Scaling Learning-based Policy Optimization for Temporal Logic Tasks by Controller Network Dropout

NAVID HASHEMI, University of Southern California, Los Angeles, California, USA
BARDH HOXHA, DANIL PROKHOROV, and GEORGIOS FAINEKOS, Toyota NA R&D, Ann Arbor, Michigan, USA
JYOTIRMOY V. DESHMUKH, University of Southern California, Los Angeles, California, USA

This article introduces a model-based approach for training feedback controllers for an autonomous agent operating in a highly non-linear (albeit deterministic) environment. We desire the trained policy to ensure that the agent satisfies specific task objectives and safety constraints, both expressed in Discrete-Time Signal Temporal Logic (DT-STL). One advantage for reformulation of a task via formal frameworks, like DT-STL, is that it permits quantitative satisfaction semantics. In other words, given a trajectory and a DT-STL formula, we can compute the *robustness*, which can be interpreted as an approximate signed distance between the trajectory and the set of trajectories satisfying the formula. We utilize feedback control, and we assume a feed forward neural network for learning the feedback controller. We show how this learning problem is similar to training recurrent neural networks (RNNs), where the number of recurrent units is proportional to the temporal horizon of the agent's task objectives. This poses a challenge: RNNs are susceptible to vanishing and exploding gradients, and naïve gradient descent-based strategies to solve long-horizon task objectives thus suffer from the same problems. To address this challenge, we introduce a novel gradient approximation algorithm based on the idea of dropout or gradient sampling. One of the main contributions is the notion of *controller network dropout*, where we approximate the NN controller in several timesteps in the task horizon by the control input obtained using the controller in a previous training step. We show that our control synthesis methodology can be quite helpful for stochastic gradient descent to converge with less numerical issues, enabling scalable back-propagation over longer time horizons and trajectories over higher-dimensional state spaces. We demonstrate the efficacy of our approach on various motion planning applications requiring complex spatio-temporal and sequential tasks ranging over thousands of timesteps.

CCS Concepts: • **Computing methodologies** → **Regularization**; • **Theory of computation** → **Modal and temporal logics**; • **Computer systems organization** → **Robotic autonomy**;

Additional Key Words and Phrases: Signal Temporal Logic, Neural Network Control, Feedback Control, Dropout, Gradient Descent

This work was supported by the National Science Foundation through CAREER grants (SHF-2048094, CNS-1932620, CNS-2039087, FMitF-1837131, and CCF-SHF-1932620); funding by Toyota R&D and Siemens Corporate Research through the USC Center for Autonomy and AI, and the Airbus Institute for Engineering Research.
Authors' Contact Information: Navid Hashemi (corresponding author), University of Southern California, Los Angeles, California, USA; e-mail: navidhas@usc.edu; Bardh Hoxha, Toyota NA R&D, Ann Arbor, Michigan, USA; e-mail: bardh.hoxha@toyota.com; Danil Prokhorov, Toyota NA R&D, Ann Arbor, Michigan, USA; e-mail: danil.prokhorov@; toyota.com; Georgios Fainekos, Toyota NA R&D, Ann Arbor, Michigan, USA; e-mail: georgios.fainekos@toyota.com; Jyotirmoy V. Deshmukh, University of Southern California, Los Angeles, California, USA; e-mail: jdeshmuk@usc.edu.

ACM 2378-9638/2024/11-ART42
https://doi.org/10.1145/3696112

ACM Reference format:
Navid Hashemi, Bardh Hoxha, Danil Prokhorov, Georgios Fainekos, and Jyotirmoy V. Deshmukh. 2024. Scaling Learning-based Policy Optimization for Temporal Logic Tasks by Controller Network Dropout. *ACM Trans. Cyber-Phys. Syst.* 8, 4, Article 42 (November 2024), 28 pages.
https://doi.org/10.1145/3696112

1 Introduction

The use of **Neural Networks (NN)** for feedback control enables data-driven control design for highly non-linear environments. The literature about training NN-based controllers or neurocontrollers is plentiful, e.g., see [12, 15, 21, 23, 35, 45]. Techniques to synthesize neural controllers (including deep RL methods) largely focus on optimizing cost functions that are constructed from user-defined state-based rewards or costs. These rewards are often proxies for desirable long-range behavior of the system and can be error-prone [6, 49, 62] and often require careful design [28, 63].

On the other hand, in most engineered safety-critical systems, the desired behavior can be described by a set of spatio-temporal task-objectives, e.g., [11, 20]. For example, consider modeling a mobile robot where the system must reach region R_1 before reaching region R_2, while avoiding an obstacle region. Such spatio-temporal task objectives can be expressed in the mathematically precise and symbolic formalism of **Discrete-Time Variant (DT-STL)** [19] of **Signal Temporal Logic (STL)** [48]. A key advantage of DT-STL is that for any DT-STL specification and a system trajectory, we can efficiently compute the *robustness degree*, i.e., the approximate signed distance of the trajectory from the set of trajectories satisfying/violating the specification [18, 19].

Control design with DT-STL specifications using the robustness degree as an objective function to be optimized is an approach that brings together two separate threads: (1) smooth approximations to the robustness degree of STL specifications [25, 50] enabling the use of STL robustness in gradient-based learning of open-loop control policies, and (2) representation of the robustness as a computation graph allowing its use in training neural controllers using back-propagation [33, 34, 42, 76]. While existing methods have demonstrated some success in training open-loop NN policies [42, 43], and also closed-loop NN policies [33, 34, 76], several key limitations still remain. Consider the problem of planning the trajectory of an Unmanned Aerial Vehicle in a complex, GPS-denied urban environment; here, it is essential that the planned trajectory span several minutes while avoiding obstacles and reaching several sequential goals [52, 68, 71]. However, none of the existing methods to synthesize closed-loop (or even open-loop) policies scale to handle long-horizon tasks.

A key reason for this is the inherent computational challenge in dealing with long-horizon specifications. Training open-loop policies treats the sequence of optimal control actions over the trajectory horizon as decision variables to maximize the robustness of the given STL property. Typical approaches use gradient-descent where in each iteration, the new control actions (i.e., the open-loop policy) are computed using the gradient of the DT-STL property w.r.t. the control actions. If the temporal horizon of the DT-STL property is K, then, this in turn is computed using back-propagation of the DT-STL robustness value through a computation graph representing the composition of the DT-STL robustness computation graph and K copies of the system dynamics. Similarly, if we seek to train closed-loop (neural) feedback control policies using gradient descent, then we can treat the one-step environment dynamics and the neural controller as a recurrent unit that is repeated as many times as the temporal horizon of the DT-STL property. Gradient updates to the neural controller parameters are then done by computing the gradient of the STL computation graph composed with this **Recurrent Neural Network (RNN)**-like structure. In both cases, if the temporal horizon of φ is several hundred steps, then gradient computation requires back-propagation through those many steps. These procedures are quite similar to the ones used for

training an RNN with many recurrent units. It is well-known that back-propagation through RNNs with many recurrent units faces problems of vanishing and exploding gradients [8, 26]. To address these limitations, we propose a sampling-based approximation of the gradient of the objective function (i.e., the STL property), that is particularly effective when dealing with behaviors over larger time horizons. Our key idea is to approximate the gradient during back-propagation by an approximation scheme similar to the idea of dropout layers used in deep NNs [64]. The main idea of dropout layers is to probabilistically set the output of some neurons in the layer to 0 in order to prevent over-fitting. We do a similar trick: In each training iteration we pick some recurrent units to be "frozen," i.e., we use older fixed values of the NN parameters for the frozen layers, effectively approximating the gradient propagation through those layers. We show that this can improve training of NN controllers by at least an *order of magnitude*. Specifically, we reduce training times from hours to minutes, and can also train reactive planners for task objectives that have larger time horizons.

To summarize, we make the following contributions:

(1) We develop a sampling-based approach, inspired by dropout [64], to approximate the gradient of DT-STL robustness w.r.t. the NN controller parameters. Emphasizing the timesteps that contribute the most to the gradient, our method randomly samples time-points over the trajectory. We utilize the structure of the STL formula and the current system trajectory to decide which time-points represent critical information for the gradient.
(2) We develop a back-propagation method that uses a combination of the proposed sampling approach and the smooth version of the robustness degree of a DT-STL specification to train NN controllers.
(3) We demonstrate the efficacy of our approach on higher-dimensional non-linear dynamical systems involving longer-horizon and more complex temporal specifications.

1.1 Organization and Notations

The rest of the article is organized as follows. In Section 2, we introduce the notation and the problem definition. We propose our learning-based control synthesis algorithms in Section 3, present experimental evaluation in Section 5, and conclude in Section 6. We use bold letters to indicate vectors and vector-valued functions, and calligraphic letters to denote sets. A feed forward NN with ℓ hidden layers is denoted by the vector $[n_0, n_1, \cdots n_{\ell+1}]$, where n_0 denotes the number of inputs, $n_{\ell+1}$ is the number of outputs and for all $i \in 1, 2, \cdots, \ell$, n_i denotes the width of ith hidden layer. The notation $x \overset{u}{\sim} \mathcal{X}$ implies the random variable x is sampled uniformly from the set $\mathcal{X}$.

2 Preliminaries

NN Feedback Control Systems (NNFCS). Let $\mathbf{s}$ and $\mathbf{a}$ denote the state and action variables that take values from compact sets $\mathcal{S} \subseteq \mathbb{R}^n$ and $C \subseteq \mathbb{R}^m$, respectively. We use $\mathbf{s}_k$ (respectively, $\mathbf{a}_k$) to denote the value of the state (respectively, action) at time $k \in \mathbb{Z}^{\geq 0}$. We define an NNFCS as a recurrent difference equation

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k, \mathbf{a}_k), \tag{1}$$

where $\mathbf{a}_k = \pi_\theta(\mathbf{s}_k, k)$ is the control policy. We assume that the control policy is a parameterized function π_θ, where θ is a vector of parameters that takes values in Θ. Later in the article, we instantiate the specific parametric form using an NN for the controller. That is, given a fixed vector

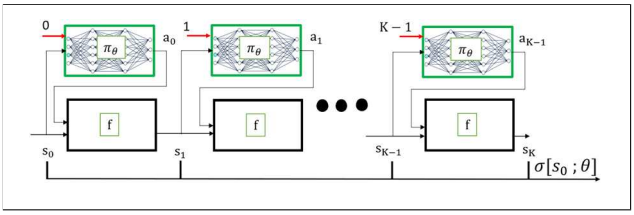


Fig. 1. Shows an illustration of the recurrent structure for the control feedback system.

of parameters θ, the parametric control policy π_θ returns an action $\mathbf{a}_k$ as a function of the current state $\mathbf{s}_k \in \mathcal{S}$ and time k, i.e., $\mathbf{a}_k = \pi_\theta(\mathbf{s}_k, k)$.[1]

Closed-Loop Model Trajectory. For a discrete-time NNFCS as shown in Equation (1), and a set of designated initial states $\mathcal{I} \subseteq \mathcal{S}$, under a pre-defined feedback policy π_θ, Equation (1) represents an autonomous discrete-time dynamical system. For a given initial state $\mathbf{s}_0 \in \mathcal{I}$, a system trajectory $\sigma[\mathbf{s}_0;\theta]$ is a function mapping time instants $k \in 0, 1, \cdots, K$ to $\mathcal{S}$, where $\sigma[\mathbf{s}_0;\theta](k) = \mathbf{s}_k$, and for all $k \in 0, 1, \cdots, K-1$, $\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k, \pi_\theta(\mathbf{s}_k, k))$. In case the dependence to θ is obvious from the context, we utilize the notation $\mathbf{s}_k$ to refer to $\sigma[\mathbf{s}_0;\theta](k)$. Here, K is some integer called the trajectory horizon, and the exact value of K depends on the DT-STL task objective that the closed-loop model trajectories must satisfy. The computation graph for this trajectory is a recurrent structure. Figure 1 shows an illustration of this structure and its similarity to an RNN.

Task Objectives and Safety Constraints. We assume that task objectives and safety constraints are specified using the syntax of DT-STL [19] of STL [48]. We assume that DT-STL formulas are specified in positive normal form, i.e., all negations are pushed to the signal predicates:[2]

$$\varphi \;=\; h(\mathbf{s}) \bowtie 0 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \mid \varphi_1 \mathbf{R}_I \varphi_2, \tag{2}$$

where $\mathbf{U}_I$ and $\mathbf{R}_I$ are the timed until and release operators, $\bowtie \in \{\leq, <, >, \geq\}$, and h is a function from $\mathcal{S}$ to $\mathbb{R}$. In this work, since we use discrete-time semantics for STL (referred to as DT-STL), the time interval I is a bounded interval of integers, i.e., $I = [a, b], a \leq b$. The timed eventually ($\mathbf{F}_I$) and always ($\mathbf{G}_I$) operators can be syntactically defined through until and release. That is, $\mathbf{F}_I\varphi \equiv \top \mathbf{U}_I \varphi$ and $\mathbf{G}_I\varphi \equiv \bot \mathbf{R}_I \varphi$ where $\top$ and $\bot$ represent true and false. The formal semantics of DT-STL over discrete-time trajectories have been previously presented in [19]. We briefly recall them here.

Boolean Semantics and Formula Horizon. We denote the formula φ being true at time k in trajectory $\sigma[\mathbf{s}_0;\theta]$ by $\sigma[\mathbf{s}_0;\theta], k \models \varphi$. We say that $\sigma[\mathbf{s}_0;\theta], k \models h(\mathbf{s}) \bowtie 0$ iff $h(\sigma[\mathbf{s}_0;\theta](k)) \bowtie 0$. The semantics of the Boolean operations ($\wedge$, $\vee$) follow standard logical semantics of conjunctions and disjunctions, respectively. For temporal operators, we say $\sigma[\mathbf{s}_0;\theta], k \models \varphi_1 \mathbf{U}_I \varphi_2$ is true if there is a time k', s.t. $k' - k \in I$ where φ_2 is true and for all times $k'' \in [k, k')$, φ_1 is true. Similarly, $\sigma[\mathbf{s}_0;\theta], k \models \varphi_1 \mathbf{R}_I \varphi_2$ is true if for all times k' with $k' - k \in I$, φ_2 is true, or there exists some time $k'' \in [k, k')$ such that φ_1 was true. The temporal scope or horizon of a DT-STL formula defines the last timestep required to evaluate the formula, $\sigma[\mathbf{s}_0;\theta], 0 \models \varphi$ (see [48]). For example, the temporal scope of the formula $\mathbf{F}_{[0,3]}(x > 0)$ is 3, and that of the formula $\mathbf{F}_{[0,3]}\mathbf{G}_{[0,9]}(x > 0)$ is $3 + 9 = 12$. We also set the horizon of trajectory equivalent to the horizon of formula, as we plan to monitor the satisfaction of the formula by the trajectory.

Quantitative Semantics (Robustness Value) of DT-STL. Quantitative semantics of DT-STL roughly define a signed distance of a given trajectory from the set of trajectories satisfying or violating

[1] Our proposed feedback policy explicitly uses time as an input. This approach is motivated by the need to satisfy temporal tasks, which requires time awareness for better decision-making.

[2] Any formula in DT-STL can be converted to a formula in positive normal form using DeMorgan's laws and the duality between the Until and Release operators.

Table 1. Quantitative Semantics of STL

φ	$\rho(\varphi, k)$	φ	$\rho(\varphi, k)$
$h(\mathbf{s}_k) \geq 0$	$h(\mathbf{s}_k)$	$\mathbf{F}_{[a,b]}\psi$	$\max_{k' \in [k+a, k+b]} \rho(\psi, k')$
$\varphi_1 \wedge \varphi_2$	$\min(\rho(\varphi_1, k), \rho(\varphi_2, k))$	$\varphi_1 \mathbf{U}_{[a,b]} \varphi_2$	$\max_{k' \in [k+a, k+b]} \left(\min \left(\rho(\varphi_2, k'), \min_{k'' \in [k, k')} \rho(\varphi_1, k'') \right) \right)$
$\varphi_1 \vee \varphi_2$	$\max(\rho(\varphi_1, k), \rho(\varphi_2, k))$	$\varphi_1 \mathbf{R}_{[a,b]} \varphi_2$	$\min_{k' \in [k+a, k+b]} \left(\max \left(\rho(\varphi_2, k'), \max_{k'' \in [k, k')} \rho(\varphi_1, k'') \right) \right)$
$\mathbf{G}_{[a,b]}\psi$	$\min_{k' \in [k+a, k+b]} \rho(\psi, k')$		

the given DT-STL formula. There are many alternative semantics proposed in the literature [2, 18, 19, 58]; in this article, we focus on the semantics from [18] that are shown in Table 1. The robustness value $\rho(\varphi, \sigma[\mathbf{s}_0 ; \theta], k)$ of a DT-STL formula φ over a trajectory $\sigma[\mathbf{s}_0 ; \theta]$ at time k is defined recursively as reported in Table 1.[3] We note that if $\rho(\varphi, k) > 0$ the DT-STL formula φ is satisfied at time k, and we say that the formula φ is satisfied by a trajectory if $\rho(\varphi, 0) > 0$.

Prior Smooth Quantitative Semantics for DT-STL. To address non-differentiability of the robust semantics of STL, there have been a few alternate definitions of smooth approximations of the robustness in the literature. The initial proposal for this improvement is provided by [50]. Later the authors in [25] proposed another smooth semantics which in addition is a guaranteed lower bound for the robustness value that can be even more advantageous computationally. We denote the smooth robustness of trajectory $\sigma[\mathbf{s}_0 ; \theta]$ for temporal specification φ, with $\tilde{\rho}(\varphi, \sigma[\mathbf{s}_0 ; \theta], 0)$.

Neurosymbolic Smooth Semantics. The prior smooth semantics for gradient computation over DT-STL [25, 42, 50] perform backward computation on a computation graph that is generated based on dynamic programming. Although these computation graphs are efficient for forward computation, they may face computational difficulty for backward computation over the robustness when the specification is highly complex or its task horizon is noticeably long. Unlike the previous computation graphs that are based on dynamic programming, the neurosymbolic computation graph STL2NN [33], directly utilizes the STL tree [18] to generate a feed forward ReLU NN, whose depth grows logarithmically with the complexity of specification. This makes back-propagation more feasible for complex specifications. Specifically, the way it formulates the robustness (feed forward NN) facilitates the back and forward propagation process, by enabling vectorized computation. However, since STL2NN is exactly identical to the non-smooth robustness introduced in Table 1, we proposed in [32] a smooth under-approximation for STL2NN replacing the ReLU activation function with swish() and softplus(), and introduced this as LB4TL, and here we utilize this smooth semantics.

Problem Definition. In this article, we provide model-based algorithms to learn a policy $\pi_{\theta^\star}$ that maximizes the degree to which certain task objectives and safety constraints are satisfied. In particular, we wish to learn an NN control policy π_θ (or equivalently the parameter values θ), s.t. for any initial state $\mathbf{s}_0 \in \mathcal{I}$, using the control policy π_θ, the trajectory obtained, i.e., $\sigma[\mathbf{s}_0 ; \theta]$ satisfies a given DT-STL formula φ. In other words, our ultimate goal is to solve the optimization problem shown in Equation (3). For brevity, we use $F(\mathbf{s}_k, k ; \theta)$ to denote $\mathbf{f}(\mathbf{s}_k, \pi_\theta(\mathbf{s}_k, k))$:

$$\theta^* = \arg\max_\theta \left(\min_{\mathbf{s}_0 \in \mathcal{I}} [\rho(\varphi, \sigma[\mathbf{s}_0 ; \theta], 0)] \right), \quad \textbf{s.t.} \ \forall (k \in \mathbb{Z} \wedge 1 \leq k < K) : \mathbf{s}_{k+1} = F(\mathbf{s}_k, k ; \theta). \tag{3}$$

However, ensuring that the robustness value is positive for all $\mathbf{s}_0 \in \mathcal{I}$ is computationally challenging. Therefore, we relax the problem to maximizing the min value of the robustness only over

[3] For brevity, we omit the trajectory from the notation, as it is obvious from the context.

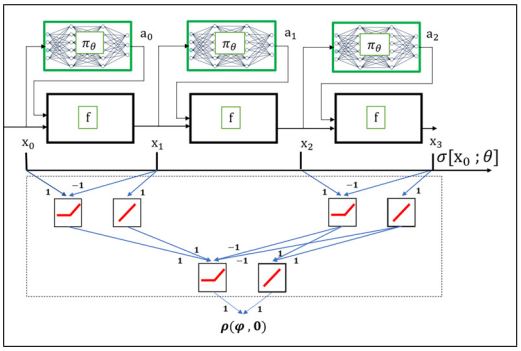


Fig. 2. This figure shows the symbolic trajectory generated by NN feedback controller, and the computation graph for DT-STL robustness. The DT-STL robustness is presented as a neurosymbolic computation graph [33] via ReLU and linear activation functions.

a set of states $\hat{\mathcal{I}}$ sampled from the initial states $\mathcal{I}$, i.e., $\theta^* \approx \arg\max_\theta \left(\min_{\mathbf{s}_0 \in \widehat{\mathcal{I}}} [\rho(\varphi, \sigma[\mathbf{s}_0\,;\theta], 0)]\right)$. We solve this problem using algorithms based on stochastic gradient descent followed by statistical verification to obtain high-confidence control policies.

3 Training NN Control Policies

Our solution strategy is to treat each timestep of the given dynamical equation in Equation (1) as a recurrent unit. We then sequentially compose or unroll as many units as required by the horizon of the DT-STL specification.

Example 1. Assume a one-step dynamics with scalar state, $\mathbf{x} \in \mathbb{R}$ and scalar feedback control policy $\mathrm{a}_k = \pi_\theta(\mathbf{x}_k)$ as, $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \pi_\theta(\mathbf{x}_k))$. If the specification is $\mathbf{F}_{[0,3]}(\mathbf{x} > 0)$, then, we use three instances of $\mathbf{f}(\mathbf{x}_k, \pi_\theta(\mathbf{x}_k))$ by setting the output of the kth unit to be the input of the $(k+1)$th unit. This unrolled structure implicitly contains the system trajectory, $\sigma[\mathbf{x}_0, \ ;\theta]$ starting from some initial state $\mathbf{x}_0$ of the system. The unrolled structure essentially represents the symbolic trajectory, where each recurrent unit shares the NN parameters of the controller (see Figure 2 for more detail). By composing this structure with the robustness semantics representing the given DT-STL specification φ, we have a computation graph that maps the initial state of the system in Equation (1) to the robustness degree of φ. Thus, training the parameters of this resulting structure to guarantee that its output is positive (for all initial states) guarantees that each system trajectory satisfies φ.

However, we face a challenge in training the NN controller that is embodied in this structure.

Challenge: Since our computation graph resembles a recurrent structure with repeated units proportional to the formula's horizon, naïve gradient-based training algorithms struggle with gradient computation when using back-propagation through the unrolled system dynamics. In other word, the gradient computation faces the same issues of vanishing and exploding gradients when dealing with long trajectories.

Controller Synthesis as an Optimization Problem. In order to train the controller, we solve the following problem:

$$\theta^* = \arg\max_\theta \left(\min_{\mathbf{s}_0 \in \widehat{\mathcal{I}}} [\rho(\varphi, \sigma[\mathbf{s}_0\,;\theta], 0)]\right), \qquad \textbf{s.t. } \sigma[\mathbf{s}_0;\theta](k+1) = F(\mathbf{s}_k, k\,;\theta). \tag{4}$$

We thus wish to maximize the expected value of the robustness for trajectories starting in states uniformly sampled from the set of initial states. An approximate solution for this optimization problem is to train the NN controller using a vanilla back-propagation algorithm to compute the gradient of the objective function for a subset of randomly sampled initial states $\widehat{\mathcal{I}} \subset \mathcal{I}$, and updates the parameters of the NN controller using this gradient.

Remark 3.1. A training-based solution to the optimization problem does not guarantee that the specification is satisfied *for all* initial states $\mathbf{s}_0 \in \mathcal{I}$. To tackle this, we can use a methodology like [33] that uses reachability analysis to verify the synthesized controller. However, given the long time horizon, this method may face computational challenges. An alternative approach is to eschew deterministic guarantees, and instead obtain probabilistic guarantees (see Section 5.6).

4 Extension to Long Horizon Temporal Tasks and Higher Dimensional Systems

In this section, we introduce an approach to alleviate the problem of exploding/vanishing gradients outlined in the previous section. Our solution approach is inspired by the idea of using dropout layers [64] in training deep NNs. In our approach, we propose a sampling-based technique, where we only select certain time-points in the trajectory for gradient computation, while using a fixed older control policy at the non-selected points. Our approach to gradient sampling can be also viewed through the lens of stochastic depth, as suggested by [36], which involves sampling layers followed by identity transformations provided in ResNet. However, our methodology differs as we employ a distinct approach that is better suited for control synthesis within the STL framework. Before starting our main discussion on this topic, we first provide an overview of this section:

—In Section 4.1, we introduce the notion of gradient approximation through sampling the trace, and justify why it is a suitable replacement for the original gradient, in case the original gradient is not accessible (e.g., long-horizon tasks).
—In Section 4.2, we put forward the notion of critical time which states that the robustness of DT-STL is only related to a specific timestep. We then propose the idea of including this timestep into our gradient approximation technique.
—In Section 4.3, we bring up the point that gradient approximation using the critical time may, in some cases, result in failure for training. In these cases, we suggest approximating the DT-STL robustness as a function of all the trace, that is the smooth version of the robustness semantics.
—In Section 4.4, we explain how to approximate the gradient for both of the scenarios we proposed above (e.g., critical time and smooth semantics). We also introduce Algorithm 1 which concludes Section 4.

4.1 Sampling-based Gradient Approximation Technique

We propose to sample random timesteps in the recurrent structure shown in Figure 1 and at the selected timestep, we do an operation that is similar to dropping the entire neural controller. However, approximating the gradient by dropping out the controller at several timesteps may result in inaccurate approximation. We compensate for this by repeating our modified dropout process and computing cumulative gradients. Restriction of dropout to sample timesteps results in less number of self-multiplication of weights and therefore alleviates the problem of vanishing/exploding gradient. To ensure that the trajectory is well-defined, when we drop out the controller unit at a selected timestep, we replace it with a constant function that returns the evaluation of the controller unit (at that specific timestep) in the forward pass. We formalize this using the notion of a sampled trajectory in Definition 4.1.

Definition 4.1 (Sub-trajectory and Sampled trajectory). Consider the set of N different sampled timesteps $\mathcal{T} = \{t_0 = 0, t_1, t_2, \cdots, t_N\}$ sampled from the horizon $\mathcal{K} = \{0, 1, 2, \cdots, K\}$, and also the initial state $\mathbf{s}_0$, and the control parameters $\theta^{(j)}$ in the gradient step j. The sub-trajectory, $\mathsf{sub}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big) = \mathbf{s}_0, \mathbf{s}_{t_1}, \mathbf{s}_{t_2}, \cdots, \mathbf{s}_{t_N}$ is simply a selection of N states from $\sigma[\mathbf{s}_0\,;\theta^{(j)}]$ with timesteps $t_i \in \mathcal{T}$. In other words, for all $i \in \{0, 1, \cdots, N\}$: $\mathsf{sub}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)\,(i) = \sigma[\mathbf{s}_0\,;\theta^{(j)}](t_i)$. Now, consider the sub-trajectory $\mathsf{sub}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)$, and a sequence of actions $\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_{K-1}$ resulting from $\mathbf{s}_0$ and $\theta^{(j)}$. For any $t_i \in \mathcal{T}$, we drop out the NN controllers on timesteps $t_i + 1, t_i + 2, \cdots, t_{i+1} - 1$ and replace them with the actions $\mathbf{a}_{1+t_i}, \mathbf{a}_{2+t_i}, \cdots \mathbf{a}_{t_{i+1}-1}$. This provides a variant of sub-trajectory called *sampled trajectory*, and we denote it by $\mathsf{smpl}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)$. In other words, for any timestep $t_i \in \mathcal{T}$, assuming the function $\mathbf{f}_{i+1} : \mathcal{S} \times \Theta \to \mathcal{S}$ (for brevity, henceforth, we denote $\mathbf{f}_{i+1}(\mathbf{s}\,;\theta^{(j)})$ by $\mathbf{f}^{(j)}_{i+1}(\mathbf{s})$):

$$\mathbf{f}^{(j)}_{i+1}(\mathbf{s}) = \mathbf{f}(\mathbf{f}(\cdots \mathbf{f}(\quad F(\mathbf{s}, t_i\,;\theta^{(j)}),\ \mathbf{a}_{1+t_i}\ ), \mathbf{a}_{2+t_i}), \cdots), \mathbf{a}_{t_{i+1}-2}),\ \mathbf{a}_{t_{i+1}-1}),$$

we have $\mathsf{smpl}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)\,(0) = \mathbf{s}_0$, and for all $i \in \{0, 1, \cdots, N-1\}$, we have,

$$\mathsf{smpl}\Big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\Big)\,(i+1) = \mathbf{f}^{(j)}_{i+1}\left(\mathsf{smpl}\Big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\Big)\,(i)\right).$$

Remark 4.2. The sub-trajectory $\mathsf{sub}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)$ with parameters $\theta^{(j)}$ can also be recursively defined as

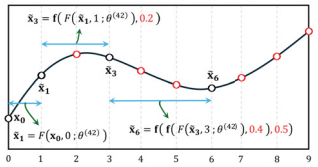
$$\begin{aligned}&\mathsf{sub}\Big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\Big)\,(i+1)\\ &\quad= F\left(\cdots\left(\quad F\left(F\left(\mathsf{sub}\Big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\Big)\,(i), t_i\ \theta^{(j)}\ \right), t_i + 1\,;\theta^{(j)}\right)\cdots\right), t_{i+1} - 1\,;\theta^{(j)}\right).\end{aligned}$$

Notice that the parameters $\theta^{(j)}$ are referenced multiple times while in $\mathsf{smpl}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)$ only once.

Figure 5 presents Definition 4.1 through visualization. This definition replaces the set of selected nodes—on a randomly selected timestep—with its pre-computed evaluation. This set of nodes are indeed a controller unit on the timesteps sampled to apply dropout.[4] Excluding the timesteps with fixed actions, we then name the set of states on the remaining timesteps—as the *sampled trajectory*, and we denote it as $\mathsf{smpl}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}\big)$.

Example 2. Let the state and action at the time k be $\mathbf{x}_k \in \mathbb{R}$ and $\mathrm{a}_k \in \mathbb{R}$, respectively. The feedback controller is $\mathrm{a}_k = \pi_\theta(\mathbf{x}_k, k), \theta \in \mathbb{R}^3$ and the dynamics is also $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathrm{a}_k)$, $\mathbf{x}_0 = 1.15$. Let's also assume a trajectory of horizon 9 over time-domain (i.e., $\mathcal{K} = \{i \mid 0 \le i \le 9\}$) with a trajectory $\sigma[\mathbf{x}_0\,;\theta] = \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9$. Suppose, we are in the gradient step $j = 42$, and in this iteration, we want to generate a sampled trajectory with $N = 3$ timesteps, where, $\mathcal{T} = \{0, t_1 = 1, t_2 = 3, t_3 = 6\}$. The control parameters at this gradient step are also $\theta^{(42)} = [1.2, 2.31, -0.92]$ that results in the control sequence $\mathbf{a} = 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8$. Given this information, we define the sampled trajectory as the sequence $\mathsf{smpl}\big(\sigma[\mathbf{x}_0\,;\theta^{(42)}], \mathcal{T}\big) = \mathbf{x}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_3, \tilde{\mathbf{x}}_6$, where,

$$\begin{aligned}&\tilde{\mathbf{x}}_1 = \mathbf{f}^{(42)}_1(\mathbf{x}_0) = F(\mathbf{x}_0, 0\,;\theta^{(42)}),\\ &\tilde{\mathbf{x}}_3 = \mathbf{f}^{(42)}_2(\tilde{\mathbf{x}}_1) = \mathbf{f}(\ F(\tilde{\mathbf{x}}_1, 1\,;\theta^{(42)}),\ 0.2),\\ &\tilde{\mathbf{x}}_6 = \mathbf{f}^{(42)}_3(\tilde{\mathbf{x}}_3) = \mathbf{f}(\ \mathbf{f}(\ F(\tilde{\mathbf{x}}_3, 3\,;\theta^{(42)}),\ 0.4),\ 0.5).\end{aligned} \quad \Longrightarrow$$



[4]The set of sampled timesteps for dropout is in fact the set-difference between $\mathcal{K}$ and $\mathcal{T}$, where $\mathcal{T}$ is the set of sampled times steps that is generated to define the sampled trajectory.

Algorithm 1: Gradient Sampling and Training the Controller for Long Horizon Tasks

Input: $\epsilon,\ M,\ N,\ N_1,\ N_2,\ \theta^{(0)},\ \varphi,\ \bar{\rho},\ \widehat{I},\ j=0$
while $\rho^{\varphi}(\sigma[\mathbf{s}_0;\theta^{(j)}]) \leq \bar{\rho}$ **do**
  $\mathbf{s}_0 \leftarrow$ Sample from $\widehat{I}$ use_smooth $\leftarrow$ False $j \leftarrow j+1$
  if use_smooth = False **then**
    $\theta_1,\ \theta_2 \leftarrow \theta^{(j)}$
    // θ_1&θ_2 are candidates for parameter update via critical predicate and waypoint.
    // The following loop updates θ_1 and θ_2 via cumulation of N_1 sampled gradients
    for $i \leftarrow 1,\cdots,N_1$ **do**
      $\sigma[\mathbf{s}_0;\theta_1],\ \sigma[\mathbf{s}_0;\theta_2] \leftarrow$ Simulate the trajectory via θ_1, θ_2, and $\mathbf{s}_0$
      $k^*, h^*(\mathbf{s}_{k^*}) \leftarrow$ obtain the critical time and the critical predicate
      $\mathcal{T}^1,\ \mathrm{smpl}(\sigma[\mathbf{s}_0;\theta_1],\mathcal{T}^1) \leftarrow$
        sample set of time steps $\mathcal{T}^1=\{0,t_1,..,t_N=k^*\}$ and its sampled trajectory
      $\mathcal{T}^2,\ \mathrm{smpl}(\sigma[\mathbf{s}_0;\theta_2],\mathcal{T}^2) \leftarrow$
        sample set of time steps $\mathcal{T}^2=\{0,t_1,..,t_N\}$ and its sampled trajectory
      $\mathcal{J} \leftarrow h^*\left(\mathrm{smpl}(\sigma[\mathbf{s}_0;\theta_1],\mathcal{T}^1)(N)\right)$ $d_1 \leftarrow [\partial\mathcal{J}/\partial\theta]_{\text{sampled}}$ $\theta_1 \leftarrow \theta_1 + \mathrm{Adam}(d_1/N_1)$
      $\mathcal{J} \leftarrow \mathcal{J}^{\mathrm{wp}}\left(\mathrm{smpl}(\sigma[\mathbf{s}_0;\theta_2],\mathcal{T}^2)\right)$ $d_2 \leftarrow [\partial\mathcal{J}/\partial\theta]_{\text{sampled}}$ $\theta_2 \leftarrow \theta_2 + \mathrm{Adam}(d_2/N_1)$
    // Update the control parameter with θ_2 if it increases the robustness value
    // Otherwise update the control parameter with θ_1 if it increases the robustness value
    // Otherwise, check for non-differentiable local maximum
    if $\rho^{\varphi}(\sigma[\mathbf{s}_0;\theta_2]) \geq \rho^{\varphi}(\sigma[\mathbf{s}_0;\theta^{(j)}])$ **then** $\theta^{(j+1)} \leftarrow \theta_2$
    else if $\rho^{\varphi}(\sigma[\mathbf{s}_0;\theta_1]) \geq \rho^{\varphi}(\sigma[\mathbf{s}_0;\theta^{(j)}])$ **then** $\theta^{(j+1)} \leftarrow \theta_1$
    else
      $\ell \leftarrow 1$ update $\leftarrow$ True
      while update & (use_smooth=False) **do**
        $\ell \leftarrow \ell/2$ $\hat{\theta} \leftarrow \theta^{(j)} + \ell(\theta_1 - \theta^{(j)})$
          // Keep the gradient direction & reduce the learning rate
        // Update the control parameter with $\hat{\theta}$ if it increases the robustness value
        if $\rho(\varphi,\sigma[\mathbf{s}_0;\hat{\theta}],0) \geq \rho^{\varphi}(\sigma[\mathbf{s}_0;\theta^{(j)}])$ **then** $\left[\theta^{(j+1)} \leftarrow \hat{\theta}\right.$ $\left.\text{update} \leftarrow \text{False}\right]$
        else if $\ell < \epsilon$ **then**
          use_smooth $\leftarrow$ True // swap the objective with $\tilde{\rho}$ if $\ell < \epsilon$
  if use_smooth = True **then**
    $\theta_3 \leftarrow \theta^{(j)}$ // θ_3 is the candidate for parameter update via smooth semantic $\tilde{\rho}$
    // The following loop updates θ_3 via cumulation of N_2 sampled gradients
    for $i \leftarrow 1,\cdots,N_2$ **do**
      $\mathcal{T}^q, \mathrm{smpl}(\sigma[\mathbf{s}_0;\theta_3],\mathcal{T}^q), q \in 1,\cdots,M \leftarrow$
        Make M sets of sampled time steps from Equation (5) & their sampled trajectories
      $\mathcal{J} \leftarrow \tilde{\rho}$ $d_3 \leftarrow [\partial\mathcal{J}/\partial\theta]_{\text{sampled}}$ $\theta_3 \leftarrow \theta_3 + \mathrm{Adam}(d_3/N_2)$
    $\theta^{(j+1)} \leftarrow \theta_3$

where the constants $0.2, 0.4, 0.5$ are the 3rd, 5th, and 6th elements in the pre-evaluated control sequence $\mathbf{a}$, respectively.

4.2 Including the Critical Predicate in Time Sampling

While it is possible to select random time-points to use in the gradient computation, in our preliminary results, exploiting the structure of the given DT-STL formula—specifically identifying and

using *critical predicates* [1]—gives better results. Proposition 3.1 in [1] introduces the notion of critical predicate. Here, we also provide this definition as follows:

Definition 4.3 (Critical Predicate). As the robustness degree of DT-STL is an expression consisting of min and max of robustness values of predicates at different times, the robustness degree is consistently equivalent to the robustness of one of the predicates $h(\cdot)$ at a specific time. This specific predicate $h^* > 0$ is called the critical predicate, and this specific time k^* is called the critical time.

Example 3. We again consider Example 1 to clarify the notion of critical predicate. In this example, we have four predicates of a unique type, e.g., $h(\mathbf{x}_k) = \mathbf{x}_k > 0$. Thus, the robustness values of the predicate $h(\mathbf{x}) > 0$ at time-points 0, 1, 2, 3 are, respectively, $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. Assume the trajectory is $\sigma[\mathbf{x}_0\,; \theta] = [\mathbf{x}_0 = 1,\ \mathbf{x}_1 = 2,\ \mathbf{x}_2 = 3,\ \mathbf{x}_3 = 1.5]$. Since the robustness function is defined as $\rho(\varphi, 0) = \max\left(h(\mathbf{x}_0),\ h(\mathbf{x}_1),\ h(\mathbf{x}_2),\ h(\mathbf{x}_3)\right)$, the robustness value is equivalent to $h(\mathbf{x}_2)$. Thus, we can conclude, the critical predicate is $h^* = h(\mathbf{x}_2) > 0$ and the critical time is $k^* = 2$.

The critical predicate and critical time of a DT-STL formula can be computed using the same algorithm used to compute the robustness value for a given DT-STL formula. This algorithm is implemented in the S-Taliro tool [7].

4.3 Safe Re-Smoothing

A difficulty in using critical predicates is that a change in controller parameter values may change the system trajectory, which may in turn change the predicate that is critical in the robustness computation. Specifically, if the critical predicate in one gradient step is different from the critical predicate in the subsequent gradient step, our gradient ascent strategy may fail to improve the robustness value, as the generated gradient in this gradient step is local.

Example 4. To clarify this with an example, we present a specific scenario in Figure 3. This figure shows the robustness value as a non-differentiable function of control parameters, that is a piece-wise differentiable relation where every differentiable segment represents a specific critical predicate. The system dynamics is $\mathbf{x}_{k+1} = 0.8\mathbf{x}_k^{1.2} - \mathbf{e}^{-4u_k \sin(u_k)^2}$, where the system starts from $\mathbf{x}_0 = 1.15$ and the controller is $u_k = \tanh(\theta \mathbf{x}_k)$. The robustness is plotted based on control parameter $-1 \le \theta \le 1$ and is corresponding to the formula $\Phi = \mathbf{F}_{[0,45]}\left(\mathbf{G}_{[0,5]}\left(\mathbf{x} > 0\right)\right) \wedge \mathbf{G}_{[0,50]}\left(1 - 10\mathbf{x} > 0\right)$. Assume the training process is in the 15th gradient step of back-propagation with $\theta = \theta^{(15)} = 0.49698$ where the critical predicate for this control parameter is denoted by $p_1 := (\mathbf{x}_1 > 0)$. The gradient generated from the critical predicate p_1 suggests increasing the value of θ, which should result in $\theta = \theta^{(16)} = 0.50672$. However, applying the gradient would move the parameter value to a region of parameter space where the critical predicate is $p_2 := (1 - 10\mathbf{x}_{45} > 0)$. In this case, the gradient generated from the critical predicate p_1 is local to this gradient step, as the critical predicate shifts from p_1 to p_2. Our approach in this scenario is to first reduce the learning rate. If this does not lead to an increase in the robustness value, we then transition to smooth semantics, which takes all predicates into account. The scenario proposed in this figure shows this local gradient may result in a drastic drop in the robustness value from 8.09 to −6.15. Therefore, the gradient of critical predicate is useful, only if the gradient step preserves the critical predicate.

Given a predefined specification φ, a fixed initial state, differentiable controller with parameter θ, and a differentiable model, the robustness value is a piece-wise differentiable function of control parameter, where each differentiable segment represents a unique critical predicate (see Figure 4). However, the Adam algorithm[5] assumes a differentiable objective function. Therefore, we utilize

[5] In this article, we utilize MATLAB's `adamupdate()` library, https://www.mathworks.com/help/deeplearning/ref/adamupdate.html

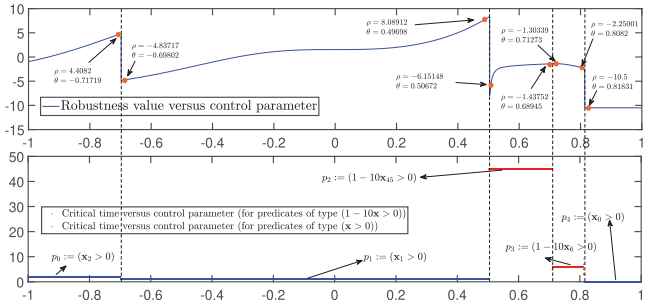


Fig. 3. This figure shows a common challenge in using critical predicate for control synthesis. This figure presents the robustness as a piece-wise differentiable function of control parameter θ (with resolution, 0.00001), where each differentiable segment represents a distinct critical predicate.

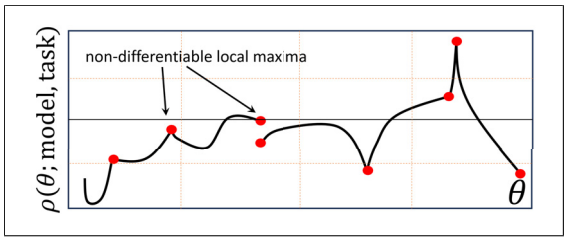


Fig. 4. This figure shows an example for the relation between control parameters and the resulting robustness as a piece-wise differentiable function. Assuming a fixed initial state, every control parameter is corresponding to a simulated trajectory, and that trajectory represents a robustness value. This robustness value is equal to the quantitative semantics for the critical predicate. Within each differentiable segment in this plot, the control parameters yield trajectories associated with a unique critical predicate.

the critical predicate as the objective function when we are in the differentiable segments, and we replace it with the smooth semantics of DT-STL robustness, $\tilde{\rho}$, at the non-differentiable local maxima where the critical predicate is updated. We refer to this shift between critical predicate and smooth semantics as safe re-smoothing. However, it is practically impossible to accurately detect the non-differentiable local maxima; thus we take a more conservative approach and we instead, utilize $\tilde{\rho}$ at every gradient step when the critical predicate technique is unable to improve the robustness.

4.4 Computing the Sampled Gradient

We now explain how we compute an approximation of the gradient of original trajectory (that we call the *original gradient*). We call the approximate gradient from our sampling technique as the *sampled gradient*. In the back-propagation algorithm—at a given gradient step j and with control parameter $\theta^{(j)}$—we wish to compute the sampled gradient $[\partial\mathcal{J}/\partial\theta^{(j)}]_{\text{sampled}}$. The objective function $\mathcal{J}$ in our training algorithm can be either the robustness for critical predicate or the smooth semantics for the robustness of trajectory, $\tilde{\rho}$. The former is defined over a single trajectory state (i.e., at critical time) while the latter is defined over the entire trajectory. In response, we propose two different approaches for trajectory sampling for each objective function:

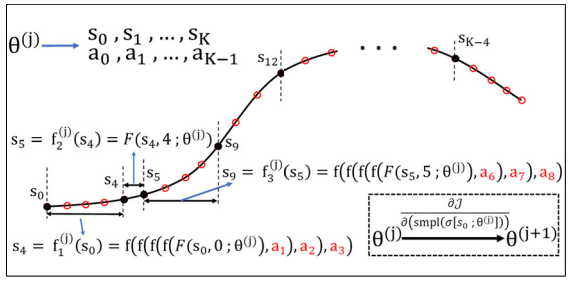


Fig. 5. This figure depicts the sampling-based gradient computation. In our approach, we freeze the controller at some time-points, while at others we assume the controller to be a function of its parameters that can vary in this iteration of back-propagation process. The actions that are fixed are highlighted in red, whereas the dependent actions are denoted in black. The red circles represent the timesteps where the controller is frozen.

(1) In case the objective function $\mathcal{J}$ is the robustness for critical predicate, it is only a function of the trajectory state $\mathbf{s}_{k^*}$. Thus, we sample the timesteps as, $\mathcal{T} = \{0, t_1, t_2, \cdots, t_N\},\ t_N = k^*$ to generate a sampled trajectory $\mathsf{smpl}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}\big)$ that ends in critical time. We utilize this sampled trajectory to compute the sampled gradient. The original gradient regarding the critical predicate can be formulated as $\partial\mathcal{J}/\partial\theta = (\partial\mathcal{J}/\partial\mathbf{s}_{k^*})\,(\partial\mathbf{s}_{k^*}/\partial\theta)$, where $\mathbf{s}_{k^*} = \mathsf{sub}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}\big)\,(N)$. However, we define $\mathcal{J}$ on our sampled trajectory and propose the sampled gradient as

$$\left[\frac{\partial\mathcal{J}}{\partial\theta}\right]_{\text{sampled}} = \left(\frac{\partial\mathcal{J}}{\partial\mathsf{smpl}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}\big)\,(N)}\right)\left(\frac{\partial\mathsf{smpl}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}\big)\,(N)}{\partial\theta}\right).$$

(2) In case the objective function is the smooth semantics for the robustness $\tilde{\rho}$, it is a function of all the trajectory states. In this case, we consequently segment the trajectory into M subsets, by random time sampling as $\mathcal{T}^q = \big\{0, t_1^q, t_2^q, \cdots, t_N^q\big\} \subseteq \mathcal{K},\ q \in \{1, \cdots, M\}$ (see Example 5), where

$$(\forall q, q' \in \{1, \cdots, M\} : \mathcal{T}^q \cap \mathcal{T}^{q'} = \{0\}) \wedge (\mathcal{K} = \bigcup_{q \in \{1, \cdots, M\}} \mathcal{T}^q). \tag{5}$$

Let's assume the sub-trajectories $\mathsf{sub}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}^q\big) = \mathbf{s}_0, \mathbf{s}_{t_1^q}, \cdots, \mathbf{s}_{t_N^q}$ and their corresponding sampled trajectories as $\mathsf{smpl}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}^q\big)$. As the sampled timesteps $\mathcal{T}^q, q \in \{1, \cdots, M\}$ have no timestep in common other than 0 and their union covers the horizon $\mathcal{K}$, we can reformulate the original gradient $(\partial\mathcal{J}/\partial\theta = \sum_{k=1}^{K}(\partial\mathcal{J}/\partial\mathbf{s}_k)(\partial\mathbf{s}_k/\partial\theta))$ as

$$\frac{\partial\mathcal{J}}{\partial\theta} = \sum_{q=1}^{M}\left(\frac{\partial\mathcal{J}}{\partial\mathsf{sub}\big(\sigma[\mathbf{s}_0; \theta^{(j)}], \mathcal{T}^q\big)}\right)\left(\frac{\partial\mathsf{sub}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}^q\big)}{\partial\theta}\right).$$

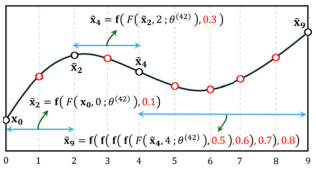
However, in our training process to compute the sampled gradient, we relax the sub-trajectories $\mathsf{sub}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}^q\big), q \in \{1, \cdots, M\}$ with their corresponding sampled trajectories $\mathsf{smpl}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}^q\big)$. In other words,

$$\left[\frac{\partial\mathcal{J}}{\partial\theta}\right]_{\text{sampled}} = \sum_{q=1}^{M}\left(\frac{\partial\mathcal{J}}{\partial\mathsf{smpl}\big(\sigma[\mathbf{s}_0 ; \theta^{(j)}], \mathcal{T}^q\big)}\right)\left(\frac{\partial\mathsf{smpl}\big(\sigma[\mathbf{s}_0; \theta^{(j)}], \mathcal{T}^q\big)}{\partial\theta}\right).$$

Remark 4.4. Unlike $\partial \mathbf{s}_{k^*}/\partial\theta$ and $\partial \mathsf{sub}\big(\sigma[\mathbf{s}_0\,;\theta^{(j)}], \mathcal{T}^q\big)/\partial\theta, q \in \{1,\cdots,M\}$ that are prone to vanish/explode problem, the alternatives, $\partial \mathsf{smpl}(\sigma[\mathbf{s}_0\,;\theta], \mathcal{T})\ (N)/\partial\theta$, and $\partial \mathsf{smpl}(\sigma[\mathbf{s}_0\,;\theta], \mathcal{T}^q)/\partial\theta$, $q \in \{1,\cdots,M\}$, can be computed efficiently.[6]

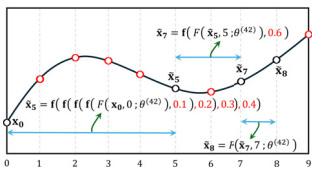
Example 5. Here, we propose an example to show our methodology to generate sampled trajectories when $\mathcal{J} = \tilde{\rho}$. We again consider Example 2, but we sample the trajectory with $M = 3$ sets of sampled timesteps $\mathcal{T}^1 = \{0, 2, 4, 9\}$, $\mathcal{T}^2 = \{0, 5, 7, 8\}$, and $\mathcal{T}^3 = \{0, 1, 3, 6\}$. Here, the timesteps are sampled such that their intersection is $\{0\}$ and their union is $\mathcal{K}$. The resulting sampled trajectory for $\mathcal{T}^1$ is $\mathsf{smpl}\big(\sigma[\mathbf{x}_0\,;\theta^{(42)}], \mathcal{T}^1\big) = \mathbf{x}_0, \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_4, \tilde{\mathbf{x}}_9$, where

$$
\begin{aligned}
\tilde{\mathbf{x}}_2 &= \mathbf{f}_1^{(42)}(\mathbf{x}_0) = \mathbf{f}(\ F(\mathbf{x}_0, 0\,;\theta^{(42)}),\ 0.1),\\
\tilde{\mathbf{x}}_4 &= \mathbf{f}_2^{(42)}(\tilde{\mathbf{x}}_2) = \mathbf{f}(\ F(\tilde{\mathbf{x}}_2, 2\,;\theta^{(42)}),\ 0.3),\\
\tilde{\mathbf{x}}_9 &= \mathbf{f}_3^{(42)}(\tilde{\mathbf{x}}_4) = \mathbf{f}(\ \mathbf{f}(\ \mathbf{f}(\ F(\tilde{\mathbf{x}}_4, 4\,;\theta^{(42)}),\ 0.5),\ 0.6),\ 0.7),\ 0.8),
\end{aligned}
\quad \Longrightarrow
$$



and the resulting sampled trajectory for $\mathcal{T}^2$ is $\mathsf{smpl}\ \big(\sigma[\mathbf{x}_0\,;\theta^{(42)}], \mathcal{T}^2\big) = \mathbf{x}_0, \tilde{\mathbf{x}}_5, \tilde{\mathbf{x}}_7, \tilde{\mathbf{x}}_8$, where

$$
\begin{aligned}
\tilde{\mathbf{x}}_5 &= \mathbf{f}_1^{(42)}(\mathbf{x}_0) = \mathbf{f}(\ \mathbf{f}(\ \mathbf{f}(\ F(\mathbf{x}_0, 0\,;\theta^{(42)}),\ 0.1),\ 0.2),\ 0.3),\ 0.4),\\
\tilde{\mathbf{x}}_7 &= \mathbf{f}_2^{(42)}(\tilde{\mathbf{x}}_5) = \mathbf{f}(\ F(\tilde{\mathbf{x}}_5, 5\,;\theta^{(42)}),\ 0.6),\\
\tilde{\mathbf{x}}_8 &= \mathbf{f}_3^{(42)}(\tilde{\mathbf{x}}_7) = F(\tilde{\mathbf{x}}_7, 7\,;\theta^{(42)}),
\end{aligned}
\quad \Longrightarrow
$$



and finally, the resulting sampled trajectory for $\mathcal{T}^3$ is $\mathsf{smpl}\ \big(\sigma[\mathbf{x}_0\,;\theta^{(42)}], \mathcal{T}^3\big) = \mathbf{x}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_3, \tilde{\mathbf{x}}_6$ that has been previously explained in Example 2. We emphasize that the introduced sampled trajectories are exclusively generated for gradient step $j = 42$ and we perform a new random sampling for the next iteration.

Remark 4.5. At the start of the training process, we can envision a desired path for the model to track. Tracking this path may not be sufficient to satisfy the temporal specification, but its availability is still valuable information, which its inclusion to the training process can expedite it. Therefore, we also utilize a desired path to generate a convex and efficient waypoint function (denoted by $\mathcal{J}^{\mathrm{wp}}\ (\sigma\ [\mathbf{s}_0\,;\theta])$) for our training process. However, Algorithm 1 performs effectively even without the waypoint function. Section 5.3.1 explores this aspect using a numerical example. Nonetheless, integrating a waypoint function enhances the efficiency of the training process.

We finally present our overall training procedure in Algorithm 1. Here, we use $\rho^{\varphi}(\sigma[\mathbf{s}_0\,;\theta])$ as shorthand for the non-smooth robustness degree of $\sigma[\mathbf{s}_0\,;\theta]$ w.r.t. φ at time 0, i.e., $\rho(\varphi, \sigma[\mathbf{s}_0\,;\theta], 0)$. We terminate the algorithm in line 2 if the robustness is greater than a pre-specified threshold $\bar{\rho} > 0$. We also evaluate the performance of the algorithm through challenging case studies. During each iteration of this algorithm, we compute the robustness value for an initial state $\mathbf{s}_0$ selected from the pre-sampled set of initial states $\widehat{\mathcal{I}}$ in line 3. This selection can be either random, or the initial state with the lowest robustness value in the set $\widehat{\mathcal{I}}$. The Boolean parameter use_smooth is provided to toggle the objective between robustness of the critical predicate and the smooth robustness for the DT-STL formula. We initialize this parameter use_smooth in line 3 to be False

[6]The efficiency results from the control parameters θ repeating in fewer timesteps over the trajectory, as most of them are fixed.

Table 2. Results on Different Case Studies

Case Study	Temporal Task	System Dimension	Time Horizon	NN Controller Structure	Number of Iterations	Runtime (seconds)	Optimization Setting $[M, N, N_1, N_2, \epsilon, b]$
12-D Quad-rotor	φ_3	12	45 steps	[13,20,20,10,4]	1,120	6,413.3	[9, 5, 30, 40, 10^{-5}, 5]
Multi-agent	φ_4	20	60 steps	[21,40,20]	2,532	6,298.2	[12, 5, 30, 1, 10^{-5}, 15]
6-D Quad-rotor and Frame	φ_5	7	1,500 steps	[8,20,20,10,4]	84	443.45	[100, 15, 30, 3, 10^{-5}, 15]
Dubins car	φ_6	2	1,000 steps	[3,20,2]	829	3,728	[200, 5, 60, 3, 10^{-5}, 15]

Here, b is the hyper-parameter we utilized to generate LB4TL in [32].

and further update it to True in line 21, in case the gradient from critical predicate is unable to increase the robustness. The lines 18, 19, and 21 aim to improve the detection of non-differentiable local maxima by employing a more accurate approach. This involves maintaining the direction of the gradient generated with the critical predicate, and exponentially reducing the learning rate until a small threshold ϵ is reached. If, even with an infinitesimal learning rate, this gradient fails to increase the robustness, it suggests a high likelihood of being in a non-differentiable local maximum.

5 Experimental Evaluation

In this section, we evaluate the performance of our proposed methodology. We executed all experiments for training with Algorithm 1 using our MATLAB toolbox.[7] These experiments were carried out on a laptop PC equipped with a Core i9 CPU. In all experiments performed using Algorithm 1, we utilize LB4TL as the smooth semantics. We also present an experiment in Section 5.5 to empirically demonstrate that NN feedback controllers provide robustness to noise compared to open-loop alternatives. Finally, we conclude this section with statistical verification of controllers.[8]

First, we provide a brief summary of results on evaluation of Algorithm 1. Following this, we elaborate on the specifics of our experimental configuration later in this section.

Evaluation Metric. We evaluate the effectiveness of our methodology outlined in Algorithm 1 through four case studies, each presenting unique challenges. First, we present two case studies involving tasks with long time horizons:

—6-dimensional quad-rotor combined with a moving platform with task horizon $K = 1{,}500$ timesteps.
—2-dimensional Dubins car with task horizon $K = 1{,}000$ timesteps.

Subsequently, we present two additional case studies characterized by high-dimensional state spaces:

—20-dimensional multi-agent system of 10 connected Dubins cars with task horizon $K = 60$ timesteps.
—12-dimensional quad-rotor with task horizon $K = 45$ timesteps.

Table 2 highlights the versatility of Algorithm 1 in handling above case studies. We use a diverse set of temporal tasks which include nested temporal operators and two independently moving objects (quad-rotor and moving platform case study). The detail of the experiments are also discussed as follows.

[7]The source code for the experiments is publicly available from https://github.com/Navidhashemicodes/STL_dropout
[8]Our results show that integrating a waypoint function in Algorithm 1 enhances the efficiency of the training process to a small extent.

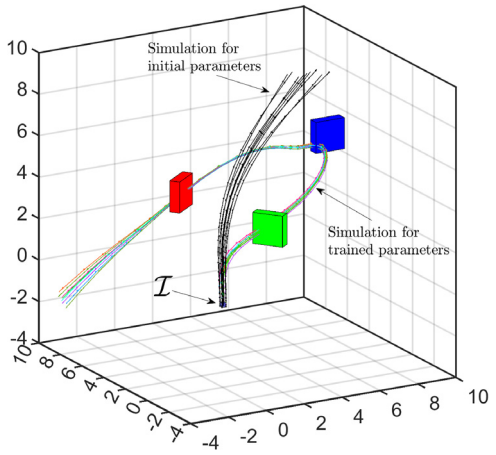


Fig. 6. This figure shows the simulation of trained control parameters to satisfy the specified temporal task in companion with the simulation result for initial guess for control parameters.

5.1 12-Dimensional Quad-Rotor (Nested 3-Future Formula)

We assume a 12-dimensional model for the quad-rotor of mass, $m = 1.4$ kg. The distance of rotors from the quad-rotor's center is also $\ell = 0.3273$ m and the inertia of vehicle is $J_x = J_y = 0.054$ and $J_z = 0.104$ (see [10] for the detail of quad-rotor's dynamics). The controller sends bounded signals $\delta_r, \delta_l, \delta_b, 0 \leq \delta_f \leq 1$ to the right, left, back, and front rotors, respectively, to drive the vehicle. Each rotor is designed such that given the control signal δ it generates the propeller force of $k_1\delta$ and also exerts the yawing torque $k_2\delta$ into the body of the quad-rotor. We set $k_1 = 0.75$ mg such that, the net force from all the rotors cannot exceed 3 times of its weight ($g = 9.81$). We also set $k_2 = 1.5\ell k_1$ to make it certain that the maximum angular velocity in the yaw axis is approximately equivalent to the maximum angular velocity in the pitch and roll axis. We use the sampling time $\delta t = 0.1$ seconds in our control process. The dynamics for this vehicle is proposed in Equation (6), where $F, \tau_\phi, \tau_\theta, \tau_\psi$ are the net propeller force, pitch torque, roll torque, and yaw torque, respectively. We plan to train an NN controller with **tanh**() activation function and structure $[13, 20, 20, 10, 4]$ for this problem that maps the vector, $[\mathbf{s}_k^\top, k]^\top$ to the unbounded control inputs $[a_{1,k}, a_{2,k}, a_{3,k}, a_{4,k}]^\top$. In addition to this, the trained controller should be valid for all initial states,

$$\mathcal{I} = \left\{ \mathbf{s}_0 \mid \left[-0.1, -0.1, -0.1, \vec{0}_{9\times 1}\right]^\top \leq \mathbf{s}_0 \leq \left[\ 0.1,\ \ 0.1,\ \ 0.1, \vec{0}_{9\times 1}\right]^\top \right\}$$

$$\begin{cases}
\dot{x}_1 = \cos(x_8)\cos(x_9)x_4 + (\sin(x_7)\sin(x_8)\cos(x_9) - \cos(x_7)\sin(x_9))x_5 \\
\quad +(\cos(x_7)\sin(x_8)\cos(x_9) + \sin(x_7)\sin(x_9))x_6 \\
\dot{x}_2 = \cos(x_8)\sin(x_9)x_4 + (\sin(x_7)\sin(x_8)\sin(x_9) + \cos(x_7)\cos(x_9))x_5 \\
\quad +(\cos(x_7)\sin(x_8)\sin(x_9) - \sin(x_7)\cos(x_9))x_6 \\
\dot{x}_3 = \sin(x_8)x_4 - \sin(x_7)\cos(x_8)x_5 - \cos(x_7)\cos(x_8)x_6 \\
\dot{x}_4 = x_{12}x_5 - x_{11}x_6 - 9.81\sin(x_8) \\
\dot{x}_5 = x_{10}x_6 - x_{12}x_4 + 9.81\cos(x_8)\sin(x_7) \\
\dot{x}_6 = x_{11}x_4 - x_{10}x_5 + 9.81\cos(x_8)\cos(x_7) - F/m \\
\dot{x}_7 = x_{10} + (\sin(x_7)(\sin(x_8)/\cos(x_8)))x_{11} + (\cos(x_7)(\sin(x_8)/\cos(x_8)))x_{12} \\
\dot{x}_8 = \cos(x_7)x_{11} - \sin(x_7)x_{12} \\
\dot{x}_9 = (\sin(x_7)/\cos(x_8))x_{11} + (\cos(x_7)/\cos(x_8))x_{12} \\
\dot{x}_{10} = -((J_y - J_z)/J_x)x_{11}x_{12} + (1/J_x)\tau_\phi \\
\dot{x}_{11} = ((J_z - J_x)/J_y)x_{10}x_{12} + (1/J_y))\tau_\theta \\
\dot{x}_{12} = (1/J_z)\tau_\psi
\end{cases}$$

$$\begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k_1 & k_1 & k_1 & k_1 \\ 0 & -\ell k_1 & 0 & \ell k_1 \\ \ell k_1 & 0 & -\ell k_1 & 0 \\ -k_2 & k_2 & -k_2 & k_2 \end{bmatrix} \begin{bmatrix} \delta_f \\ \delta_r \\ \delta_b \\ \delta_l \end{bmatrix}$$

$$\begin{aligned}
&\delta_f = 0.5(\tanh(0.5\ a_1) + 1), \\
&\delta_r = 0.5(\tanh(0.5\ a_2) + 1), \\
&\delta_b = 0.5(\tanh(0.5\ a_3) + 1), \\
&\delta_l = 0.5(\tanh(0.5\ a_4) + 1), \\
&a_1,\ a_2,\ a_3,\ a_4 \in \mathbb{R}.
\end{aligned} \tag{6}$$

Figure 6 shows the simulation of quad-rotor's trajectories with our trained controller parameters. The quad-rotor is planned to pass through the green hoop, between the 10th and 15th timestep.

Once it passed the green hoop it should pass the blue hoop in the future 10th to 15th timesteps and again once it has passed the blue hoop it should pass the red hoop again in the future next 10 to 15 timesteps. This is called a nested future formula, in which we design the controller such that the quad-rotor satisfies this specification. Assuming p as the position of quad-rotor, this temporal task can be formalized in DT-STL framework as follows:

$$\varphi_3 = \mathbf{F}_{[10,15]} \left(p \in \text{green\_hoop} \ \wedge \mathbf{F}_{[10,15]} \left(p \in \text{blue\_hoop} \ \wedge \mathbf{F}_{[10,15]} \left(p \in \text{red\_hoop} \right) \right) \right). \quad (7)$$

Figure 6 shows the simulation of trajectories, generated by the trained controller. The black trajectories are also the simulation of the initial guess for the controller, which are generated completely at random and are violating the specification. We sampled $\mathcal{I}$ with nine points, that are the corners of $\mathcal{I}$ including its center. The setting for gradient sampling is $M = 9,\ N = 5$. We trained the controller with $\bar{\rho} = 0$, in Algorithm 1 with optimization setting $(N_1 = 30,\ N_2 = 40,\ \epsilon = 10^{-5})$ over 1,120 gradient steps (runtime of 6,413.3 seconds). The runtime to generate LB4TL is also 0.495 seconds and we set $b = 5$ for it. Algorithm 1 utilizes gradients from waypoint function, critical predicate, and LB4TL, 515, 544, and 61 times, respectively.

5.2 Multi-Agent: Network of Dubins Cars (Nested Formula)

In this example, we assume a network of 10 different Dubins cars that are all under the control of an NN controller. The dynamics of this multi-agent system is

$$\begin{bmatrix} \dot{x}^i \\ \dot{y}^i \end{bmatrix} = \begin{bmatrix} v^i \cos(\theta^i) \\ v^i \sin(\theta^i) \end{bmatrix}, \qquad \begin{matrix} v^i \leftarrow \tanh(0.5a_1^i) + 1, a_1^i \in \mathbb{R} \\ \theta^i \leftarrow a_2^i \in \mathbb{R} \end{matrix}, \qquad i \in 1, \cdots, 10, \quad (8)$$

that is, a 20-dimensional multi-agent system with 20 controllers, $0 \leq v^i \leq 1 \theta^i \in \mathbb{R},\ i \in 1, \cdots, 10$. Figure 7(a) shows the initial position of each Dubins car in $\mathbb{R}^2$ in companion with their corresponding goal sets. The cars should be driven to their goal sets, and they should also keep a minimum distance of $d = 0.5$ m from each other while they are moving toward their goal sets. We assume a sampling time of $\delta t = 0.26$ seconds for this model, and we plan to train an NN controller with **tanh**() activation function and structure $[21, 40, 20]$ via Algorithm 1. For this problem, the controller maps the vector, $[\mathbf{s}_k^\top, k]^\top$ to the unbounded control inputs $\left\{a_{1,k}^i, a_{2,k}^i\right\}_{i=1}^{10}$. Note that $\mathbf{s}_k^i = (x_k^i, y_k^i)$. This temporal task can be formalized in DT-STL framework as follows:

$$\varphi_4 := \bigwedge_{i=1}^{10} \left(\mathbf{F}_{[20,48]} \mathbf{G}_{[0,12]} \left(\mathbf{s}^i \in \text{Goal}^i \right) \right) \qquad \bigwedge \qquad \bigwedge_{\substack{i \neq j \\ i,j \in \{1,\cdots,10\}}} \mathbf{G}_{[0,60]} \left(\|\mathbf{s}^i - \mathbf{s}^j\|_\infty > d \right).$$

Figure 7(c) shows the simulation of the trajectories for the trained controller, and Figure 7(b) presents the simulation of trajectories for the initial guess for control parameters. We observe that our controller manages the agents to finish the task in different times. Thus, we present the timestamps with asterisk markers to enhance the clarity of the presentation regarding satisfaction of the specification in Figure 7(c). Although the task is not a long horizon task, due to the high dimension and complexity of the task, we were unable to solve this problem without time sampling. However, we successfully solved this problem with Algorithm 1 within 6,298 seconds and 2,532 gradient steps.

We also set the optimization setting as $M = 12, N = 5, N_1 = 30, N_2 = 1, \epsilon = 10^{-5}$. The runtime to generate LB4TL is also 6.2 seconds and we set $b = 15$ for it. Over the course of the training process we utilized 187, 1,647, and 698 gradients from waypoint function, critical predicate, and LB4TL, respectively.

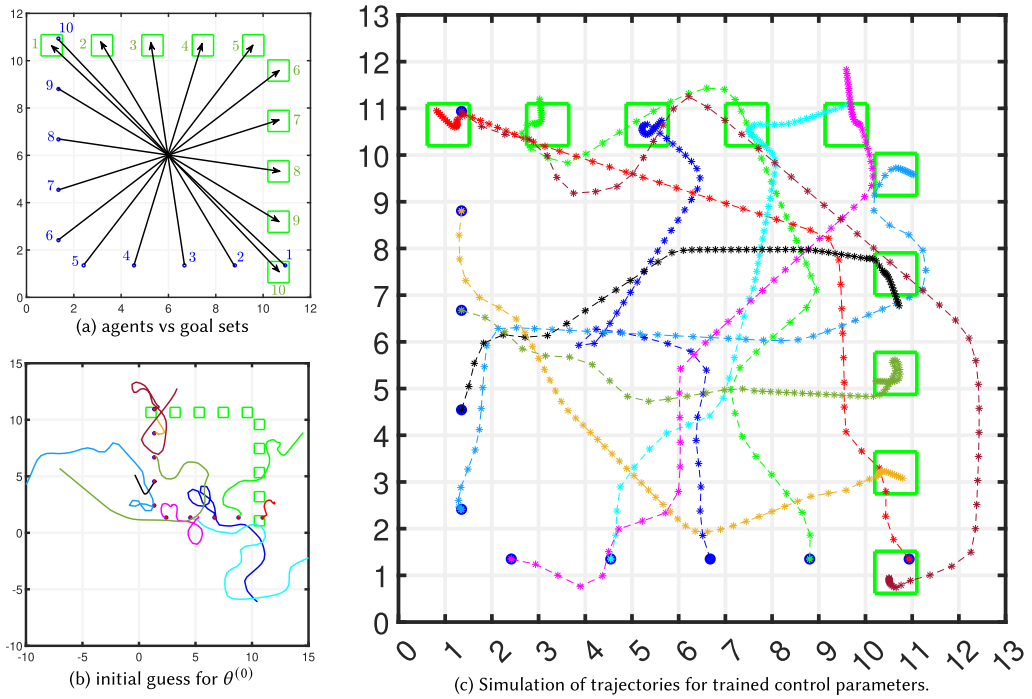


Fig. 7. These figures show a multi-agent system of 10 connected Dubins cars. (a) shows the start (blue dots) and goal points (green squares) for agents. (b) and (c) show simulated system trajectories with both the initial untrained controller and the centralized NN controller trained with Algorithm 1. The controller coordinates all cars to reach their respective goals between 20 and 48 seconds, and then stay in their goal location for at least 12 seconds. It also keeps the cars at a minimum distance from each other. We remark that the agents finish their tasks (the first component of φ_4) at different times.

5.3 6-Dimensional Quad-Rotor and Moving Platform: Landing a Quad-Rotor

We use a 6-dimensional model for quad-rotor dynamics as follows:

$$\begin{bmatrix}\dot{x} & \dot{y} & \dot{z} & \dot{v}_x & \dot{v}_y & \dot{v}_z\end{bmatrix} = \begin{bmatrix}v_x & v_y & v_z & g\tan(u_1) & -g\tan(u_2) & g-u_3\end{bmatrix}, \text{ where,}$$
$$u_1 \leftarrow 0.1\tanh(0.1a_1), \quad u_2 \leftarrow 0.1\tanh(0.1a_2), \quad u_3 \leftarrow g - 2\tanh(0.1a_3), \quad a_1, a_2, a_3 \in \mathbb{R}. \quad (9)$$

Let $\mathbf{x} = (x, y, z)$ denote the quad-rotor's position and $\mathbf{v} = (v_x, v_y, v_z)$ denote its velocity along the three coordinate axes. The control inputs u_1, u_2, u_3 represent the pitch, roll, and thrust inputs, respectively. We assume that the inputs are bounded as follows: $-0.1 \leq u_1, u_2 \leq 0.1,\ 7.81 \leq u_3 \leq 11.81$.

The horizon of the temporal task is 1,500 timesteps with $\delta t = 0.05s$. The quad-rotor launches at a helipad located at $(x_0, y_0, z_0) = (-40, 0, 0)$. We accept a deviation of 0.1 for x_0 and y_0 and train the controller to be valid for all the states sampled from this region. The helipad is also 40 m far from a building located at $(0, 0, 0)$. The building is 30 m high, where the building's footprint is 10 m $\times$10 m. We have also a moving platform with dimension 2 m $\times$ 2 m $\times$ 0.1 m that is starting to move from $(10, 0, 0)$ with a variable velocity, modeled as, $\dot{x}^f = u_4$. We accept a deviation of 0.1 for x_0^f, and our trained controller is robust with respect to this deviation. We define $\widehat{\mathcal{I}}$ with nine samples located at the corners of $\mathcal{I}$ and the center of $\mathcal{I}$. The frame is required to keep a minimum distance of 4.5 m from the building. We train the NN controller to control both the quad-rotor and

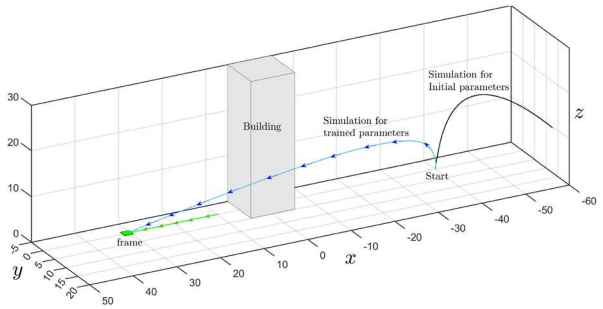


Fig. 8. This figure shows the simulated trajectory for trained controller in comparison to the trajectories for naive initial random guess. The frame is moving with a velocity determined with the controller that also controls the quad-rotor.

the platform to ensure that the quad-rotor will land on the platform with relative velocity of at most 1 m/s in x, y, and z directions, and its relative distance is at most 1 m in x, y direction and 0.4 m in z direction. Let $p = (x, y, z)$ be the position of the quad-rotor, this temporal task can be formulated as a reach-avoid formula in DT-STL framework as follows:

$$\varphi_5 = \mathbf{G}_{[0,1500]}\ (p \notin \text{obstacle})\ \wedge \mathbf{F}_{[1100,1500]}(p \in \text{Goal})\ \wedge \mathbf{G}_{[0,1500]}(x_k^f > 9.5), \tag{10}$$

where the goal set is introduced in Equation (11). We plot the simulated trajectory for the center of set of initial states $\mathcal{I}$, in Figure 8. The NN controller's structure is specified as $[8, 20, 20, 10, 4]$ and uses tanh() activation function. We initialize it with a random guess for its parameters. The simulated trajectory for initial guess of parameters is also depicted in black. The setting for gradient sampling is $M = 100,\ N = 15$. We trained the controller with $\bar{\rho} = 0$, over 84 gradient steps (runtime of 443 seconds). The runtime to generate LB4TL is also 7.74 seconds and we set $b = 15$, for it. In total, Algorithm 1 utilizes gradients from waypoint function, critical predicate, and LB4TL, 5, 71, and 8 times, respectively.

$$\text{Goal} = \left\{ \begin{bmatrix} x_k \\ y_k \\ z_k \\ v_{x,k} \\ v_{y,k} \\ v_{z,k} \\ x_k^f \end{bmatrix} \mid \begin{bmatrix} -1 \\ -1 \\ 0.11 \\ 0 \\ -1 \\ -1 \end{bmatrix} \leq \begin{bmatrix} x_k - x_k^f \\ y_k \\ z_k \\ v_{x,k} \\ v_{y,k} \\ v_{z,k} \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 0.6 \\ 2 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{11}$$

5.3.1 Influence of Waypoint Function, Critical Predicate, and Time Sampling on Algorithm 1. Here, we consider the case study of landing a quad-rotor, and perform an ablation study over the impact of including (1) critical predicate, (2) waypoint function, and (3) time sampling, in the training process via Algorithm 1. To that end, we compare the results once these modules are excluded from the algorithm. In the first step, we remove the waypoint function and show the performance of the algorithm. In the next step, we also disregard the presence of critical time in time sampling and train the controller with completely at random time sampling, and finally we examine the impact of time sampling on the mentioned results. Table 3 shows the efficiency of training process in each case, and Figure 9 compares the learning curves. Our experimental result shows the control synthesis for quad-rotor (landing mission) faces a small reduction in efficiency when the waypoint

Table 3. Ablation Studies for Picking Different Options for the Optimization Process

Learning Curve's Color in Figure 9	Waypoint Function	Critical Predicate	Time Sampling	Number of Iterations	Runtime
	✓	✓	✓	84	443 seconds
	×	✓	✓	107	607 seconds
	✓	×	✓	DNF[−0.74]	6971 seconds
	×	×	✓	DNF[−1.32]	4822 seconds
	✓	✓	×	DNF[−4.52]	1505 seconds
	×	✓	×	DNF[−11.89]	1308 seconds

This table shows the results of the training algorithm in case study 5.3.1. We indicate that the training does not result in positive robustness within 300 gradient steps by DNF (*did not finish*) with the value of robustness in iteration 300 in brackets. The table represents an ablation study, where we disable the various heuristic optimizations in Algorithm 1 in different combinations and report the extent of reduction in efficiency. We use ✓, × to respectively indicate a heuristic being included or excluded. The time sampling technique is utilized in all the experiments.

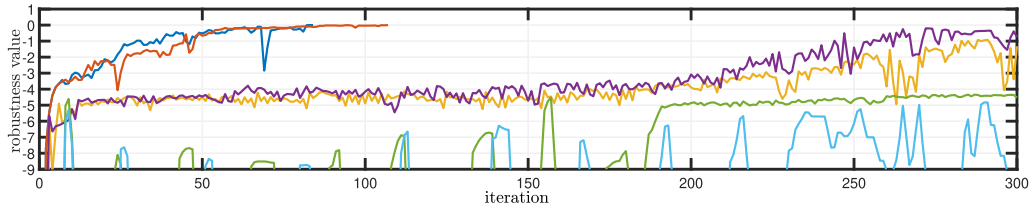


Fig. 9. This figure shows the learning curve for training processes. Note, the figure has been truncated and the initial robustness for all the experiments at iteration 0 is −47.8. This figure shows that Algorithm 1 in the presence of the waypoint function concludes successfully in 84 iterations while when the waypoint function is not included, it terminates in 107 iterations. The algorithm also fails if the critical predicate is not considered in time sampling.

function is disregarded and fails when the critical predicate is also removed from time sampling. This also shows that control synthesis fails when time sampling is removed.

5.4 Dubins Car: Growing Task Horizon for Dubins Car (Ablation Study on Time Sampling)

In this experiment, we consider Dubins car with dynamics,

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}v\cos(\theta)\\ v\sin(\theta)\end{bmatrix}, \quad v \leftarrow \tanh(0.5a_1)+1, a_1 \in \mathbb{R}, \quad \theta \leftarrow a_2 \in \mathbb{R},$$

and present an ablation study on the influence of gradient sampling on control synthesis. Given a scale factor $a > 0$, a time horizon K, and a pre-defined initial guess for control parameters $\theta^{(0)}$, we plan to train a tanh() NN controller with structure $[3, 20, 2]$, to drive a Dubins car, to satisfy the temporal task, $\varphi_6 := \mathbf{F}_{[0.9K,K]}\,(p \in \text{Goal}) \wedge \mathbf{G}_{[0,K]}\,(p \notin \text{Obstacle})$, where $p = (x, y)$ is the position of Dubins car. The Dubins car starts from $(x_0, y_0) = (0, 0)$. The obstacle is also a square centered on $(a/2, a/2)$ with the side length $2a/5$. The goal region is again a square centered on $(9a/10, 9a/10)$ with the side length $a/20$. We solve this problem for $K = 10$, 50, 100, 500, 1,000 and we also utilize $a = K/10$ for each case study. We apply standard gradient ascent (see Algorithm 2) to solve each case study, both with and without gradient sampling.

Algorithm 2: Standard Gradient Ascent Backpropagation via Smooth Semantics

Initialize variables
while $\left(\min_{\mathbf{s}_0 \in \widehat{I}} \left(\rho(\varphi, \sigma[\mathbf{s}_0 ; \theta^{(j)}], 0)\right) < \bar{\rho}\right)$ **do**
  $\mathbf{s}_0 \leftarrow$ `Sample from` $\widehat{I}$
  $\sigma[\mathbf{s}_0 ; \theta^{(j)}] \leftarrow$ `Simulate using policy` $\pi_{\theta^{(j)}}$
  $d \leftarrow \nabla_\theta \tilde{\rho}(\sigma[\mathbf{s}_0\ \theta^{(j)}])$ `using` $\sigma[\mathbf{s}_0 ; \theta^{(j)}]$
  $\theta^{(j+1)} \leftarrow \theta^{(j)} + \mathsf{Adam}(d)$
  $j \leftarrow j + 1$

Table 4. Ablation Study

Horizon	Standard Gradient Ascent (No Time Sampling)		Standard Gradient Ascent (With Time Sampling)		Algorithm 1(No Waypoint) (With Time Sampling)		Algorithm 1(with Waypoint) (With Time Sampling)	
	Num. of Iterations	Runtime (seconds)	Num. Iterations	Runtime (seconds)	Num. of Iterations	Runtime (seconds)	Num. of Iterations	Runtime (seconds)
10	34	2.39	11	1.39	6	0.9152	4	5.61
50	73	2.46	53	14.01	20	2.7063	25	6.09
100	152	8.65	105	112.6	204	79.33	157	90.55
500	DNF[−1.59]	4,986	3,237	8,566	2,569	2,674	624	890.24
1,000	DNF[−11.49]	8,008	DNF[−88.42]	28,825	812	1,804	829	3,728

We mark the experiment with DNF[.] if it is unable to provide a positive robustness within 8,000 iterations, and the value inside brackets is the maximum value of robustness it finds. We magnify the environment proportional to the horizon. All experiments for $K = 10, 50, 100$ use a unique guess for initial parameter values, and all the experiments for $K = 500, 1,000$ use another unique initial guess. Here, we utilized critical predicate module in both cases of Algorithm 1 (columns 3 and 4).

Furthermore, in addition to standard gradient ascent, we also utilize Algorithm 1 to solve them. Consider we set the initial guess and the controller's structure similar, for all the training processes, and we also manually stop the process once the number of iterations exceeds 8,000 gradient steps. We also assume a singleton as the set of initial states $\{(0, 0)\}$ to present a clearer comparison. The runtime and the number of iterations for each training process is presented in Table 4. Figure 10 displays the simulation of trajectories trained using Algorithm 1 for $K = 1{,}000$ timesteps (via gradient sampling), alongside the simulations for the initial guess of controller parameters.

Table 4 shows our approximation technique outperforms the original gradient when the computation for original gradient faces numerical issues (such as longer time horizons $K = 500, 1{,}000$). However, in case the computation for original gradient does not face any numerical issues, then the original gradient outperforms the sampled gradient which is expected. This table also shows that the standard gradient ascent (with time sampling) is still unable to solve for the case $K = 1{,}000$ while Algorithm 1 solves for this case efficiently. This implies the combination of time sampling, critical predicate, and safe-resmoothing provides significant improvement in terms of scalability. The experiment $K = 500$ in this table also shows that inclusion of waypoint in Algorithm 1 is sometimes noticeably helpful.

5.5 Robustness of NN Feedback Controllers over Open-loop Alternatives

In this section, we empirically demonstrate that feedback NN controllers are more robust to noise and uncertainties compared to open-loop controllers, even when the feedback controller is not

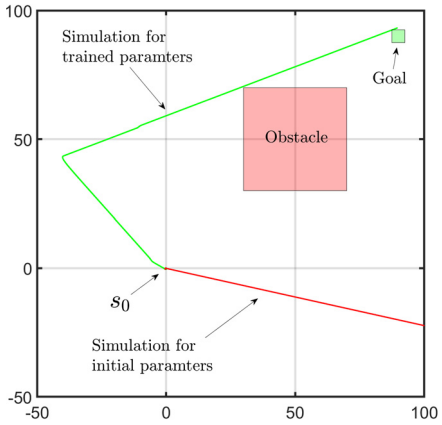


Fig. 10. This figure shows the simulation of the results for Dubins car in the ablation study proposed in Section 5.4. In this experiment, the task horizon is 1,000 timesteps.

trained in the presence of noise. We then show that if we train the feedback controller after introducing a stochastic noise in the original system dynamics, the performance vastly outperforms open-loop control trained in the presence of noise. To illustrate, we use the example proposed in [42] but add a stochastic noise and also include some uncertainty on the choice of initial condition. The modified system dynamics are shown in Equation (12), where the sampling time $\mathbf{d}t = 0.1$:

$$\mathbf{s}_{k+1} = \mathbf{s}_k + u_k \mathbf{d}t + c_1 v_k, \qquad \mathbf{s}_0 = [-1,\ -1]^\top + c_2 \eta. \tag{12}$$

Here, for $k \in 1, \cdots, K$ and v_k and η are both i.i.d. random variables with standard distribution, e.g., $\eta, v_k \sim \mathcal{N}(0_{2\times 1}, I_{2\times 2})$, where $I_{2\times 2}$ is the identity matrix. In this example, the desired objective for the system is

$$\varphi_8 = \mathbf{F}_{[0,44]}\left(\mathbf{G}_{[0,5]}\left(\mathbf{s} \in \text{Goal}_1\right)\right) \bigwedge \mathbf{F}_{[0,44]}\left(\mathbf{G}_{[0,5]}\left(\mathbf{s} \in \text{Goal}_2\right)\right) \bigwedge \mathbf{G}_{[0,49]} \neg \left(\mathbf{s} \in \text{Unsafe}\right), \tag{13}$$

where the regions Goal_1, Goal_2, and Unsafe are illustrated in Figure 11.[9]

In the first step of the experiment, we train the feedback and open-loop controllers in the absence of the noise ($c_1 = c_2 = 0$) and deploy the controllers on the noisy environment ($c_1 = 0.0314, c_2 = 0.0005$) and compare their success rate.[10] In the second step of the experiment, we train both the feedback and open-loop controllers on the noisy environment ($c_1 = 0.0314, c_2 = 0.0005$), and also deploy them in the noisy environment ($c_1 = 0.0314, c_2 = 0.0005$) to compare their success rate. If we train the open-loop and NN feedback controllers in the absence of noise, then the controllers will, respectively, satisfy the specification in 3.7% and 65.4% of trials when deployed in a noisy environment. However, we can substantially improve performance of feedback controllers by training in the presence of noise; here, the controllers satisfy the specification 5.4% and 94.4% of trials, respectively, showing that the NN feedback controller has better overall performance in the presence of noise, which open-loop control lacks.

[9]We also add the following updates to the original problem presented in [42]:
—We omit the requirement $\mathbf{s}_K = [1,\ 1]^\top$ from both control problems for simplicity.
—We increase the saturation bound of the controller to $u_k \leq 4\sqrt{2}$. We also apply this condition to the open-loop controller proposed in [42].

[10]To report the success rate, we deploy the controllers 1,000 different times and compute the percentage of the trajectories that satisfy the specification.

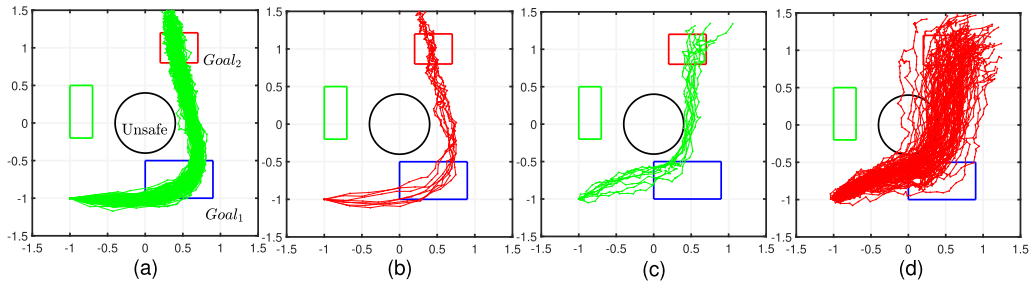


Fig. 11. This figure shows the simulation of trajectories when the trained controller is deployed on the noisy deployment environment, both controllers are trained in the presence of noise. The trajectories of NN feedback controller that satisfy (a) and violate (b) the specification and those of the open-loop controller that satisfy (c) and violate (d) the specification are shown.

We utilized STLCG PyTorch toolbox [42] to solve for the open-loop controller. We also utilized the standard gradient ascent proposed in Algorithm 2 (via LB4TL as smooth semantics $\tilde{\rho}$) for training the feedback controllers. We let the training process in Algorithm 2 and STLCG to run for 5,000 iterations, and then terminated the training process. Figure 11 shows the simulation of trained controllers when they are deployed to the noisy environment. Here, we generate 100 random trajectories via trained controllers and plot them in green and red when they satisfy or violate the specification, respectively. However, all trained feedback controllers in this article exhibit the same level of robustness to noise.

5.6 Statistical Verification of Synthesized Controllers

In [33], we showed that if the trained NN controller, the plant dynamics, and the NN representing the STL quantitative semantics all use ReLU activation functions, then we can use tools such as NNV [66] that compute the forward image of a polyhedral input set through an NN to verify whether a given DT-STL property holds *for all* initial states of the system. However, there are few challenges in applying such deterministic methods here: we use more general activation functions, the depth of the overall NN can be significant for long-horizon tasks, and the dimensionality of the state-space can also become a bottleneck. In this article, we thus eschew the use of deterministic techniques, instead reasoning about the correctness of our NN feedback control scheme using a statistical verification approach. In other words, given the coverage level $\delta_1 \in (0,1)$ and confidence level $\delta_2 \in (0,1)$ we are interested in a probabilistic guarantee of the form, $\Pr[\ \Pr[\sigma[\mathbf{s}_0;\theta] \models \varphi] \ \geq \delta_1\ ] \geq \delta_2$.

The main inspiration for our verification is drawn from the theoretical developments in conformal prediction [70]. Of particular significance to us is the following lemma:

Lemma 5.1 (From [16]). *Consider m independent and identically distributed (i.i.d.), real-valued data points drawn from some distribution $\mathcal{D}$. After they are drawn, suppose we sort them in ascending order and denote the ith smallest number by R_i,(i.e., we have $R_1 < R_2 < \ldots < R_m$). Let $\mathbf{Beta}(\alpha,\beta)$ denote the Beta distribution.*[11] *Then, for an arbitrary R_{m+1} drawn from the same distribution $\mathcal{D}$, the following holds:*

$$\Pr\left[R_{m+1} < R_\ell\right] \sim \mathbf{Beta}(\ell, m+1-\ell), \quad 1 \leq \ell \leq m. \tag{14}$$

[11]The Beta distribution is a family of continuous probability distributions defined on the interval $0 \leq x \leq 1$ with shape parameters α and β, and with probability density function $f(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$, where the constant $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\mathbf{d}t$ is the Gamma function.

The original m i.i.d. data points are called a *calibration set*. The above lemma says that the probability for a *previously unseen* data point R_{m+1} drawn from the same distribution $\mathcal{D}$ being less than the lth smallest number in the calibration set is itself a random variable that has a specific Beta distribution. We next show how we can exploit this lemma to obtain probabilistic correctness guarantees for our trained controllers.

We assume that there is some user-specified distribution over the set of initial states in $\mathcal{I}$, and that we can sample m initial states $\mathbf{s}_{0,1}, \ldots, \mathbf{s}_{0,m}$ from this distribution. For a sampled initial state $\mathbf{s}_{0,i}, i \in 1, \cdots, m$, we can obtain the corresponding negative robust satisfaction value, and set: $R_i = -\rho(\varphi, \sigma[\mathbf{s}_{0_i}; \theta], 0), i \in 1, \cdots, m$.

From Lemma 5.1, we know that for a previously unseen initial state $\mathbf{s}_{0,m+1}$, the corresponding (negative value of) robustness R_{m+1} satisfies the relation in Equation (14). Now, almost all sampled trajectories generated by a trained controller are expected to have positive robustness value, so we expect the quantities $R_1, \ldots, R_m$ to be all negative. In the pessimistic case, we expect at least the first ℓ of these quantities to be negative. If so, the guarantee in Equation (14) essentially quantifies the probability of the robustness of a trajectory for a previously unseen initial state to be positive. Note that

$$(R_{m+1} < R_\ell) \wedge (R_\ell < 0) \implies (R_{m+1} < 0) \implies (\sigma[\mathbf{s}_{0,m+1}; \theta] \models \varphi) \tag{15}$$

$$\therefore \Pr(\sigma[\mathbf{s}_{0,m+1}; \theta] \models \varphi) \geq \Pr(R_{m+1} < R_\ell) \sim \mathbf{Beta}(\ell, m+1-\ell) \tag{16}$$

In addition, from [16] we know that the mean and variance of the Beta distribution are given as follows:

$$\mathbb{E}\big[\Pr[R_{m+1} < R_\ell]\big] = \frac{\ell}{m+1} \quad \mathbf{Var}\big[\Pr[R_{m+1} < R_\ell]\big] = \frac{\ell(m+1-\ell)}{(m+1)^2(m+2)}. \tag{17}$$

As the Beta distribution has small variance and is noticeably sharp, the desired coverage level for a probabilistic guarantee can be obtained in the vicinity of its mean value. From the closed form formula in Equation (17), we observe that in case we wish to have a coverage level close to $(1 - 10^{-4})$ or 99.99%, then we can set $\ell = \lceil (m+1)(1-10^{-4}) \rceil$. Here we also set m to 10^5, giving the value of $\ell = 99{,}991$. Let's denote $\Pr[R_{m+1} < R_\ell]$ as δ. Since δ is a random variable sampled from $\mathbf{Beta}(\ell, m+1-\ell)$ where $((\ell = 99{,}991, m = 10^5)$,[12] we can utilize the cumulative density function of Beta distribution (i.e., regularized incomplete Beta function) and for a given $\delta_1 \in (0, 1)$ propose the following guarantee,

$$\Pr[\delta \geq \delta_1] = 1 - I_{\delta_1}(\ell, m+1-\ell), \text{ where } I_x(\,.\,,\,.\,) \text{ is the regularized incomplete Beta function at point } x.$$

Here δ_1 is the desired confidence level that we consider for the probabilistic guarantee. However, if we set $\delta_1 = 0.9999$ then $\Pr[\delta \geq 0.9999] = 0.54$ which indicates that the confidence in the 99.99% guarantee is low. If we instead set $\delta = 0.9998$, this results in $\Pr[\delta \geq 0.9998] = 0.995$, which indicates a much higher level of confidence. Finally, based on Equation (16), we can consider the provided guarantee also for the trajectories and conclude,

$$\Pr[\quad \Pr[\sigma[\mathbf{s}_0; \theta] \models \varphi] \quad \geq 99.98\% \quad] \quad \geq 99.5\%. \tag{18}$$

To summarize, in each of our case studies, we sample $m = 10^5$ i.i.d. trajectories, compute their sorted negative robustness values $R_1, \ldots, R_m$, and check that R_ℓ for $\ell = 99{,}991$ is indeed negative. This gives us the probabilistic guarantee provided in Equation (18) that from unseen initial conditions the system will not violate the DT-STL specification.

[12] We can compute for its mean and variance via Equation (17) as $\mu = \mathbb{E}[\delta] = 0.9999$ and $\mathbf{var}[\delta] = 9.9987 \times 10^{-10}$.

6 Related Work and Conclusion

Related Work. In the broad area of formal methods, robotics, and cyber-physical systems, there has been substantial research in synthesizing controllers from temporal logic specifications. This research involves different considerations. First, the plant dynamics may be specified as either a differential/difference equation-based model [22, 25, 27, 46, 51, 55, 56], or as a Markov decision process [29, 37, 59] that models stochastic uncertainty, or may not be explicitly provided (but is implicitly available through a simulator that samples model behaviors). The second consideration is the expressivity of the specification language, i.e., if the specifications are directly on the real-valued system behaviors or on Boolean-valued propositions over system states, and if the behaviors are over a discrete set of timesteps or over dense time. Specification languages such as **Linear Temporal Logic (LTL)** [53], **Metric Temporal Logic (MTL)** [39], and **Metric Interval Temporal Logic (MITL)** [3] are over Boolean signals, while STL [48] and DT-STL considered in this article are over real-valued behaviors. MTL, MITL, and STL are typically defined over dense time signals while LTL and DT-STL are over discrete timesteps. The third consideration is the kind of controller being synthesized. Given the plant dynamics, some techniques find the entire sequence of control actions from an initial state to generate a desired optimal trajectory (open-loop control) [46, 50, 55, 75], while some focus on obtaining a feedback controller that guarantees satisfaction of temporal logic objectives in the presence of uncertainty [76] (in the initial states or during system execution). We now describe some important sub-groups of techniques in this space that may span the categories outlined above.

Reactive Synthesis. A reactive synthesis approach models the system interaction with its environment as a turn-based game played by the system and the environment over a directed graph [13]. The main idea is to convert temporal logic specifications (such as LTL) into winning conditions and identify system policies that deterministically guarantee satisfaction of the given specification [40]. As reactive synthesis is a computationally challenging problem, there are many sub-classes and heuristics that have been explored for efficiency; for instance, in [73] a receding horizon framework is used; in [67], the authors focus on piece-wise affine non-deterministic systems, while [56] investigates reactive synthesis for STL.

*Reinforcement and Deep **Reinforcement Learning (RL)**.* RL algorithms learn control policies that maximize cumulative rewards over long-term horizons. Recently, RL temporal has been used to infer reward functions that can guarantee satisfaction of an LTL specification [14, 24, 31, 60]. The work in [9, 30, 38, 54, 69] generates reward functions from STL specifications. While the ultimate objective of these methods is similar to our problem setting, we adopt a model-based approach to control synthesis where we assume access to a differentiable model of the system and use gradient ascent to train the controller in contrast to RL algorithms that may rely on adequate exploration of the state space to obtain near-optimal policies (that may guarantee satisfaction of specifications).

***Model Predictive Control (MPC)** and **Mixed Integer Linear Programming (MILP)**.* A clever encoding of LTL as mixed integer linear constraints was presented in [72] for the purpose of reactive synthesis. This idea was then extended in [55] to show that model predictive control of linear/piece-wise affine systems w.r.t. STL objectives (with linear predicates) can be solved using MILP solvers. MILP is an NP-hard problem, and various optimization improvements to the original problem [25, 41, 50, 65] and extensions to stochastic systems [22, 61] have been proposed. In contrast to a model-predictive controller, we obtain an NN feedback controller that does not require online optimization required in MPC.

Barrier Function-based Approaches. A **Control Barrier Function (CBF)** can be thought of as a safety envelope for a controlled dynamical system. As long as the CBF satisfies validity conditions (typically over its Lie derivative), the CBF guarantees the existence of a control policy that keeps

the overall system safe [74]. CBFs can be used to enforce safety constraints and also to enforce temporal specifications such as STL [4, 5, 17, 46]. The design of barrier functions is generally a hard problem, though recent research studies compute for the CBFs through learning [57, 69], and using quantitative semantics of STL [34].

Gradient-based Optimization Methods. This class of methods investigates learning NN controllers by computing the gradient of the robustness function of STL through back-propagation STL. For instance, training feedback NN controllers is studied in [34, 35, 47, 75, 76] and for open-loop controllers is investigated in [44]. The main contributions in this article over previous work are to scale gradient descent to long time horizons using the novel idea of dropout, and a more efficient (and smooth) computation graph for STL quantitative semantics.

Prior Work on NN Controllers for STL. The overall approach of this article is the closest to the work in [33, 34, 42, 43, 76], where STL robustness is used in conjunction with back-propagation to train controllers. The work in this article makes significant strides in extending previous approaches to handle very long horizon temporal tasks, crucially enabled by a novel sampling-based gradient approximation. Due to the structure of our NN-controlled system, we can seamlessly handle time-varying dynamics and complex temporal dependencies. We also note that while some previous approaches focus on obtaining open-loop control policies, we focus on synthesizing closed-loop, feedback NN-controllers which can be robust to minor perturbations in the system dynamics. In addition, we cover a general DT-STL formula for synthesis, and we utilize LB4TL [32] for backward computation that has shown significant improvement for efficiency of training over complex DT-STL formulas.

Limitations. Some of the key limitations of our approach include the following: (1) we do not address infinite time horizon specifications; (2) we only consider a DT-STL; (3) our approach would fail if the chosen NN architecture for the controller has too few parameters (making it difficult to control highly non-linear environment dynamics) or if it has too many parameters (making it a difficult optimization problem); (4) we assume full system observability and do not consider stochastic dynamics.

Conclusion. Using NN feedback controllers for control synthesis offers robustness against noise and uncertainties, making them preferable over open-loop controllers. However, training these controllers can be challenging due to issues like vanishing or exploding gradients, especially in long time horizons or high-dimensional systems. To address this challenge, we introduced a gradient sampling technique inspired by dropout [64] and stochastic depth [36]. Additionally, we proposed incorporating critical predicates into this technique to enhance training efficiency, and we tested our approach on various challenging control synthesis problems.

References

[1] Houssam Abbas and Georgios Fainekos. 2013. Computing descent direction of MTL robustness for non-linear systems. In *2013 American Control Conference*. IEEE, 4405–4410.

[2] Takumi Akazaki and Ichiro Hasuo. 2015. Time robustness in MTL and expressivity in hybrid system falsification. In *International Conference on Computer Aided Verification*. Springer, 356–374.

[3] Rajeev Alur. 1991. *Techniques for Automatic Verification of Real-Time Systems*. Stanford University.

[4] Aaron D. Ames, Jessy W. Grizzle, and Paulo Tabuada. 2014. Control barrier function based quadratic programs with application to adaptive cruise control. In *53rd IEEE Conference on Decision and Control*. IEEE, 6271–6278.

[5] Aaron D. Ames, Xiangru Xu, Jessy W. Grizzle, and Paulo Tabuada. 2016. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control* 62, 8 (2016), 3861–3876.

[6] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. arXiv:1606.06565. Retrieved from https://arxiv.org/abs/1606.06565

[7] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. 2011. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 254–257.

[8] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. arXiv:1607.06450.
[9] Anand Balakrishnan and Jyotirmoy V. Deshmukh. 2019. Structured reward shaping using signal temporal logic specifications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3481–3486.
[10] Randal Beard. 2008. Quadrotor Dynamics and Control Rev 0.1.
[11] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. 2017. *Formal Methods for Discrete-Time Dynamical Systems*, Vol. 89. Springer.
[12] Luigi Berducci, Edgar A. Aguilar, Dejan Ničković, and Radu Grosu. 2021. Hierarchical potential-based reward shaping from task specifications. arXiv:2110.02792. Retrieved from https://arxiv.org/abs/2110.02792
[13] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. 2018. Graph games and reactive synthesis. In *Handbook of Model Checking*, 921–962.
[14] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. 2020. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 10349–10355.
[15] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 31.
[16] Herbert A. David and Haikady N. Nagaraja. 2004. *Order Statistics*. John Wiley & Sons.
[17] Jyotirmoy V. Deshmukh, James Kapinski, Tomoya Yamaguchi, and Danil V. Prokhorov. 2019. Learning deep neural network controllers for dynamical systems with safety guarantees. In *the International Conference on Computer-Aided Design (ICCAD)*, 1–7.
[18] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.
[19] Georgios Fainekos and George J. Pappas. 2006. Robustness of temporal logic specifications. In *Formal Approaches to Testing and Runtime Verification* (LNCS, Vol. 4262). Springer, 178–192.
[20] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. 2009. Temporal logic motion planning for dynamic robots. *Automatica* 45, 2 (2009), 343–352.
[21] Bin Fang, Shidong Jia, Di Guo, Muhua Xu, Shuhuan Wen, and Fuchun Sun. 2019. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications* 3 (2019), 362–369.
[22] Samira S. Farahani, Vasumathi Raman, and Richard M. Murray. 2015. Robust model predictive control for signal temporal logic synthesis. *IFAC-PapersOnLine* 48, 27 (2015), 323–328.
[23] Thomas Parisini, Danil Prokhorov, Donald C. Wunsch, Frank L. Lewis, and Jie Huang. 2005. Special issue on neural networks for feedback control systems. *IEEE Transactions on Neural Networks* 16, 6 (2005), 423–442. DOI: https://doi.org/10.1109/TNN.2007.902966
[24] Jie Fu and Ufuk Topcu. 2014. Probably approximately correct MDP learning and control with temporal logic constraints. arXiv:1404.7073. Retrieved from https://arxiv.org/abs/1404.7073
[25] Yann Gilpin, Vince Kurtz, and Hai Lin. 2020. A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters* 5, 1 (2020), 241–246.
[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.
[27] Meng Guo and Michael M. Zavlanos. 2018. Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control* 63, 12 (2018), 4051–4066.
[28] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J. Russell, and Anca Dragan. 2017. Inverse reward design. In *Advances in Neural Information Processing Systems*, 30.
[29] Sofie Haesaert, Sadegh Soudjani, and Alessandro Abate. 2018. Temporal logic control of general Markov decision processes by approximate policy refinement. *IFAC-PapersOnLine* 51, 16 (2018), 73–78.
[30] Nathaniel Hamilton, Preston K. Robinette, and Taylor T. Johnson. 2022. Training agents to satisfy timed and untimed signal temporal logic specifications with reinforcement learning. In *International Conference on Software Engineering and Formal Methods*. Springer, 190–206.
[31] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2018. Logically-constrained reinforcement learning. arXiv:1801.08099. Retrieved from https://arxiv.org/abs/1801.08099
[32] Navid Hashemi, Samuel Williams, Bardh Hoxha, Danil Prokhorov, Georgios Fainekos, and Jyotirmoy Deshmukh. 2024. LB4TL: A smooth semantics for temporal logic to train neural feedback controllers. *IFAC-PapersOnline* 58, 11 (2024), 183–188.
[33] Navid Hashemi, Bardh Hoxha, Tomoya Yamaguchi, Danil Prokhorov, Georgios Fainekos, and Jyotirmoy Deshmukh. 2023. A neurosymbolic approach to the verification of temporal logic properties of learning-enabled control systems. In *International Conference on Cyber-Physical Systems (ICCPS)*, 98–109.
[34] Navid Hashemi, Xin Qin, Jyotirmoy V. Deshmukh, Georgios Fainekos, Bardh Hoxha, Danil Prokhorov, and Tomoya Yamaguchi. [n. d.]. Risk-awareness in learning neural controllers for temporal logic objectives. In *2023 American Control Conference (ACC)*, 4096–4103.

[35] Wataru Hashimoto, Kazumune Hashimoto, and Shigemasa Takai. 2022. Stl2vec: Signal temporal logic embeddings for control synthesis with recurrent neural networks. *IEEE Robotics and Automation Letters* 7, 2 (2022), 5246–5253.
[36] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. 2016. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference*, Proceedings, Part IV 14. Springer, 646–661.
[37] Krishna C. Kalagarla, Rahul Jain, and Pierluigi Nuzzo. 2020. Synthesis of discounted-reward optimal policies for Markov decision processes under linear temporal logic specifications. arXiv:2011.00632. Retrieved from https://arxiv.org/abs/2011.00632
[38] Parv Kapoor, Anand Balakrishnan, and Jyotirmoy V. Deshmukh. 2020. Model-based reinforcement learning from signal temporal logic specifications. arXiv:2011.04950. Retrieved from https://arxiv.org/abs/2011.04950
[39] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-Time Systems* 2, 4 (1990), 255–299.
[40] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics* 25, 6 (2009), 1370–1381.
[41] Vincent Kurtz and Hai Lin. 2022. Mixed-integer programming for signal temporal logic with fewer binary variables. *IEEE Control Systems Letters* 6 (2022), 2635–2640.
[42] Karen Leung, Nikos Aréchiga, and Marco Pavone. 2019. Backpropagation for parametric STL. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 185–192.
[43] Karen Leung, Nikos Arechiga, and Marco Pavone. 2021. Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. In *Algorithmic Foundations of Robotics XIV*. Steven M. LaValle, Ming Lin, Timo Ojala, Dylan Shell, and Jingjin Yu (Eds.), Springer, 432–449.
[44] Karen Leung, Nikos Aréchiga, and Marco Pavone. 2023. Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. *The International Journal of Robotics Research* 42, 6 (2023), 356–370.
[45] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. 2017. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3834–3839.
[46] Lars Lindemann and Dimos V. Dimarogonas. 2018. Control barrier functions for signal temporal logic tasks. *IEEE Control Systems Letters* 3, 1 (2018), 96–101.
[47] Wenliang Liu, Noushin Mehdipour, and Calin Belta. 2021. Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints. *IEEE Control Systems Letters* 6 (2021), 91–96.
[48] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.
[49] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. 2022. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations*.
[50] Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. 2017. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 1235–1240.
[51] Yash Vardhan Pant, Houssam Abbas, Rhudii A. Quaye, and Rahul Mangharam. 2018. Fly-by-logic: Control of multi-drone fleets with temporal logic objectives. In *the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 186–197.
[52] Yash Vardhan Pant, He Yin, Murat Arcak, and Sanjit A. Seshia. 2021. Co-design of control and planning for multi-rotor UAVs with signal temporal logic specifications. In *2021 American Control Conference (ACC)*. IEEE, 4209–4216.
[53] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS '77)*. IEEE, 46–57.
[54] Aniruddh G. Puranic, Jyotirmoy V. Deshmukh, and Stefanos Nikolaidis. 2022. Learning performance graphs from demonstrations via task-based evaluations. *IEEE Robotics and Automation Letters* 8, 1 (2022), 336–343.
[55] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2014. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. IEEE, 81–87.
[56] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *the 18th International Conference on Hybrid Systems: Computation and Control*, 239–248.
[57] Alexander Robey, Lars Lindemann, Stephen Tu, and Nikolai Matni. 2021. Learning robust hybrid control barrier functions for uncertain systems. *IFAC-PapersOnLine* 54, 5 (2021), 1–6.
[58] Alëna Rodionova, Lars Lindemann, Manfred Morari, and George J. Pappas. 2022. Combined left and right temporal robustness for control under STL specifications. *IEEE Control Systems Letters* (2022).
[59] Dorsa Sadigh and Ashish Kapoor. 2016. Safe control under uncertainty with probabilistic signal temporal logic. In *Proceedings of Robotics: Science and Systems XII*.
[60] Dorsa Sadigh, Eric S. Kim, Samuel Coogan, S. Shankar Sastry, and Sanjit A. Seshia. 2014. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. IEEE, 1091–1096.

[61] Sadra Sadraddini and Calin Belta. 2015. Robust temporal logic model predictive control. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 772–779.
[62] Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems* 35 (2022), 9460–9471.
[63] Jonathan Sorg, Richard L. Lewis, and Satinder Singh. 2010. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems*, 23.
[64] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[65] Yoshinari Takayama, Kazumune Hashimoto, and Toshiyuki Ohtsuka. 2023. Signal temporal logic meets convex-concave programming: A structure-exploiting SQP algorithm for STL specifications. In *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE, 6855–6862.
[66] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*. Springer, 3–17.
[67] Jana Tůmová, Boyan Yordanov, Calin Belta, Ivana Černá, and Jiří Barnat. 2010. A symbolic approach to controlling piecewise affine systems. In *49th IEEE Conference on decision and control (CDC)*. IEEE, 4230–4235.
[68] George Vachtsevanos, Liang Tang, Graham Drozeski, and Luis Gutierrez. 2005. From mission planning to flight control of unmanned aerial vehicles: Strategies and implementation tools. *Annual Reviews in Control* 29, 1 (2005), 101–115.
[69] Harish Venkataraman, Derya Aksaray, and Peter Seiler. 2020. Tractable reinforcement learning of signal temporal logic objectives. In *Learning for Dynamics and Control*. PMLR, 308–317.
[70] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. 2005. *Algorithmic Learning in a Random World*, Vol. 29. Springer.
[71] Robert D. Windhorst, Todd A. Lauderdale, Alexander V. Sadovsky, James Phillips, and Yung-Cheng Chu. 2021. Strategic and tactical functions in an autonomous air traffic management system. In *AIAA Aviation 2021 Forum*, 2355.
[72] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. 2014. Optimization-based control of nonlinear systems with linear temporal logic specifications. In *the International Conference on Robotics and Automation*, 5319–5325.
[73] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. 2012. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control* 57, 11 (2012), 2817–2830.
[74] Xiangru Xu, Paulo Tabuada, Jessy W. Grizzle, and Aaron D. Ames. 2015. Robustness of control barrier functions for safety critical control. *IFAC-PapersOnLine* 48, 27 (2015), 54–61.
[75] Shakiba Yaghoubi and Georgios Fainekos. 2019. Gray-box adversarial testing for control systems with machine learning components. In *the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 179–184.
[76] Shakiba Yaghoubi and Georgios Fainekos. 2019. Worst-case satisfaction of STL specifications using feedforward neural network controllers: A Lagrange multipliers approach. *ACM Transactions on Embedded Computing Systems* 18, 5S (2019).

Received 15 April 2024; revised 10 July 2024; accepted 13 August 2024