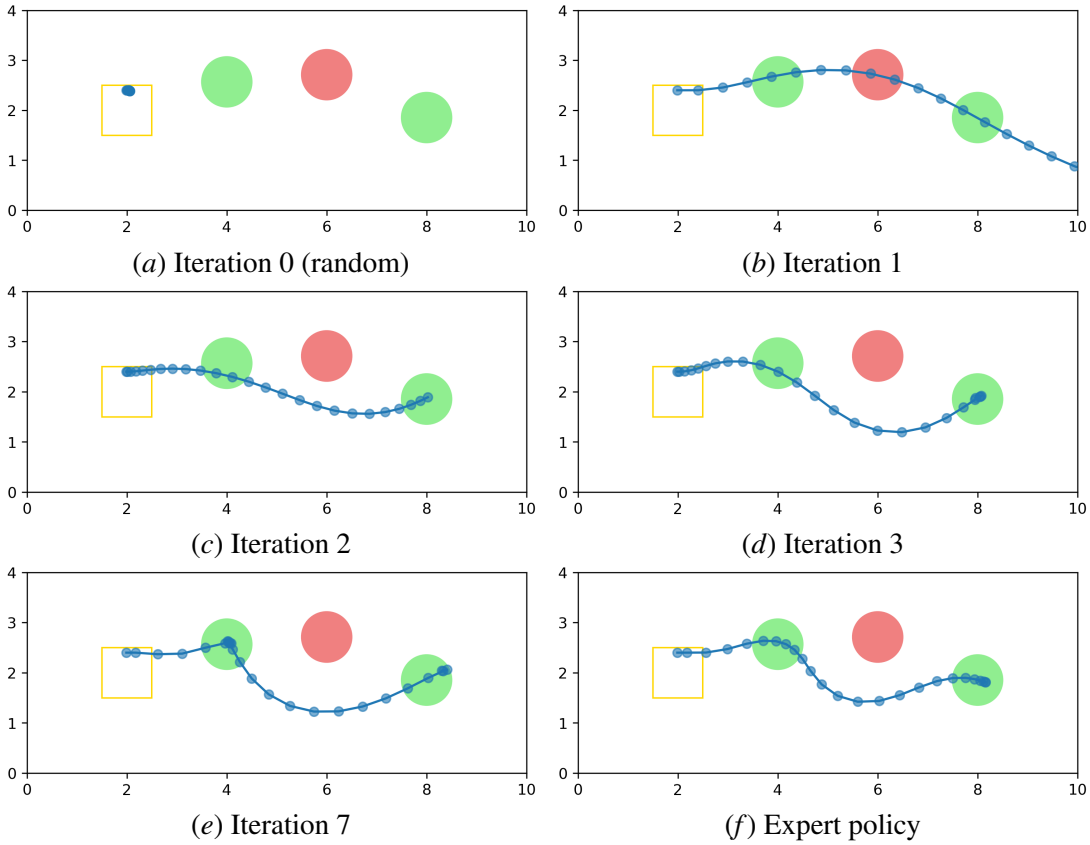
For the Reacher, the system state, including agent state and environment state, is same as the observation given by Gym:

$$x = [\cos\theta_1, \cos\theta_2, \sin\theta_1, \sin\theta_2, p_x, p_y, \omega_1, \omega_2, d_x, d_y, d_z] \in \mathbb{R}^{11},$$

where θ_1 and θ_2 are the angles of the first and second arms, ω_1 and ω_2 are their angular velocities, p_x and p_y are the coordinates of the target, d_x, d_y and d_z are the differences between the target and the fingertip in the x, y, z directions. The features we selected are $y = h(x) = [\cos\theta_1, \cos\theta_2, \sin\theta_1, \sin\theta_2, \omega_1, \omega_2, d_x, d_y]$. One of the learned STL formula is:

$$\begin{aligned} &\square_{[22,50]}(d_x < 0.04) \wedge \square_{[10,50]}(d_x > -0.04) \\ \wedge &\lozenge_{[16,23]}(d_y < 0.04) \wedge \square_{[15,28]}(d_y > -0.03) \\ \wedge &\lozenge_{[23,50]}(\sin\theta_2 > 0.38) \wedge \lozenge_{[19,37]}(\omega_2 < 0.4), \end{aligned} \tag{10}$$

The first 4 subformulas in (10) require the fingertip to reach the target, while the last 2 subformulas describe how the expert accomplishes this as the inverse kinematics does not have a unique solution.



(*a*) Iteration 0 (random) (*b*) Iteration 1

(*c*) Iteration 2 (*d*) Iteration 3

(*e*) Iteration 7 (*f*) Expert policy

Figure 7: Trajectories generated by the policy during the training process.

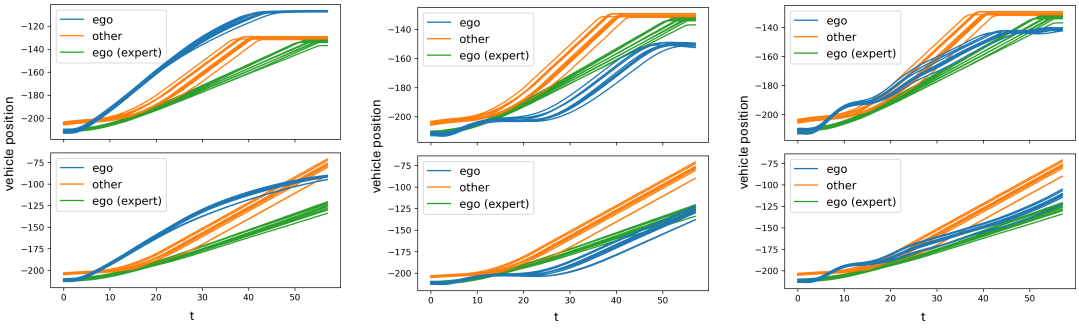
B.3. A Self-Driving Scenario

In Sec. 6.3, the features are identical with the system state, i.e., $y = h(x) = x$. The learned STL formula ϕ_{θ_I} is:

$$\begin{aligned}&\Big(\square_{[45,47]}(v_{ot} > 2.35) \wedge \square_{[20,57]}(v_{eg} > 1.31) \wedge \square_{[24,46]}(v_{eg} < 5.55)\Big) \vee \\ &\Big(\Diamond_{[42,55]}(v_{ot} < 2.94) \wedge \Diamond_{[20,57]}(v_{eg} < 0.01) \wedge \square_{[24,46]}(v_{eg} < 5.55)\Big).\end{aligned} \tag{11}$$

Formula (11) can be translated as "if v_{ot} is always greater than 2.35 over the time window $[45, 47]$ (the other vehicle does not stop), ego's velocity should always be greater than 1.31. If v_{ot} eventually becomes less than 2.94 in the time interval $[42, 55]$ (the other vehicle decelerates), ego's velocity should also be eventually less than 0.01 between $[20, 57]$, which means a full stop". In both cases, v_{eg} should be always less than 5.55 in the time window $[24, 46]$. This formula explicitly reflects the rules that the ego should obey.

Some sampled trajectories of ego and other vehicles' positions generated by the learned policy during the training process are shown in Fig. 8. Again, the trajectories increasingly align with the expert demonstrations, indicating progressive improvement in policy performance.



(*a*) Iteration 1 (*b*) Iteration 2 (*c*) Iteration 3

Figure 8: (a)-(c): Sampled trajectories of ego and other vehicles' positions generated by the policy network after each iteration. y-axes are the positions of the ego and other vehicles. Blue curves stand for ego and orange curves stand for the other vehicle. Green curves are expert demonstrations of the ego. The top figures show the situations when there are pedestrians, while the bottom figures show the situations when there is no pedestrian.