Signal Temporal Logic Neural Predictive Control

Yue Meng and Chuchu Fan, *Member, IEEE*

***Abstract*—Ensuring safety and meeting temporal specifications are critical challenges for long-term robotic tasks. Signal temporal logic (STL) has been widely used to systematically and rigorously specify these requirements. However, traditional methods of finding the control policy under those STL requirements are computationally complex and not scalable to high-dimensional or systems with complex nonlinear dynamics. Reinforcement learning (RL) methods can learn the policy to satisfy the STL specifications via hand-crafted or STL-inspired rewards, but might encounter unexpected behaviors due to ambiguity and sparsity in the reward. In this letter, we propose a method to directly learn a neural network controller to satisfy the requirements specified in STL. Our controller learns to roll out trajectories to maximize the STL robustness score in training. In testing, similar to Model Predictive Control (MPC), the learned controller predicts a trajectory within a planning horizon to ensure the satisfaction of the STL requirement in deployment. A backup policy is designed to ensure safety when our controller fails. Our approach can adapt to various initial conditions and environmental parameters. We conduct experiments on six tasks, where our method with the backup policy outperforms the classical methods (MPC, STL-solver), model-free and model-based RL methods in STL satisfaction rate, especially on tasks with complex STL specifications while being 10X-100X faster than the classical methods.**

***Index Terms*—Motion and path planning, machine learning for robot control, AI-based methods.**

I. Introduction

LEARNING to control a robot to satisfy long-term and complex safety requirements and temporal specifications is critical in autonomous systems and artificial intelligence. For example, the vehicles should make a complete stop before entering the intersection with a stop sign, wait a few seconds, and then drive through it if no other cars are there. And a robot navigating through obstacles to reach the destination should always reach and stay at a charging station for a while to get charged when it is low on battery.

However, designing the controller to satisfy those specifications is challenging. Traditional rule-based methods often require expert knowledge and several rounds of trial and error to find the best design to handle the problem. Other learning-based approaches either learn from demonstrations or rewards to find the control policy to satisfy the behavior specification. Those methods need plenty of expert data or great effort in reward design (an improper reward will result in a learned policy to generate unexpected behaviors)

To let the controller satisfy the exact behaviors, another direction of solving this problem is to describe the requirements in Signal Temporal Logic (STL) and find out the feasible plan by solving an online optimization problem. Mixed-integer linear programming (MILP) [1] is proposed to handle simple dynamics and easy-to-evaluate atomic propositions, which has exponential complexity and is hard to solve. For more complicated systems, gradient-based [2] and sampling-based methods (such as Cross-Entropy Method (CEM) and covariance matrix adaptation evolution strategy (CMA-ES) [3]) are proposed to synthesize controllers to maximize the STL robustness score (which measures how well the STL is satisfied). However, they still need to solve the problem online for each initial state condition and each scenario, which limits their usage for more general cases.

Motivated by the line of work in robustness score [4] and controller synthesis [2], we propose Signal Temporal Logic Neural Predictive Control, which learns a Neural Network (NN) controller to generate STL-satisfied trajectories. The "STL solving" process is conducted in training, and in the test phase, we use the trained controller (potentially with a backup policy) to roll out trajectories. Thus, we do not need the heavy online optimization/searching required for those gradient-based or sampling-based methods.

The whole pipeline is as follows: We construct an NN controller and sample from the initial state (which might include environment information) distribution. In training, we roll out trajectories using the NN controller. We evaluate the approximated robustness score on those trajectories to maximize the robustness score. In testing, we follow the MPC procedure: our learned controller predicts a trajectory that is safe/rule-satisfied in the short-term horizon, and we pick the first (or first several) actions in the deployment. Finally, when the learned controller is detected violating the STL constraints, a sampling-based backup policy is triggered to guarantee the robot's safety. We conduct experiments on six tasks shown in Fig. 1: driving near intersections, reach-and-avoid problem, safe ship control, safe ship tracking control, robot navigation and manipulation. On tasks with simple dynamics or simple STL, our approach is on par with the RL methods in terms of STL accuracy and we surpass traditional methods such as MPC and STL solvers. We achieve the highest STL accuracy on hard tasks such as ship safe

Manuscript received 9 May 2023; accepted 4 September 2023. Date of publication 14 September 2023; date of current version 12 October 2023. This letter was recommended for publication by Associate Editor R. Lioutikov and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported in part by MIT-Ford Alliance Program, and in part by National Science Foundation CAREER under Award CCF-2238030. *(Corresponding author: Yue Meng.)*

The authors are with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: mengyue@mit.edu; chuchu@mit.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3315536, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3315536

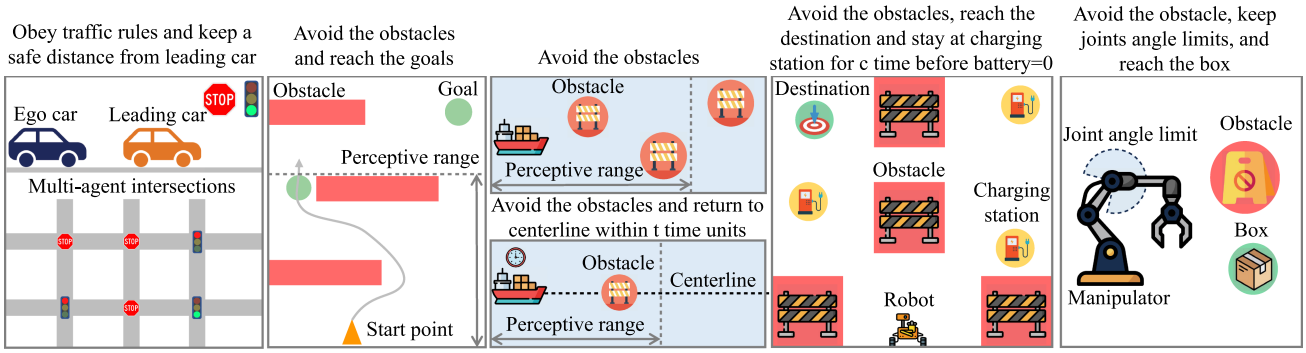


Fig. 1. Benchmarks: learning traffic rules, reach-and-avoid game, ship safe/tracking control, navigation, and manipulation.

tracking control and robot navigation tasks, $20\% \sim 40\%$ higher than the second best approach. Our training time is similar to RL, and our inference speed is 1/10-1/100X faster than classical methods.

Our contributions are: (1) we are the first to use NN controllers to predict trajectories to satisfy STL in a self-supervised manner (without demonstrations) (2) we propose a backup policy to ensure the safety of the robot when the learned policy fails to generate STL-satisfied trajectories (3) we conduct challenging experiments with complicated dynamics and STL constraints and outperform other baselines.

II. Related Work

Temporal logic (LTL [5], MTL [6], and STL [7]) can express rich and complex robot behaviors and hence is useful for controller verification and synthesis. Plenty of motion planning algorithms have been aiming to fulfill STL specifications. We refer the readers to this survey [8].

Abstraction-based methods [9], [10], [11] emerged the earliest, where an automaton or a graph is constructed and the search-based planning is conducted on this discrete form of abstraction. These methods often require domain knowledge and are challenging to construct automatically.

The optimization-based approach came out for specific dynamics (e.g., linear), where the STL specifications are modeled as linear constraints and the control policy is found via Convex Quadratic Programming [12] or Mixed Integer Programming (MIP) [13]. Though they do not need discretization or domain expertise, the computation cost is still too high and they cannot solve complicated systems. As a remedy, [1] plans feasible line segments and then uses tracking controllers to adapt to complex dynamics, and [14] uses separation principles to boost the MILP-solving process.

To better handle complex dynamics, sampling-based approaches are proposed which are computationally efficient and suitable for real-time applications. Representative works include STyLuS* [15] that uses biased sampling and its concurrent works [16], [17], [18] which use RRT or RRT*. Another line of work to cope with nonlinear dynamics is gradient-based methods. A differentiable measure for MTL satisfaction is developed in [19]. Inspired by it, STL-cg [20] handles backpropagation for (parametrized) STL formulas under machine learning frameworks. The work [21] learns STL for complex satellite mission planning. The work [2] provides a counter-example guided framework to learn robust control policies. However, gradient descent is well-known to be slow and might stuck into the local minimum. Both types of methods in this section may not guarantee the optimality or completeness of the solution.

Recently, machine learning has been widely used for STL/LTL controller synthesis, where neural networks (NN) or other forms of parametrized policy are trained to satisfy STL. These works can be further categorized into model-free reinforcement learning (RL) [22], [23], [24], model-based RL (MBRL) [3], [25], [26], and imitation learning [27], [28], [29]. Our method is similar to MBRL, where we train an NN controller via stochastic gradient descent (SGD) to predict policy sequences to maximize the robustness score.

III. Preliminaries and Problem Definition

A. Controlled Hybrid System

Consider a continuous-time hybrid system:

$$\begin{cases} \dot{x} = f(x,u), & x \in \mathcal{C} \\ x^+ = h(x^-), & x \in \mathcal{D} \end{cases} \tag{1}$$

where $x \in \mathbb{R}^n$ denotes the system state and $u \in \mathbb{R}^m$ denotes the control input. The system state here can contain both the agent and environment information. The set $\mathcal{C}$ is the flow set where states follow a continuous flow map and $\mathcal{D}$ is the jump set where states encounter instantaneous changes. A jump captures the scenarios where the agent updates its local observations or resets timers. Here we consider the states and controls at discrete time steps with time horizon T. Given an initial state x_0 and a control sequence $(u_0, u_1, \ldots, u_{T-1})$, following the system dynamics we can generate a trajectory $(x_0, x_1, \ldots, x_T)$. In our context, we call this trajectory a trace (or signal) and denote it as s. In this letter, we aim to learn a policy that can generate traces satisfying temporal logic properties. The temporal requirements rbtare formally introduced below.

B. Signal Temporal Logic (STL)

An STL formula comprises predicates, logical connectives, and temporal operators [30]. The predicates are of the form $\mu(x) \geq 0$ where $\mu : \mathbb{R}^n \to \mathbb{R}$ is a function with the state x as the

input and returns a scalar value. STL formulas are constructed in the Backus-Naur form:

$$\phi ::= \top \mid \mu \geq 0 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathrm{U}_{[a,b]} \phi_2 \tag{2}$$

where $\top$ means "true", $\neg$ means "negation", $\wedge$ means "and", U means "until" and $[a, b]$ is the time interval from a to b. Other operators can be written from the elementary operators above, such as "or": $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, "infer": $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$, "eventually": $\Diamond_{[a,b]}\phi = \top \mathrm{U}_{[a,b]}\phi$ and "always": $\Box_{[a,b]}\phi = \neg\Diamond_{[a,b]}\neg\phi$. We denote $s, t \models \phi$ if a signal s at time t satisfies an STL formula ϕ. The detailed Boolean semantics in [7] is iteratively defined as:

$$\begin{aligned}
&s,t \models \top && \text{(naturally satisfied)}\\
&s,t \models \mu \geq 0 && \Leftrightarrow \quad \mu(s(t)) > 0\\
&s,t \models \neg\phi && \Leftrightarrow \quad s,t \not\models \phi\\
&s,t \models \phi_1 \wedge \phi_2 && \Leftrightarrow \quad s,t \models \phi_1 \text{ and } s,t \models \phi_2\\
&s,t \models \phi_1 \vee \phi_2 && \Leftrightarrow \quad s,t \models \phi_1 \text{ or } s,t \models \phi_2\\
&s,t \models \phi_1 \Rightarrow \phi_2 && \Leftrightarrow \quad \text{if } s,t \models \phi_1 \text{ then } s,t \models \phi_2\\
&s,t \models \phi_1 \mathrm{U}_{[a,b]} \phi_2 && \Leftrightarrow \quad \exists t' \in [t+a, t+b] \text{ s.t. } s,t' \models \phi_2\\
&&& \quad \text{and } \forall t'' \in [t, t'] \ s,t'' \models \phi_1\\
&s,t \models \Diamond_{[a,b]}\phi && \Leftrightarrow \quad \exists t' \in [t+a, t+b] \ s,t' \models \phi\\
&s,t \models \Box_{[a,b]}\phi && \Leftrightarrow \quad \forall t' \in [t+a, t+b] \ s,t' \models \phi
\end{aligned} \tag{3}$$

To measure how well the trace satisfies the STL formula, [4] proposes a quantitative semantics called robustness score ρ: the STL formula is satisfied if $\rho > 0$ and is violated if $\rho < 0$, and a larger ρ reflects a larger margin of satisfaction. The robustness score is calculated with the following rules:

$$\begin{aligned}
\rho(s,t,\top) &= 1, \quad \rho(s,t,\mu \geq 0) = \mu(s(t))\\
\rho(s,t,\neg\phi) &= -\rho(s,t,\phi)\\
\rho(s,t,\phi_1 \wedge \phi_2) &= \min\{\rho(s,t,\phi_1), \rho(s,t,\phi_2)\}\\
\rho(s,t,\phi_1 \vee \phi_2) &= \max\{\rho(s,t,\phi_1), \rho(s,t,\phi_2)\}\\
\rho(s,t,\phi_1 \Rightarrow \phi_2) &= \max\{-\rho(s,t,\phi_1), \rho(s,t,\phi_2)\}\\
\rho(s,t,\phi_1 \mathrm{U}_{[a,b]} \phi_2) &= \sup_{t' \in [t+a, t+b]}\\
&\quad \times \min\left\{\rho(s,t',\phi_2), \inf_{t'' \in [t,t']} \rho(s,t'',\phi_1)\right\}\\
\rho(s,t,\Diamond_{[a,b]}\phi) &= \sup_{t' \in [t+a, t+b]} \rho(s,t',\phi)\\
\rho(s,t,\Box_{[a,b]}\phi) &= \inf_{t' \in [t+a, t+b]} \rho(s,t',\phi)
\end{aligned} \tag{4}$$

C. Problem Formulation

With the definition of the dynamical system and STL specifications, the problem under consideration is as follows.

Problem 1: Given a system model in (1), an STL formula ϕ in (2) and an initial state set $\mathcal{X}_0 \subset \mathbb{R}^n$, find a control policy π such that starting from any state $x \in \mathcal{X}_0$, the trajectory $s_\pi(x)$ of the resulting closed-loop system satisfies the formula ϕ, i.e., $\forall x \in \mathcal{X}_0$, $s_\pi(x) \models \phi$.

TABLE I
DIFFERENT SETUPS IN TRAINING UNDER CAR BENCHMARK

(a) Satisfication threshold γ

γ	Train Acc.	Val. Acc.
0.0	0.855	0.788
0.1	0.920	0.912
0.2	0.918	0.915
0.5	0.957	0.957
0.8	0.957	0.958
1.0	0.958	0.955

(b) Scaling factor k

k	Train Acc.	Val. Acc.
1	0.542	0.542
10	0.950	0.946
100	0.957	0.957
1000	0.958	0.957
10000	0.956	0.956

(c) Neural network size (#neurons x #layers)

Size	Train Acc.	Val. Acc.
32x2	0.951	0.954
32x3	0.951	0.950
64x2	0.951	0.951
64x3	0.951	0.956
128x2	0.954	0.949
128x3	0.956	0.952
256x2	0.952	0.951
256x3	0.957	0.957

(d) Training samples N

N	Train Acc.	Val. Acc.
200	0.800	0.663
400	0.755	0.651
800	0.941	0.836
1600	0.947	0.874
3200	0.952	0.903
6400	0.962	0.920
12800	0.957	0.942
25600	0.954	0.953
50000	0.957	0.957

IV. METHODOLOGY

A. STL Satisfaction as an Optimization Problem

To satisfy ϕ, we measure the satisfaction rate of ϕ using the robustness score and thus form the boolean STL satisfaction task as an optimization problem. Denote the policy parametrized with θ as π_θ. For $x \in \mathcal{X}_0$, the policy predicts a sequence of controls: $\pi_\theta(x) = u_{0:T-1}$. Following system dynamics, we can generate the trajectory $s_{\pi_\theta}(x)$ with horizon $T+1$. Then we compute the robustness score $\rho(s_{\pi_\theta}(x), \phi)$ for the STL formula ϕ on the trajectory $s_{\pi_\theta}(x)$ at time 0 (we omit t for brevity). Our goal is: (omit dynamic constraints)

$$\text{Find } \pi_\theta, \text{s.t. } \rho(s_{\pi_\theta(x)}, \phi) > 0, \forall x \in \mathcal{X}_0 \tag{5}$$

Assuming x follows uniform distribution on $\mathcal{X}_0$, we aim to find the optimal policy which maximizes the expected truncation robustness score:

$$\pi_\theta^* = \arg\max_{\pi_\theta} \mathop{\mathbb{E}}_{x \sim \mathcal{X}_0} \left[\min\left\{\rho(s_{\pi_\theta(x)}, \phi), \gamma\right\}\right] \tag{6}$$

where $\gamma > 0$ is the truncation factor. The min operator in the expectation encourages the policy to improve "hard" trajectories (with robustness scores $< \gamma$) rather than further increasing "easy" trajectories that already achieve high robustness scores ($\geq \gamma$). This helps achieve high robustness score for all possible cases. In addition, if the optimal value is γ, ϕ is guaranteed to be satisfied on all sampled initial states. In the experiments, we set $\gamma = 0.5$. An ablation study on γ is in Table I.

B. Neural Network Controller Learning for STL Satisfication

We aim to solve (6) using neural networks. Unlike [2], which solves the STL satisfaction online, we learn the control policy in training, and thus, our approach can run in real-time in testing. We use a fully-connected network (MLP) π_θ to represent the control policy. At each training step, we first sample initial states from $\mathcal{X}_0$ and use π_θ to predict a sequence of actions. Then we roll-out trajectories based on these actions and calculate the robustness score to form a loss function shown in (6). Finally, we update the parameters of the neural network controller guided by the loss function via stochastic gradient descent. However, there remain two challenges for us to apply the gradient method. First, the hybrid systems in (1) are non-differentiable at the mode-switching instant. Secondly, ρ is not differentiable due to the max (min) and sup (inf) operators in (4).

To tackle the non-smoothness in the hybrid systems dynamics, we assume a membership function $I_C : \mathcal{X} \to \mathbb{R}$ exists to imply whether a state x_t is in the flow set ($I_C(x_t) > 0$) or the jump set ($I_C(x_t) < 0$). Thus the forward dynamics can be written as $x_{t+1} = \mathbb{1}\{I_C(x_t) > 0\} f(x_t, u_t)\Delta t + (1 - \mathbb{1}\{I_C(x_t) > 0\})h(x_t)$. Next, we approximate the $\mathbb{1}(I_C(x_t) > 0)$ using $\widetilde{I}_w(x_t) = (1 + \text{Tanh}(w \cdot I_C(x_t)))/2$ with a scaling factor $w > 0$ to control the approximation ($\lim_{w\to\infty} \widetilde{I}_w(x_t) = \mathbb{1}(I_C(x_t) > 0)$). The dynamics now is:

$$x_{t+1} = \widetilde{I}_w(x_t) f(x_t, u_t)\Delta t + (1 - \widetilde{I}_w(x_t))h(x_t) \tag{7}$$

and we can learn STL satisfaction from this hybrid system.

To backpropagate the gradient through STL, we use the approximated robustness score $\tilde{\rho}$ [19], which replaces the max (min) and sup (inf) operators with smooth max (min):

$$\begin{aligned} \widetilde{\max}_k(x_1, x_2, \ldots) &:= \frac{1}{k} \log\left(e^{kx_1} + e^{kx_2} + \cdots\right) \\ \widetilde{\min}_k(x_1, x_2, \ldots) &:= -\widetilde{\max}_k(-x_1, -x_2, \ldots) \end{aligned} \tag{8}$$

where k is a scaling factor for this approximation. If $k \to \infty$, the operator $\widetilde{\max} = \max$ and similarly $\widetilde{\min} = \min$. We use $k = 500$ in our training. An ablation study on k is in Table I.

Now the framework is differentiable for both the STL robustness score calculation and the system dynamics, we encode the objective in (6) using the loss function:

$$\mathcal{L}_{\text{STL}} = \frac{1}{|\mathcal{D}_0|} \sum_{x \in \mathcal{D}_0} \max(0, \gamma - \tilde{\rho}(s_{\pi_\theta(x)}, \phi)) \tag{9}$$

where a finite number of states x are uniformly sampled from $\mathcal{X}_0$ to form the training set $\mathcal{D}_0$. Aside from constraint satisfaction, the agent might also need to maximize some performance indices (e.g., "reach the destination as fast as possible.") We hence form the following loss function:

$$\mathcal{L}_{Total} = \mathcal{L}_{Perf} + \lambda \mathcal{L}_{\text{STL}} \tag{10}$$

where $\lambda > 0$ weighs the performance objective $\mathcal{L}_{Perf}$ and the STL violations $\mathcal{L}_{STL}$. We admit the $\lambda\mathcal{L}_{\text{STL}}$ term cannot guarantee the learned policy always satisfy the STL requirement, but empirically we found out this can bring high STL satisfaction rate without much effort in hyperparameter tuning. The $\mathcal{L}_{Perf}$ is often the Euclidean distance between state x_t (in the trajectories starting from $x \sim \mathcal{D}_0$) and goal state x^*, i.e., $\mathcal{L}_{Perf} = \frac{1}{|\mathcal{D}_0|(T+1)} \sum_{x\sim\mathcal{D}_0} |x_t - x^*|$.

C. MPC-Based Deployment With a Backup Policy

In testing, we follow an online MPC manner. At each time step t, the controller π_θ receives the state x_t and predicts a sequence of commands $u_{0:T-1}$, then we choose the first command u_0 for the agent to execute. Ideally, this policy will satisfy the STL constraints. However, this might not hold in testing due to: (1) imperfect training and (2) out-of-distribution scenario. Thus, we propose a backup policy. We monitor whether the predicted trajectory satisfies the STL specification. If a violation occurs, we sample M trajectories in length $T_0 + 1$ with $T_0 < T$. For the i-th trajectory $\tilde{\xi}_i = \{x_0^i, x_1^i, \ldots x_{T_0}^i\}$, we evaluate our controller at the final state $x_{T_0}^i$ and rollout a trajectory $\hat{\xi}_i = \{x_{T_0+1}^i, x_{T_0+2}^i \ldots, x_T^i\}$ (we only keep the first $T - T_0$ timesteps). Since Neural Networks allow batch operations, all the sampled trajectories $\{\tilde{\xi}_i\}_{i=1}^M$ can be efficiently processed in one forward step to get $\{\hat{\xi}_i\}_{i=1}^M$. Finally, we select the trajectory $\xi_i = (\tilde{\xi}_i, \hat{\xi}_i) = \{x_0^i, \ldots, x_{T_0}^i, x_{T_0+1}^i, \ldots x_T^i\}$ with the highest robustness score and pick its first action to execute.

If this trajectory still cannot satisfy the STL specification, we choose a trajectory that only satisfies the safety condition:

$$\underset{i}{\text{argmax}}\, \rho(\xi_i, t, \phi), \text{ s.t. } \xi_i, t| = \phi_{\texttt{safe}} \tag{11}$$

where $\phi_{\texttt{safe}}$ contains all the safety constraints in ϕ. We start with $T_0 = 1$, obtain ξ_i using the above method, and gradually increase T_0 until a solution for (11) is found. If there is still no feasible solution after $T_0 = T$, we choose ξ_i with the longest sub-trajectory starting from x_0^i that satisfies $\phi_{\texttt{safe}}$ and pick the first action to execute. This ensures at least $\phi_{\texttt{safe}}$ are satisfied and the agent can recover to satisfy ϕ in the earliest time. we discretize the action space to L bins, and hence the sampling size is $M = L^{T_0}$. We prove the backup policy has a probabilistic guarantee to find a feasible solution.

Theorem 1: Assume that a policy u^* exists with a δ-radius neighborhood satisfying constraints ϕ, i.e., $u \models \phi$, $\forall u \in \{u | \max_{t,i} |u_{t,i} - u^*_{t,i}|_1 \le \delta\}$, and each step's policy u_t^* is uniformly distributed in $\prod_k [u_i^{\texttt{min}}, u_i^{\texttt{max}}]$. The probability our algorithm finds a solution is: $\min\{1, (\frac{((2L-4)\delta)^{mT}}{\prod_{i=1}^m (u_i^{\texttt{max}} - u_i^{\texttt{min}})^T}\}$.

Proof 1: We sample from a grid in the policy space at each time step. The probability that a solution can be found is equal to that the δ-region contains a grid point at each time step, which is greater than or equal to the probability that the union of the δ-hypercube centered at all grids covers the policy u^*. The volume of the policy space at each time step is $\prod_{i=1}^m (u_i^{\texttt{max}} - u_i^{\texttt{min}})$. Each hypercube has a side length 2δ. Thus the volume of the union of the hypercubes is greater than (as we omit the cubes that are at the boundary of the policy space) $(L-2)^m (2\delta)^m$. Thus the probability of the union covering u_t^* is $\min\{1, \frac{((2L-4)\delta)^m}{\prod_{i=1}^m (u_i^{\texttt{max}} - u_i^{\texttt{min}})}\}$, and for T steps, the probability is powered by T to derive expected result shown in Theorem 1.

D. Remarks on the STL Constraints

To handle different configurations, we augment the system states with additional parameters such as obstacle radius, locations and time constraints. These parameters stay constant unless encountering a reset. The policy learned from this augmentation can solve a family of STL formulas and can adapt to unseen configurations without further fine-tuning.

A wide range of requirements commonly used in robot tasks can be represented by STL, owing to the flexible form of atomic propositions (AP). Denote the 2D location of a robot as $x_{[0:2]} \in \mathbb{R}^2$. We use $\mu(x) = r - ||x_{[0:2]} - x_{\texttt{obs}}||_2$ to check collision with a round obstacle at $x_{\texttt{obs}} \in \mathbb{R}^2$ with radius $r \in \mathbb{R}$, where $||\cdot||_2$ represents the Euclidean norm. For a polygon region $S = \{y : Ay \leq b\}$ with $A \in \mathbb{R}^{k\times 2}$ and $b \in \mathbb{R}^k$, $\mu(x) = b - Ax_{[0:2]}$ can check whether the robot is in S. We use $\mu(x) = -(x_{[0]} - x_{\texttt{min}})(x_{[0]} - x_{\texttt{max}})$ to check whether the agent's x-coordinate is in the interval $[x_{\texttt{min}}, x_{\texttt{max}}]$. Furthermore, we can evaluate whether a system mode has been activated by checking an indicator $I_1 \in \{0, 1\}$ via AP: $\mu(I_1) = I_1 - 0.5$.

However, it is hard to cope with constraints with a time interval. Consider $\phi = \Box_{[5,10]}\text{Stay}(A)$ which means *"Always stay at A in the time interval [5,10]"*. One step later, the agent will still try to satisfy $\Box_{[5,10]}\text{Stay}(A)$, which actually should be updated to $\Box_{[4,9]}\text{Stay}(A)$. Thus we need our policy to be aware of the STL time interval updates. We augment the system state with the timer variables and transform the original STL formula to extra STL constraints on those timer variables. To be more specific, the timer variables follow the dynamic: $\tau_{t+1} = \tau_t + \Delta t$ and the extra STL constraint is $\Box_{[0,T]}((5 \leq \tau \leq 10) \rightarrow \text{Stay}(A))$.

V. Experiments

We conduct experiments with diverse dynamics and task specifications. Our method's training time is only 0.5X the time needed for training RL. In testing, our method is on par with the best baseline on simple benchmarks and achieves the highest STL accuracy on more complicated cases. As for runtime, our approach (without backup policy) is 10X-100X faster than classic planning methods such as MPC and MILP.

A. Experiment Setups

Baselines: We compare with RL, model-based RL (MBRL), and classical approaches. For RL, we train Soft Actor Critic [31] [32] under five random seeds with varied rewards. $\mathbf{RL_R}$: uses a hand-crafted reward. $\mathbf{RL_S}$: uses STL robustness score as the reward. $\mathbf{RL_A}$: uses STL accuracy as the reward. The MBRL baselines are **MBPO**: [33] with STL accuracy as the reward, **PETS**: [34] with a hand-crafted reward, and **CEM**: Cross Entropy Method [35] with STL robustness reward. The rests are **MPC**: Model predictive control via Casadi [36] for nonlinear systems and Gurobi [37] for linear dynamics, $\mathbf{STL_M}$: An official implementation for STL-MiLP [1] with a PD control for nonlinear dynamics if needed, and $\mathbf{STL_G}$: A gradient-based method similar to [2].

Implementation details: For our method, we set $\gamma = 0.5$, $k = 500$, $T \in [10, 25]$ and $\Delta t \in [0.1\text{s}, 0.2\text{s}]$. Our controller is a three-layer MLP with 256 hidden units in each layer. We uniformly sample 50000 points and train for 50 k steps (250 k for navigation). We update the controller via the Adam optimizer [38] with a learning rate 3×10^{-4} for most tasks. Training in PyTorch [39] takes 2-12 hours on a V100 GPU.

Metrics: In testing we evaluate the average STL accuracy (the ratio of the short segments starting at each step satisfies the STL) and the computation time. For RL baselines ($\mathbf{RL_R}, \mathbf{RL_S}, \mathbf{RL_A}$), we also compare the STL accuracy in training.

B. Benchmarks

1) Driving With Traffic Rules: We consider driving near intersections where routes and lateral control are provided. The state $(x, v, I_{\texttt{light}}, \tau, \Delta x, v_{\texttt{lead}}, I_{\texttt{yield}})^T$ is for ego car offset, velocity, light indicator (0 for stop sign and 1 for traffic light), timer (stopped time or traffic light phase), leading vehicle distance, its speed, and yield signal (1 for yield and 0 for not). The (partial) dynamics are: $\dot{x} = v$, $\dot{v} = u$, $(\dot{\Delta x}) = v_{\texttt{lead}} - v$, $\dot{\tau} = (1 - I_{\texttt{light}})\mathbb{1}(\text{at stop sign}) + I_{\texttt{light}}$ where $\mathbb{1}(\text{at stop sign}) = \mathbb{1}(x(x+1) \leq 0)$ as $x = 0$ means being at the intersection, and u is the control. $x, I_{\texttt{light}}, \tau$ will reset at a new intersection, $\Delta x, v_{\texttt{lead}}$ will reset when the leading car changes, and $I_{\texttt{yield}}$ will change when the external yield command is emitted. The rules are: (1) never collide with the leading car, (2) stop by the stop sign for 1 s and then enter the intersection if no yield (3) stop by the intersection if it is red light. Thus, $\Phi = (\neg I_{\texttt{light}} \Rightarrow \phi_1) \wedge (I_{\texttt{light}} \Rightarrow \phi_2) \wedge \phi_3$ where ($T_{\texttt{total}} = T_r + T_g$):

$$\begin{aligned}
\phi_1 &= \Diamond_{[0,T]}(\tau > 1) \wedge \left(I_{\texttt{yield}} \Rightarrow \Box_{[0,T]}(x < 0)\right) \\
\phi_2 &= \Box_{[0,T]}(\tau \% (T_{\texttt{total}}) > T_r \vee x(x - x_{\texttt{inter}}) > 0) \\
\phi_3 &= \Box_{[0,T]}(\Delta x > 0)
\end{aligned} \tag{12}$$

where T_r and T_g are red and green light phase time, $\%$ is modulo operator and $x_{\texttt{inter}}$ is the intersection width. We train π_θ under each sampled scenario (traffic light, stop sign, yield, leading vehicle) at one intersection. In testing, we conduct multi-agent planning with 20 cars and 20 junctions.

2) Reach-N-Avoid Game: An agent navigates to reach goals and avoid obstacles in a 2D maze shown in Fig. 1(b). It can see up to two levels. The agent state is $(x, v, \Delta y, x_0, l_0, g_0, x_1, l_1, g_1)^T$ where x and v denote the agent's horizontal position and velocity, Δy is the vertical distance to the nearest level above. Here x_0 is the obstacle's leftmost horizontal position, l_0 is the obstacle width, and g_0 is the goal location relative to the obstacle (-1 if no goal). x_1, l_1, g_1 are for the second level above. The agent dynamics are: $\dot{x} = v, \dot{v} = a, (\dot{\Delta y}) = c$ where a is the control and c is a constant. Here $\Delta y, \Delta x_0, l_0, g_0, \Delta x_1, l_1, g_1$ will reset once the agent passes a level. The STL is $\Phi = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4$:

$$\begin{aligned}
\phi_1 &= g_0 > 0 \Rightarrow \Diamond_{[0,T]}\left((x - l_0)^2 + \Delta y^2 < r^2\right) \\
\phi_2 &= \Box_{[0,T]}(\Delta y(\Delta y - h) < 0 \Rightarrow \Delta x_0(\Delta x_0 - l_0) > 0) \\
\phi_3 &= g_1 > 0 \Rightarrow \Diamond_{[0,T]}\left((x - l_1)^2 + (\Delta y - d)^2 < r^2\right) \\
\phi_4 &= \Box_{[0,T]}(\Delta y_1(\Delta y_1 - h) < 0 \Rightarrow \Delta x_1(\Delta x_1 - l_1) > 0)
\end{aligned} \tag{13}$$

where r is the goal radius, h is the obstacle's height, $\Delta x_i = x - x_i$, $\Delta y_1 = \Delta y - d$ and d is the gap between two levels. In testing, we control for 500 time steps for evaluation.

3) Ship Collision Avoidance: We control a ship (modeled in [40](Section 4.2)) to avoid obstacles with varied radii. The 12-dim system state has x, y, ψ, u, v, r to describe the pose and x_1, y_1, r_1 (x_2, y_2, r_2) to denote the (second) closest obstacle with radius r_1 (r_2) and relative position x_1, y_1 (x_2, y_2). The controls are thrust T and rudder angle δ. The dynamics are: $\dot{x} = u\cos\psi - v\sin\psi$, $\dot{y} = u\sin\psi + v\cos\psi$, $\dot{\psi} = r$, $\dot{u} = T$, $\dot{v} = 0.01\delta$, $\dot{r} = 0.5\delta$. The rules are: (1) always be in the river (2) always avoid obstacles. The STL is $\Phi = \phi_1 \wedge \phi_2 \wedge \phi_3$:

$$\begin{aligned}\phi_1 &= G_{[0,T]}(|y| < D/2)\\ \phi_2 &= G_{[0,T]}((x-x_1)^2 + (y-y_1)^2 \le r_1^2\\ \phi_3 &= G_{[0,T]}((x-x_2)^2 + (y-y_2)^2 \le r_2^2 \end{aligned} \tag{14}$$

where D is the width of the river. In testing, we roll out 20 trajectories for 200 steps to evaluate the performance.

4) Ship Safe Centerline Tracking: Besides collision avoidance, the ship must also not deviate more than c time units from the centerline. The 10-dim state now has x, y, ψ, u, v, r to denote the ship state, x_1, y_1, r_1 for the closest front obstacle, and τ for the remaining time the ship can deviate, with $\dot{\tau} = -\mathbb{1}(|y| > \gamma)$ where γ is the deviation threshold. x_1, y_1, r_1 and τ will get reset once the ship passes the current obstacle. The STL is: $\Phi = \phi_1 \wedge \phi_2 \wedge \phi_3$ where,

$$\begin{aligned}\phi_1 &= G_{[0,T]}(|y| < D/2)\\ \phi_2 &= G_{[0,T]}((x-x_1)^2 + (y-y_1)^2 \le r_1^2\\ \phi_3 &= (\tau > 0)\, U_{[0,T]}\left(G_{[0,T]}(|y| < \gamma)\right) \end{aligned} \tag{15}$$

We rollout 20 trajectories for 200 steps for evaluation.

5) Robot Navigation: A battery-powered robot navigates to reach the destinations and charging stations. The state $(x, y, x_d, y_d, x_c, y_c, \tau_b, \tau_s)^T$ denotes the robot, the target, the charging station, the battery, and the remaining time at the charging station. The controls are speed v and heading θ. The dynamics are: $\dot{x} = v\cos\theta$, $\dot{y} = v\sin\theta, \tau_b = -1$, $\tau_s = -\mathbb{1}\{\text{station}\}$ where its battery will reset: $\tau_b^+ = T$ once it reaches the charger station and the remaining stay time will get reset $\tau_c^+ = c$ once the robot leaves the charging station. The rules are: (1) always avoid obstacles (2) go to the target if high battery (3) if low battery, go to the charging station and stay for c time units and (4) keep the battery level non-negative. The STL is $\Phi = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5$:

$$\begin{aligned}&\phi_1 = G_{[0,T]}(\neg\text{In(Obstacles)}) \quad \phi_4 = G_{[0,T]}(\tau_b > 0)\\ &\phi_2 = \tau_b > 1 \Rightarrow F_{[0,T]}(\text{Near}(x_d, y_d))\\ &\phi_3 = \tau_b < 1 \Rightarrow F_{[0,T]}(\text{Near}(x_c, y_c))\\ &\phi_5 = \text{Near}(x_c, y_c) \Longrightarrow G_{[0,c]}(\text{Near}(x_c, y_c) \vee \tau_s < 0) \end{aligned} \tag{16}$$

where In(Obstacles) checks if the robot is in any obstacle, and Near(x',y') $= (x-x')^2 + (y-y')^2 \le r^2$. In testing, we constructed a sequence of destinations and five charging stations. After the robot reaches one destination, the next one will show up. The robot can choose any station for charging.

6) Manipulation: A 7DoF Franka Emika robot aims to reach the goal without collisions or breaking the joints (We use PyBullet for visualization). The state $(q_0 \ldots q_6, x, y, z)^T$ denotes the joint angles and the goal location. The dynamics are $\dot{q}_i = u_i, i = 0, \ldots, 6$ where u_i controls the i-th joint. The obstacle is at $(0.3, 0.3, 0.5)$. The STL is $\Phi = \phi_1 \wedge \phi_2 \wedge \ldots \phi_9$:

$$\begin{aligned}\phi_1 &= F_{[0,T]}\left((x_e-x)^2 + (y_e-y)^2 + (z_e-z)^2 < r_g^2\right)\\ \phi_2 &= G_{[0,T]}\left((x_e-0.3)^2 + (y_e-0.3)^2 + (z_e-0.5)^2 > r_o^2\right)\\ \phi_{i+3} &= G_{[0,T]}\left(q_i > q_i^{min} \ \wedge \ q_i < q_i^{max}\right),\ i = 0, \ldots, 6 \end{aligned} \tag{17}$$

where x_e, y_e, z_e is the end effector, q_i^{min}, q_i^{max} are the joint limits and r_g, r_o are the goal / obstacle radii. In evaluation, the arm is asked to reach a sequence of goals in 250 steps.

C. Training and Testing Comparisons

During training, we compare the STL accuracy of our method and the model-free RL baselines ($\mathbf{RL_R}$, $\mathbf{RL_S}$ and $\mathbf{RL_A}$). As shown in Fig. 2, our method in most cases reaches the highest STL accuracy. For tasks with simple dynamics (Traffic, Reach-n-avoid) or simple STL specifications (Ship-safe), the best RL baselines can have similar STL accuracy to ours. However, no one RL baseline can consistently outperform the others. For tasks with moderate system complexity and complicated STL specification (Ship-track, navigation and manipulation), our approach will have $10\% \sim 40\%$ gain in the STL accuracy. We speculate the gain here is because our approach leverages the system dynamics and the gradient information from the STL formula. This shows the advantage of our approach in policy learning compared to RL methods.

In testing, we compare with all baselines and show the result with our backup policy ($\mathbf{Ours_F}$). As shown in Fig. 3, aside from **CEM**, **Ours** can outperform the best baselines by 20% for the ship-track task, 45% for the navigation task and 8% for the manipulation task, and only 3% lower than the best approaches on three tasks with simple dynamics or STL formulas. Compared to **CEM**, **Ours** has a slightly lower accuracy on ship safe control, tracking control, and navigation but a much higher accuracy for the rest three tasks. With the backup policy, $\mathbf{Ours_F}$ achieves the same accuracy as the best RL baselines on Reach-n-avoid and Ship-safe tasks and 1.7% lower on the Traffic task and consistently outperforms **CEM**. The high STL accuracy of **CEM** might be due to the short tasks horizon and low action dimension. The inferior performance for **MPC**, $\mathbf{STL_M}$ **and** $\mathbf{STL_G}$ might be because the solvers encounter numerical issues and cannot converge or the optimizer stucks at local minimum. **MBPO** and **PETS** are worse, which might be because they need careful hyper-parameters tuning and learning the dynamics (for the augmented state space) is hard, especially for high-dimension tasks (e.g., navigation and manipulation). As for the computation time, as shown in Fig. 4, **Ours** is on par with RL and 10X-100X faster than classic MPC or STL solvers. While $\mathbf{Ours_F}$ is slower due to the backup policy, it is still 3X faster than **CEM** and other classical methods.

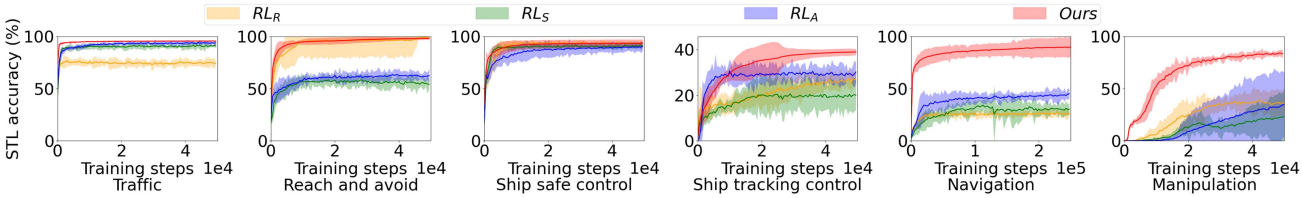


Fig. 2. STL accuracy during training.

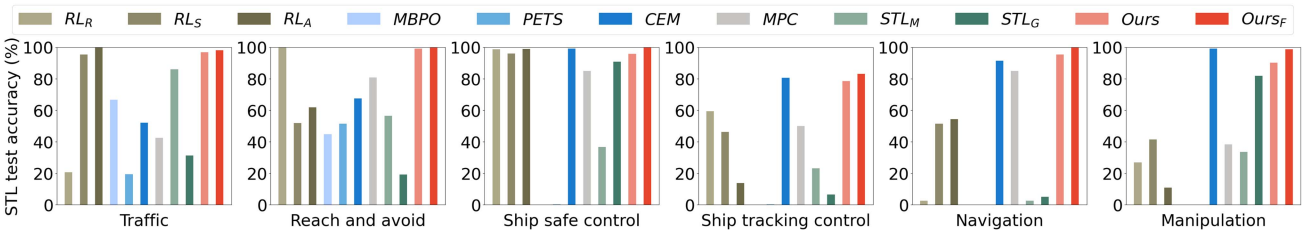


Fig. 3. STL accuracy at test phase.

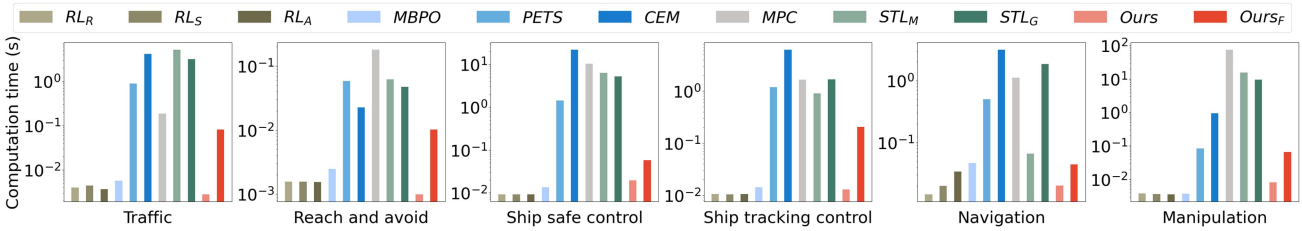
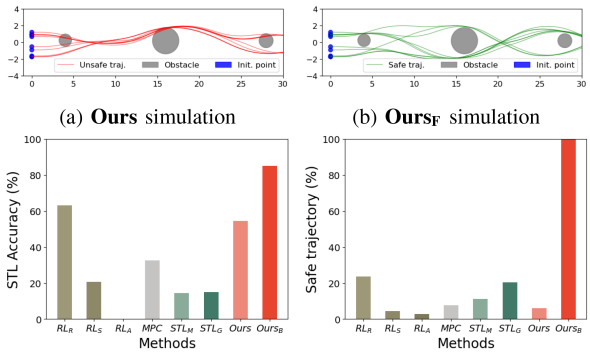


Fig. 4. Computation time at test phase.



(a) **Ours** simulation (b) **Ours$_F$** simulation

(c) STL Accuracy Comparision (d) Safety Rate Comparision

Fig. 5. Backup policy in testing.

D. Testing Backup Policy for Out-of-Distribution Scenarios

When the testing case is out of distribution (OOD), our proposed backup policy can at least maintain the agent's safety and improve the STL accuracy after recovering from the unseen distribution. In the ship-track benchmark, we shift the first obstacle vertically from the centerline and enlarge the second obstacle, which makes the test case OOD (as in training, the obstacles are smaller and on the centerline). From Fig. 5(c) and (d), we can see that without backup policy, **Ours** only achieves 6% safety rate and 54% STL accuracy (from Fig. 5(a) we can see that most of the agents will collide with the first obstacle due to OOD). With the backup policy, **Ours$_F$** achieves 100% safety and increases STL accuracy to 85%. Other baselines are plotted in Fig. 5(c) and (d) for reference.

E. Ablation Studies

Here on the traffic benchmark we show how different parameters and architectures affect the training and the validation accuracy. As shown in Table I, for hyperparameters such as γ, k and network size, a wide range of valid values can achieve similar performances ($0.5 < \gamma < 1.0$, $10 < k < 10000$, NN from 64x3 to 256x3). As γ, k, the NN size and the training samples increase, the train/validation STL accuracy almost monotonously increases initially and saturates eventually. Another interesting finding is that, with only 800 training samples, we can already achieve 83.6% STL test accuracy, which is slightly higher than **STL$_M$** (83%, the best classical baseline on the Traffic benchmark). This shows that our approach has a high learning efficiency.

F. Limitations

First, our gradient-based method might be stuck at a local minimum. In testing, our learned policy cannot always satisfy the STL due to the approximation for the robustness score and the generalization error. Although we propose a backup policy to tackle it, a more robust and time-efficient approach is needed. Besides, we explicitly encode map information into states, which is inefficient for complex and moving obstacles. Representation learning might handle this issue.

VI. Conclusion

We propose a neural network controller learning framework to fulfill STL specifications in robot tasks. Unlike RL methods, our approach learns the policy directly via gradient descent to maximize the approximated robustness score. Experimental results show that our approach achieves the highest STL accuracy compared to other approaches. A backup policy is proposed for STL monitoring process and guarantees the basic safety. In future, we aim to solve more general STL formulas and perception-based controls.

Acknowledgment

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

[1] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3451–3458, Apr. 2022.

[2] C. Dawson and C. Fan, "Robust counterexample-guided optimization for planning from differentiable temporal logic," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 7205–7212.

[3] P. Kapoor, A. Balakrishnan, and J. V. Deshmukh, "Model-based reinforcement learning from signal temporal logic specifications," 2020, *arXiv:2011.04950*.

[4] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, Klosterneuburg, Austria, Springer, 2010, pp. 92–106.

[5] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annu. Symp. Found. Comput. Sci.*, 1977, pp. 46–57.

[6] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.*, vol. 2, no. 4, pp. 255–299, 1990.

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

[8] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI Commun.*, vol. 29, no. 1, pp. 151–162, 2016.

[9] P. Tabuada and G. J. Pappas, "Model checking LTL over controllable linear systems is decidable," in *Proc. Int. Workshop Hybrid Syst.: Comput. Control*, Springer, 2003, pp. 498–513.

[10] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[11] J. McMahon and E. Plaku, "Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 3726–3733.

[12] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE Control Syst. Lett.*, vol. 3, no. 1, pp. 96–101, Jan. 2019.

[13] G. Yang, C. Belta, and R. Tron, "Continuous-time signal temporal logic planning with control barrier functions," in *Proc. Amer. Control Conf.*, 2020, pp. 4612–4618.

[14] Z. Zhang and S. Haesaert, "Modularized control synthesis for complex signal temporal logic specifications," 2023, *arXiv:2303.17086*.

[15] Y. Kantaros and M. M. Zavlanos, "STyLuS*: A. temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 812–836, 2020.

[16] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3840–3847.

[17] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 1002–1028, 2020.

[18] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15537–15543, 2020.

[19] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *Proc. IEEE Conf. Control Technol. Appl.*, 2017, pp. 1235–1240.

[20] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation for parametric STL," in *Proc. IEEE Intell. Veh. Symp.*, 2019, pp. 185–192.

[21] A. Pantazides, D. Aksaray, and D. Gebre-Egziabher, "Satellite mission planning with signal temporal logic specifications," in *Proc. AIAA Scitech Forum*, 2022, Paper 1091.

[22] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. IEEE 55th Conf. Decis. Control*, 2016, pp. 6565–6570.

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

[24] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 3481–3486.

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

[26] M. H. Cohen and C. Belta, "Model-based reinforcement learning for approximate optimal control with temporal logic specifications," in *Proc. 24th Int. Conf. Hybrid Syst.: Comput. Control*, 2021, pp. 1–11.

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

[28] W. Liu, N. Mehdipour, and C. Belta, "Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 91–96, 2021.

[29] W. Hashimoto, K. Hashimoto, and S. Takai, "STL2vec: Signal temporal logic embeddings for control synthesis with recurrent neural networks," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 5246–5253, Apr. 2022.

[30] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for STL," in *Proc. 25th Int. Conf. Comput. Aided Verification*, Saint Petersburg, Russia, Springer, 2013, pp. 264–279.

[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[32] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 12348–12355, 2021.

[33] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 12519–12530.

[34] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 4759–4770.

[35] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Ann. Operations Res.*, vol. 134, pp. 19–67, 2005.

[36] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, pp. 1–36, 2019.

[37] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2021.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[39] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[40] T. I. Fossen, "A survey on nonlinear ship control: From theory to practice," *IFAC Proc. Volumes*, vol. 33, no. 21, pp. 1–16, 2000.