



Tolerance of Reinforcement Learning Controllers Against Deviations in Cyber Physical Systems

Changjian Zhang¹, Parv Kapoor¹, Rômulo Meira-Góes², David Garlan¹, Eunsuk Kang^{1(✉)}, Akila Ganlath³, Shatadal Mishra³, and Nejib Ammar³



¹ Carnegie Mellon University, Pittsburgh, PA, USA

{changjiz,parvk,dg4d,eunsukk}@andrew.cmu.edu

² the Pennsylvania State University, State College, PA, USA

romulo@psu.edu

³ Toyota InfoTech Labs, Mountain View, CA, USA

{akila.ganlath,shatadal.mishra,nejib.ammarr}@toyota.com

Abstract. Cyber-physical systems (CPS) with reinforcement learning (RL)-based controllers are increasingly being deployed in complex physical environments such as autonomous vehicles, the Internet-of-Things (IoT), and smart cities. An important property of a CPS is *tolerance*; i.e., its ability to function safely under possible disturbances and uncertainties in the actual operation. In this paper, we introduce a new, expressive notion of tolerance that describes how well a controller is capable of satisfying a desired system requirement, specified using *Signal Temporal Logic (STL)*, under possible *deviations* in the system. Based on this definition, we propose a novel analysis problem, called the *tolerance falsification problem*, which involves finding small deviations that result in a violation of the given requirement. We present a novel, two-layer simulation-based analysis framework and a novel search heuristic for finding small tolerance violations. To evaluate our approach, we construct a set of benchmark problems where system parameters can be configured to represent different types of uncertainties and disturbances in the system. Our evaluation shows that our falsification approach and heuristic can effectively find small tolerance violations.

1 Introduction

The *tolerance* of a CPS characterizes the ability of an engineered system to function correctly in the presence of uncertainties. Modern *cyber-physical systems (CPS)* operate in dynamic and uncertain environments, such as autonomous vehicles, medical devices, the Internet of Things (IoT), and smart cities. The mission-critical and safety-critical nature of CPS accentuate the need to provide a high level of tolerance against uncertainties, as a failure to do so could result in severe consequences, from safety hazards to economic losses.

C. Zhang, P. Kapoor—Both authors contributed equally to this research.

© The Author(s) 2025

A. Platzer et al. (Eds.): FM 2024, LNCS 14934, pp. 267–285, 2025.

https://doi.org/10.1007/978-3-031-71177-0_17

As CPS grow in complexity and scale, *reinforcement learning (RL)* techniques are gaining popularity for learning CPS controllers. In general, these controllers perceive the state of the CPS and take an action that maximizes the *long-term utility*. The utility is captured through reward functions designed by engineers. An RL controller is trained via a trial-and-error process where an agent takes actions in a simulator of the CPS and uses the simulated results of the actions to discover an optimal control strategy. Hence, the fidelity of the simulator plays a big role in the effectiveness of a trained controller. Often, there are reality gaps between the actual deployed environment and the simulator due to approximation and under-modeling of physical phenomena, which makes controllers trained in simulations perform poorly in the real world [1]. This performance degradation can also manifest as unsafe system behaviors in the actual environment.

To make an RL controller tolerant of possible errors due to these reality gaps, existing works often focus on the training stage, such as robust RL [2,3] and domain randomization [4–6]. They investigate the problem of training a controller that is capable of maintaining desired system behavior in the presence of possible *system deviations*—environmental uncertainties, observation or actuation errors, disturbances, and modeling errors. However, these methods are limited in how desired system behaviors are expressed. In RL, the desired behavior is often expressed using a reward function [2,3]; it is well-known that encoding a high-level system requirement using a reward function is a challenging task that requires a significant amount of domain expertise and manual effort via reward shaping [7,8]. Additionally, certain requirements cannot be directly encoded as rewards, especially those that capture time-varying behavior (e.g., “the vehicle must come to a stop in the next 3 s”).

Due to the limitation in reward functions and the data-driven nature of RL, these training-oriented methods in general do not provide formal guarantees about tolerance. Also, there is a lack of focus on post-training analysis for the tolerance of RL controllers, especially in the sense of maintaining a desired, complex system specification. Moreover, a formal definition of tolerance for RL controllers with respect to system behavior (beyond rewards) is also missing.

To fill the missing gap in post-training tolerance analysis of RL controllers, we propose a new notion of tolerance based on specifications in *Signal Temporal Logic (STL)* [9]. Our definition assumes a *parametric* representation of a system, where *system parameters* capture the dynamics of the system (e.g., acceleration of a nearby vehicle) that are affected by system deviations (e.g., sensor errors). A system is initially assigned a set of nominal parameters that describe its expected dynamics. Then, a change in parameters, denoted by δ , corresponds to a deviation that may occur. Finally, a controller is said to be *tolerable* against certain deviations with respect to a STL specification if and only if the controller is capable of *satisfying* the specification even under those deviations.

Based on this tolerance definition, we propose a new type of analysis problem called the *tolerance falsification*. The goal is to find deviations in system parameters that result in a violation of the desired system specification. Specifically, we argue that identifying a violation closer to the nominal system parameters

would be more valuable, since such a violation is more likely to occur in practice. Intuitively, our system needs to tolerate these deviations before addressing the ones that are further away from the nominal set. These identified violations could be used to retrain the controller for improved tolerance, or to build a run-time monitor to detect when the system deviates into an unsafe region.

In addition, we propose a novel simulation-based framework where the tolerance falsification problem is formulated as a two-layer optimization problem. In the lower layer, for a given system deviation δ (representing a particular system dynamics), an optimization-based method is used to find a falsifying signal; i.e., a sequence of system states that results in a violation of the given STL specification. In the upper layer, the space of possible deviations is explored to find small deviations that result in a specification violation, repeatedly invoking the lower-layer falsification. The results generated from the lower layer guide the upper-layer search towards small violating deviations. Furthermore, we present a novel heuristic that leverages the differences between the trajectories from the normative and deviated environments, captured via cosine distances, to improve the effectiveness of the upper layer search algorithm.

To evaluate the effectiveness of our falsification approach, we have constructed a set of benchmark case studies. In particular, these benchmark systems are configurable with system parameters to generate a range of systems with different behaviors due to the parameters' impact on how the system evolves. Our evaluation shows that our approach can be used to effectively find small deviations that cause a specification violation in these systems.

This paper makes the following contributions:

- We present a novel, formal definition of tolerance for RL controllers (Sect. 4), and a new analysis problem named *tolerance falsification problem* (Sect. 5).
- We propose a two-layer optimization-based algorithm and a novel search heuristic for finding small violating deviations (Sect. 6).
- We present an RL tolerance analysis benchmark and evaluate the effectiveness of our approach through experimental results on it (Sect. 7).

2 Preliminaries

Markov Decision Process. We model the systems under study as discrete-time stochastic systems in *Markov Decision Processes* (MDPs) [10]. An MDP is a tuple $\mathbf{M} = \langle S, A, T, I, R \rangle$, where $S \subseteq \mathbb{R}^n$ is the set of states, $A \subseteq \mathbb{R}^m$ is the set of actions (e.g., control inputs), $T : S \times A \times S \rightarrow [0, 1]$ is the transition function where $T(s, a, s')$ represents the probability from state s to s' by action a and $\forall s \in S, a \in A : \sum_{s' \in S} T(s, a, s') = 1$, $I : S \rightarrow [0, 1]$ is the initial state distribution, and $R : S \rightarrow \mathbb{R}$ is the reward function. As is often the case for real-world systems, we assume that the transition function is unknown.

We consider black-box deterministic control policies for a system. Formally, a policy $\pi : S \rightarrow A$ for an MDP maps states to actions. Reinforcement learning (RL) [11] is the process of learning an optimal policy π^* that maximizes the cumulative discounted reward for this MDP. Additionally, a trajectory σ of an

MDP given an initial state $s_0 \sim I$ and a policy π is defined accordingly as $\sigma = (s_0 \xrightarrow{a_0} s_1 \dots s_i \xrightarrow{a_i} s_{i+1} \dots)$ where $a_i = \pi(s_i)$ and $s_{i+1} \sim T(s_i, a_i)$. Finally, we use $\mathcal{L}(\mathbf{M} || \pi)$ to represent the behavior of the controlled system, i.e., it is the set of all trajectories of a system \mathbf{M} under the control of π .

Signal Temporal Logic. A signal \mathbf{s} is a function $\mathbf{s} : T \rightarrow D$ that maps a time domain $T \subseteq \mathbb{R}_{\geq 0}$ to a k real-value space $D \subseteq \mathbb{R}^k$, where $\mathbf{s}(t) = (v_1, \dots, v_k)$ represents the value of the signal at time t . Then, an STL formula is defined as:

$$\phi := \mu \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \mathcal{U}_{[a,b]} \psi$$

where μ is a predicate of the signal \mathbf{s} at time t in the form of $\mu \equiv \mu(\mathbf{s}(t)) > 0$ and $[a, b]$ is the time interval (or simply I). The *until* operator \mathcal{U} defines that ϕ must be true until ψ becomes true within a time interval $[a, b]$. Two other operators can be derived from *until*: *eventually* ($\Diamond_{[a,b]} \phi := \top \mathcal{U}_{[a,b]} \phi$) and *always* ($\Box_{[a,b]} \phi := \neg\Diamond_{[a,b]} \neg\phi$).

The satisfaction of an STL formula can be measured in a quantitative way as a real-valued function $\rho(\phi, \mathbf{s}, t)$ (also known as the STL *robustness* value), which represents the difference between the actual signal value and the expected one [9]. For example, given a formula $\phi \equiv \mathbf{s}(t) - 3 > 0$, if $\mathbf{s} = 5$ at time t , then the satisfaction of ϕ can be evaluated by $\rho(\phi, \mathbf{s}, t) = \mathbf{s}(t) - 3 = 2$. The definition of ρ is as follows (ρ for the other operators can be formulated from these):

$$\begin{aligned} \rho(\mu, \mathbf{s}, t) &= \mu(\mathbf{s}(t)) & \rho(\neg\phi, \mathbf{s}, t) &= -\rho(\phi, \mathbf{s}, t) \\ \rho(\phi \wedge \psi, \mathbf{s}, t) &= \min\{\rho(\phi, \mathbf{s}, t), \rho(\psi, \mathbf{s}, t)\} \\ \rho(\phi \mathcal{U}_I \psi, \mathbf{s}, t) &= \sup_{t_1 \in I+t} \min\{\rho(\psi, \mathbf{s}, t_1), \inf_{t_2 \in [t, t_1]} \rho(\phi, \mathbf{s}, t_2)\} \end{aligned}$$

3 Motivating Example

We use an RL system which is required to satisfy a safety specification to illustrate our tolerance definition and analysis. Consider the *CarRun* safe RL system implemented in bullet-safety-gym¹, depicted in Fig. 1. The CarRun system has a four-wheeled agent based on *MIT Racecar*² placed between two safety boundaries. The safety boundaries are non-physical bodies that can be breached without causing a collision. The objective is to go through the avenue between the boundaries without penetrating them. The agent velocity also needs to be maintained below a user-defined threshold. Formally, it can be specified by an STL invariant: $\Box(|y_{pos}| < C_1 \wedge |v| < C_2)$, where C_1 and C_2 are the constant thresholds for the y coordinate and the velocity, respectively.

Given the CarRun system, we can train an RL controller such that the car agent satisfies the safety specification above using methods from safe RL [12] [13]. However, to transfer this “safe” controller to the real world, we need to account

¹ <https://github.com/SvenGronauer/Bullet-Safety-Gym>.

² <https://github.com/mit-racecar>.

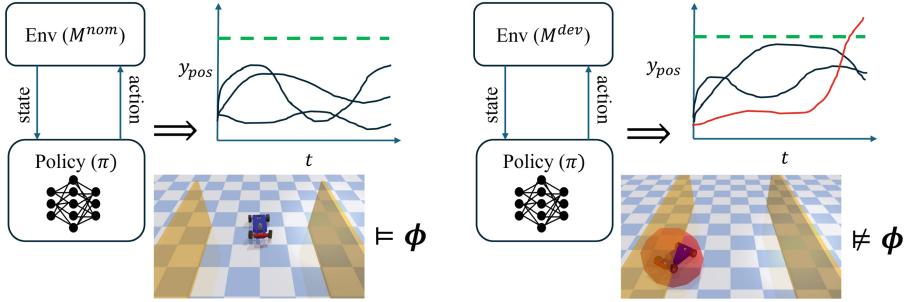


Fig. 1. Behavior of the CarRun system under different system parameters. In the nominal condition (left), y_{pos} in all trajectories is below the threshold (green line) and thus the system is safe. However, in the deviated condition (right), there exists a trajectory where y_{pos} exceeds the threshold and hence the safety requirement is violated.

for the reality gap between the simulator and the deployed environment. This reality gap might arise due to inaccurate modeling of contact surfaces, actuator errors, and incorrect physical parameter configuration (e.g., friction and mass). These reality gaps can lead to the agent violating the safety specification in the real world, despite satisfying them in simulation. Additionally, since the RL controllers are black-box neural networks, it is extremely hard to capture their concrete behaviors. The difficulty in reasoning about the controller's behaviors coupled with the stochasticity of the system leads to a challenging analysis problem of understanding their tolerance ability. This has long been one of the key drawbacks that limit the application of these controllers in the real world [6, 14].

Since it can be challenging to quantitatively measure these reality gaps, we take a parametric approach. We approximate the reality gap between the simulator and the deployed environment quantitatively using deviations as parameters. For example, we model the CarRun system as being *parametric* with two controllable *system parameters*, tm (turn multiplier, a factor for the steering control) and sm (speed multiplier, a factor for the speed control). These parameters govern the impact of the action provided by the controller, e.g., a larger sm will result in more aggressive accelerations. The intuition behind these deviations is to account for actuation issues that arise while deploying agents in the real world. Figure 1 shows the behavior of CarRun under different system parameters. In Fig. 1(a), the agent is deployed in the nominal condition with default system parameters. In this scenario, the controller successfully manages to drive the Car agent through the avenue and also maintains a safe velocity, i.e., the safety specification is satisfied. In Fig. 1(b), we show the same controller deployed under a deviated CarRun environment with different turn and speed multipliers. In this scenario, the controller makes the car behave erratically, which eventually makes the car cross the safety boundary, i.e., the safety specification is violated.

This example highlights the brittleness of these controllers concerning safety specifications and the need for stakeholders to address pre-deployment questions

like: *What are the possible deviations that these RL controllers can tolerate?* More specifically, *how much change in the system parameters can the controller tolerate before it begins to violate the given safety specification?* We formulate this question as a type of analysis problem called *tolerance falsification*, where the goal is to find deviations in system parameters (e.g., the changes in the turn and speed multiplier of CarRun) where the deviated system violates the given specification. This analysis problem is challenging due to the stochastic, black-box nature of the system as well as the opacity of NN-based RL controllers.

Additionally, a notion of “quality of solution” while searching for system parameters is necessary to factor in the practical assumptions about the operating context of this system. For example, deviations that are closer to the nominal parameters are more likely to occur in practice and hence need to be prioritized when analyzing. This helps avoid impractically large deviation values that might cause a violation but offers little insight to system designers. Thus, our falsification process attempts to find violations with *small* deviations; i.e., minimal parameter changes that introduce a risk of specification violation into the system. The output of this analysis (i.e., violations) can help the engineer identify RL-based controller brittleness and can be used to redesign or retrain the controller to improve its tolerance.

4 Tolerance Definition

4.1 Definition of Specification-Based Tolerance

In this work, we use STL to specify the desired properties of a system, and system parameters to capture the deviations in system dynamics. Parameters can represent a variety of deviations such as environmental disturbances (e.g., wind or turbulence), internal deviations (e.g., mass variation of a vehicle), observation errors (e.g., sensor errors), or actuation errors (e.g., errors in steering control). Then, to capture systems with such diverse dynamics using parameters, we leverage the notion of *parametric control systems* [15, 16].

A *parametric* discrete-time stochastic system \mathbf{M}^Δ defines a set of systems such that $\Delta \subseteq \mathbb{R}^k$ represents the parameter domain, and for any $\delta \in \Delta$, an instance of a parametric system \mathbf{M}^δ is an MDP $\mathbf{M}^\delta = \langle S, A, T^\delta, I^\delta, R \rangle$, where the initial state distribution I^δ and the state transition distributions T^δ are both defined by the parameter δ . Parameter δ represents a deviation to a system and Δ represents the domain of all deviations of interest. In addition, we use $\delta_0 \in \Delta$ to represent the zero-deviation point, i.e., the parameter under which the system \mathbf{M}^{δ_0} exhibits the expected, normative behavior. Then, we define a system as being tolerable against a certain deviation as follows:

Definition 1. *For a system \mathbf{M} , a policy π , a deviation parameter δ , and an STL property ϕ , we say the system can tolerate the deviation when the parametric form of \mathbf{M} with δ under the control of π satisfies the property, i.e., $\mathbf{M}^\delta || \pi \models \phi$.*

Then, the tolerance of a controller can be defined as all the possible deviations that the system can tolerate. Formally:

Definition 2. For a system \mathbf{M} , a policy π , and an STL property ϕ , the tolerance of the controller is defined as the **maximal** $\Delta \subseteq \mathbb{R}^k$ s.t. $\forall \delta \in \Delta : \mathbf{M}^\delta \parallel \pi \models \phi$.

In other words, the tolerance of a control policy π is measured by the maximal parameter domain Δ of a system where each deviated system \mathbf{M}^δ of it still satisfies the property under the control of π .

4.2 Strict Evaluation of Tolerance

In this work, we focus on a specific evaluation of tolerance. Specifically, Def. 1 and 2 depend on the interpretation of $\mathbf{M}^\delta \parallel \pi \models \phi$, i.e., a system satisfying a STL property; however, STL satisfaction is computed over a single trajectory. From the literature [17], one common evaluation criteria is that *a system must not contain a trajectory that violates the STL property*. In other words, even in the worst-case scenario that is less likely to occur in a stochastic system, it should still guarantee the property. This interpretation enforces a strong guarantee of the system, and thus we call it the *strict* satisfaction of STL in this work. Formally:

Definition 3. A discrete-time stochastic system \mathbf{M} strictly satisfies an STL property ϕ under the control of a policy π iff every controlled trajectory produces a non-negative STL robustness value, i.e., $\mathbf{M} \parallel \pi \models \phi \Leftrightarrow \forall \sigma \in \mathcal{L}(\mathbf{M} \parallel \pi) : \rho(\phi, \mathbf{s}_\sigma, 0) \geq 0$, where \mathbf{s}_σ is the signal of state values of trajectory σ .

With this interpretation, we can then restate Def. 2 as:

Definition 4. The tolerance of a policy π that strictly satisfies an STL property ϕ is the maximal Δ s.t. $\forall \delta \in \Delta, \sigma \in \mathcal{L}(\mathbf{M}^\delta \parallel \pi) : \rho(\phi, \mathbf{s}_\sigma, 0) \geq 0$

Although this definition delineates a strong tolerance guarantee, it can also be extended to more relaxed notions with probabilistic guarantees. In that case, other evaluation techniques for STL specification satisfaction such as [18–20] can be leveraged. We leave this as an extension of our work in the future.

5 Tolerance Analysis

5.1 Tolerance Falsification

According to Def. 4, to compute the tolerance of a controller, we need to: (1) (formally) show that a stochastic system does not contain a trajectory that violates the STL property, and (2) compute the maximal parameter set Δ , which could be in any non-convex or even non-continuous shape, where all system instances \mathbf{M}^δ should satisfy step (1). This exhaustive computation is intractable due to the black-box RL controllers coupled with the stochasticity in system.

Therefore, in this work, instead of computing or approximating the tolerance Δ , we consider the problem of falsifying a given estimation of tolerance $\widehat{\Delta}$, i.e., finding a deviation $\delta \in \widehat{\Delta}$ that the system cannot tolerate for a given controller.

Problem 1 (Tolerance Falsification). For a system \mathbf{M} , a policy π , and an STL property ϕ , given a tolerance estimation $\widehat{\Delta} \subseteq \mathbb{R}^k$, the goal of a *tolerance falsification problem* $\mathcal{F}(\mathbf{M}, \pi, \phi, \widehat{\Delta})$ is to find a deviation $\delta \in \widehat{\Delta}$ s.t. $\exists \sigma \in \mathcal{L}(\mathbf{M}^\delta || \pi) : \rho(\phi, \mathbf{s}_\sigma, 0) < 0$.

5.2 Minimum Tolerance Falsification

Intuitively, a larger deviation (i.e., a deviation that is far away from the expected system parameter) would likely cause a larger deviation in the system behavior leading to a specification violation. However, controllers are generally not designed to handle arbitrarily large deviations in the first place, and analyzing their performance in these situations offers limited insight to the designer. Moreover, if the designer decides to improve the tolerance of a controller (which is a costly endeavor), deviations closer to the nominal system are given high priority due to their higher likelihood of occurrence. In light of these practical design and deployment assumptions, we focus on the *minimum* deviation problem.

Problem 2. Given a *minimum* tolerance falsification problem $\mathcal{F}_{min}(\mathbf{M}, \pi, \phi, \widehat{\Delta})$, let $\delta_0 \in \widehat{\Delta}$ be the zero-deviation point, the goal is to find a deviation $\delta \in \widehat{\Delta}$ s.t. $\mathbf{M}^\delta || \pi \not\models \phi$ and δ minimizes a distance measure $\|\delta - \delta_0\|_p$.

5.3 Falsification by Optimization

Since the satisfaction of STL can be measured quantitatively, the tolerance falsification problem can be formulated as an optimization problem. Consider a real-valued system evaluation function $\Gamma(\mathbf{M}, \pi, \phi) \in \mathbb{R}$. We assume that if this function's value is negative, the controlled system violates the property, i.e., $\Gamma(\mathbf{M}, \pi, \phi) < 0 \Leftrightarrow \mathbf{S} || \pi \not\models \phi$, and the smaller the value, the larger the degree of property violation. Then, a tolerance falsification problem $\mathcal{F}(\mathbf{M}, \pi, \phi, \widehat{\Delta})$ can be formulated as the following optimization problem:

$$\arg \min_{\delta \in \widehat{\Delta}} \Gamma(\mathbf{M}^\delta, \pi, \phi) \quad (1)$$

i.e., by finding a parameter $\delta \in \widehat{\Delta}$ that minimizes the evaluation function Γ and observing this value can give information about system's property satisfaction. Concretely, if the minimum function value is negative, then the associated parameter δ indicates a deviation where the system violates the property ϕ . Specifically, in the case of strict evaluation of tolerance, the system evaluation function Γ is defined as:

$$\Gamma(\mathbf{M}, \pi, \phi) = \min\{\rho(\phi, \mathbf{s}_\sigma, 0) \mid \sigma \in \mathcal{L}(\mathbf{M} || \pi)\} \quad (2)$$

Note that Eq. 2 is a typical formulation for solving a CPS falsification problem that intends to find a trajectory that violates an STL specification [17].

Finally, we can formulate a minimum tolerance falsification problem $\mathcal{F}_{min}(\mathbf{M}, \pi, \phi, \hat{\Delta})$ as a constrained optimization problem:

$$\arg \min_{\delta \in \hat{\Delta}} \|\delta - \delta_0\|_p \text{ s.t. } \Gamma(\mathbf{M}^\delta, \pi, \phi) < 0 \quad (3)$$

Equation 1 and 3 can both be seen as a bi-level optimization problem [21]; the upper-layer task searches for deviation parameters (δ) and the lower-layer searches for system trajectories. The problem of finding *any* tolerance violation (Eq. 1) can also be formulated as a min-min optimization problem, which can be solved by existing CPS falsifiers such as Breach [22] and PsyTaLiRo [23, 24].

However, the *minimum* falsification problem (Eq. 3) features multi-objective optimization or min-max optimization characteristics—minimizing the deviation distance ($\|\delta - \delta_0\|_p$) would likely cause a larger system evaluation value (Γ). Since these objectives are inherently conflicting, nuanced techniques are required to find solutions. Although, existing CPS falsifiers can be configured to represent this additional cost/objective function (either via specification modification or through explicit cost function definition), the underlying optimization techniques do not have a multi-layer setup to handle this off the shelf. Therefore, we present a novel two-layer search for solving the tolerance falsification problems, particularly effective in finding minimum violating deviations.

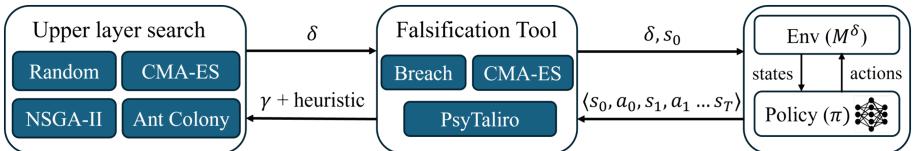


Fig. 2. Overview of the two-layer falsification algorithm.

6 Simulation-Based Tolerance Analysis Framework

In this section, we outline our analysis framework to solve the tolerance falsification problems for black-box CPS and RL controllers (as shown in Fig. 2). We first explain our novel two layer falsification algorithm and then present a heuristic for more effective solving of the minimum falsification problem.

6.1 A Two-Layer Falsification Algorithm

Algorithm 1 describes our two-layer framework. Lines 3–13 indicate the upper-layer search. In each iteration, the upper-layer searches a set of deviation samples. For a deviation δ , it instantiates a deviated system \mathbf{M}^δ (line 6), computes the system evaluation value γ (line 7), and then computes the objective function

Algorithm 1: A Two-Layer Tolerance Falsification Algorithm

```

Input :  $\mathbf{M}, \pi, \phi, \widehat{\Delta}$ , and objective function  $f$ 
Output: violation  $\delta_{best} \in \widehat{\Delta}$ 
1  $\delta_{best} \leftarrow nil$ ;
2  $X \leftarrow$  initial candidates from  $\widehat{\Delta}$  ;
3 while termination criteria = false do
4    $V \leftarrow \langle \rangle$  ;
5   for  $\delta \in X$  do
6      $M^\delta \leftarrow \text{Instantiate}(\mathbf{M}^{\widehat{\Delta}}, \delta)$  ;
7      $\gamma \leftarrow \text{CPSFalsification}(M^\delta, \pi, \phi)$  ;
8      $v \leftarrow f(\delta, \gamma)$  ;                                // heuristic computation.
9      $V \leftarrow V \frown \langle v \rangle$  ;
10  end
11   $\delta_{best} \leftarrow \text{UpdateBest}(X, V)$  ;
12   $X \leftarrow \text{NextCandidates}(f, X, V, \widehat{\Delta})$  ;
13 end

```

value v (line 8). The objective value indicates the quality of a deviation sample, e.g., whether it causes a violation and has a small distance to the zero-deviation point. Finally, the objective values are used to update the best result so far (line 11) and generates the next candidate solutions (line 12). In particular, line 7 indicates the lower-layer task. It corresponds to the system evaluation function Γ (which is the minimal STL robustness value according to Eq. 2).

Given the characteristics of our falsification problem, we propose this two-layer structure for multiple reasons: First, the separation of deviations and the lower-layer CPS falsification allows us to define richer evaluation metrics and heuristics that are solely relevant for deviation searching. These heuristics, if used in a single layer objective, would lead to an ill-posed optimization problem exacerbated by the highly non-convex landscapes of traditional CPS falsification. Second, this separation of concerns allow us to find deviations closer to nominal points even for systems with high-dimensional state spaces, complex dynamics, and rugged robustness landscapes with multiple local minimas. In these settings, an one-layer search would converge to local solutions without exploring the search space extensively. Finally, this two-layer structure provides us enough extensibility to:

- Integrate many off-the-shelf optimization methods for the upper-layer like we have for Uniform Random, CMA-ES [25], NSGA-II [26], and Ant Colony [27].
- Integrate state-of-the-art CPS falsifiers (we integrated CMA-ES, Breach [22], and PsyTaLiRo [24]) and simulation platforms (we used OpenAI-Gym [28], PyBullet [29], and Matlab Simulink).
- Extend to other STL evaluation methods (function Γ), e.g., evaluation with probabilistic guarantees [18–20], cumulative STL [30], or mean STL [31].

6.2 Heuristic for Efficient Minimum Tolerance Falsification

We present a novel heuristic for more effective discovery of minimum violating deviations. Our heuristic is based on the known issues of RL policy overfitting. It has been highlighted in related literature that RL policies can overfit to the specific parameterized system used for training and this dependence can reduce their applicability to real-world scenarios [4–6]. We exploit this over-fitting tendency to guide the search for δ that leads to a violation. Our heuristic is the cosine similarity between a deviated system’s worst-case trajectory and a nominal system’s worst-case trajectory. Formally:

$$dist(\delta) = \frac{\mathbf{Tr}_\delta \cdot \mathbf{Tr}_{\delta_0}}{\|\mathbf{Tr}_\delta\| \cdot \|\mathbf{Tr}_{\delta_0}\|}$$

Specifically, when computing the objective function value v (line 8), we add the similarity value $dist(\delta)$ to the system evaluation value γ . Our intuition is that once a controller has been trained in a system parameterized by δ_0 , it overfits to that specific system. Then, when the controller is deployed in a deviated system, its worst-case trajectory will be similar to the nominal worst-case trajectory if the distance between the two MDPs, measured by the Euclidean distance between the parameters, is small. We measure the similarity between trajectories using cosine similarity. Thus, as the distance from the nominal MDP increases, the similarity score between the worst-case trajectories decreases. This heuristic provides more information about the search space: i.e. in the case there are two deviations where the robustness values are similar (which is possible due to the worst case semantics of STL robustness), cosine similarity can help in directing the search toward more violating directions.

A more in-depth discussion of this heuristic, along with an example, can be found in the extended version of this paper [32].

7 Evaluation

We implemented our proposed framework in a Python package³ and evaluate our technique through comprehensive experimentation. Our evaluation focuses on the minimum tolerance falsification problem. Specifically, we measure our technique’s effectiveness through three key metrics: (1) *the number of violations found*, (2) *the minimum distance of violations*, and (3) *the average distance of violations*. Based on these metrics, we formulate the following research questions:

- **RQ1:** Is our two-layer falsification framework more effective than leveraging an existing CPS falsifier?
- **RQ2:** Does our heuristic improve the effectiveness for finding minimum violating deviations, compared to off-the-self optimization algorithms?

³ <https://github.com/SteveZhangBit/STL-Robustness>.

Although existing CPS falsifiers [22–24] cannot directly solve our minimum tolerance falsification problem (Problem 2), they allow customizing the objective function to optimize for both the deviation distance and STL robustness value to find minimum deviations. We call this technique *one-layer search*. For RQ1, we benchmark against the one-layer search baseline for the minimum tolerance falsification problem. For RQ2, we evaluate whether our proposed heuristic described in Sect. 6.2 further improves the effectiveness of our two-layer search, specifically the minimum distance.

7.1 Experimental Setup and Implementational Details

To answer these research questions, we first present a benchmark with systems and controllers trained to satisfy complex safety specifications. The benchmark contains six systems with non-linear dynamics adopted from OpenAI-Gym, PyBullet, and Matlab Simulink. We extend the interfaces of these systems so that users can configure their behavior for tolerance analysis by changing the system parameters. Details about the benchmark problems can be found in the extended version of this paper [32].

Then, we solve the corresponding minimum tolerance falsification problems for them. For each problem, we conduct the following experiments:

- One-layer search leveraging an existing CPS falsifier by modifying the objective function to factor in the deviation distance and STL robustness value,
- Two-layer search with CMA-ES for both the upper and lower layers,
- Two-layer search with CMA-ES+Heuristic for the upper layer and CMA-ES for the lower layer.

Specifically, for the one-layer search, we employ the state-of-the-art CPS falsifiers, Breach [22] (with CMA-ES) and PsyTaLiRo [24] (with dual annealing). We replace their default objective functions with the sum of the normalized deviation distance and STL robustness value.

For the two-layer search, due to the complexity of the CPS and the non-convex nature of STL robustness, the upper-layer optimization is also non-convex and has multiple local minima. Additionally, we assume black-box systems and controllers. Thus, due to these two considerations, we made the decision to adopt derivative-free evolutionary algorithms. Specifically, we primarily utilized CMA-ES as the upper-layer algorithm because it is widely used for black-box optimization and in our preliminary experiments outperformed other evolutionary methods. However, other algorithms can also be integrated. Furthermore, we also use CMA-ES for the lower-layer search as it is a widely used in CPS falsification tools [17, 22] and works competitively for both Python and Matlab environments. Finally, we implement our heuristic and use it alongside the evaluation function for the upper-layer search.

Each problem was run three times on a Linux machine with a 3.6 GHz CPU and 24 GB memory. For fair evaluation, we set the budget in terms of the number of interactions with the simulator for all our techniques. Specifically, for

one run, the budget for the one-layer search is 10,000 simulations; and the budget for the two-layer search is 100 for the upper-layer and 100 for the lower-layer falsification.

7.2 Results

Table 1 summarizes the results for solving the minimum tolerance falsification problems. The *Viol.* column shows the number of violations found in total from the three runs. The *Min Dst.* and *Avg. Dst.* columns show the minimum and average normalized l_2 distance to the zero-deviation point (i.e., $\|\delta - \delta_0\|_2$) of the found violations, respectively. The performance of our approach heavily depends

Table 1. Minimum tolerance falsification results.

	One-layer search			CMA-ES			CMA-ES w/ Heuristic		
	Viol.	Min Dst.	Avg. Dst.	Viol.	Min Dst.	Avg. Dst.	Viol.	Min Dst.	Avg. Dst.
Cartpole	90	0.300	0.399	69	0.285	0.449	79	0.256	0.417
LunarLander	–	–	–	74	0.026	0.222	84	0.020	0.293
CarCircle	11	0.143	0.255	22	0.102	0.219	57	0.068	0.454
CarRun	25	0.191	0.249	68	0.161	0.449	109	0.156	0.399
ACC	N/A	N/A	N/A	43	0.110	0.323	110	0.138	0.415
WTK	300	0.299	0.443	54	0.296	0.454	45	0.319	0.533

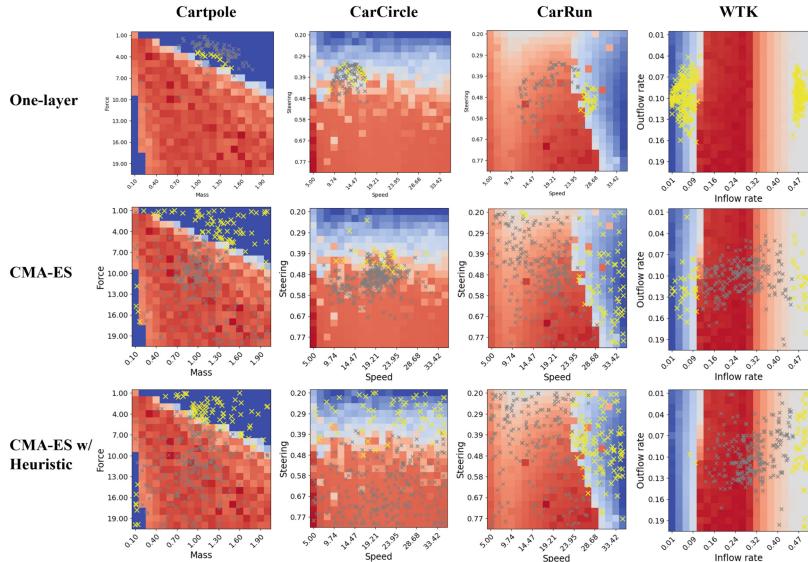


Fig. 3. Search spaces, deviation samples and violations processed by each algorithm. In each graph, the axes are the parameter domains. A red cell is a positive STL robustness value and a blue cell is a negative value. A grey cross is a deviation sample that is not falsified in the given budget; a yellow cross is a violation. (Color figure online)

on the underlying simulation time of a system that vastly outweighs the overhead added by our evolutionary search algorithms. Thus, we share comparable performance, measured by total run time, as tools like Breach and PsyTaLiRo given the same budget of simulation calls.

In addition, to qualitatively exhibit our approach’s effectiveness in finding deviations, we visualize the search space landscape for different problems in heat maps. Each heat map is generated by slicing the space (i.e., the estimated domain of system parameters) into a 20×20 grid and using a CPS falsifier to find the minimum STL robustness value for each grid cell. However, this processing is only done for visualization purposes and is not used in any of the algorithms. This brute force sampling requires far more resources than our falsification approach. Finally, we draw the deviation samples and violations from our analysis on the heat maps. The final results are illustrated visually in Fig. 3.

Answer to RQ1. From the table, the one-layer search fails to find violations in LunarLander, and it cannot represent the type of system parameters we need in ACC (due to falsification tool implementation). On the other hand, our two-layer search with CMA-ES solves all the problems and finds smaller deviations than the one-layer search in all problems. Moreover, from the heat maps, since the distance value is directly added to the STL robustness value in the one-layer search, it fails to find small deviations that *barely* violate the property because it would result in a larger objective value. Thus, it is hard for it to converge to the minimum violating deviations. On the other hand, our two-layer search can better converge to the boundary of safe and unsafe regions. However, it also causes it to find fewer violations because it searches for more samples in the safe region close to the boundary where violations can be rare.

Answer to RQ2. Our two-layer search with CMA-ES+Heuristic finds smaller violating deviations than the original CMA-ES in 4/6 problems. It also finds more violations in 5/6 problems. However, the average distances also increase in 4/6 problems due to more exploration of violations encouraged by our heuristic. Despite that, from the heat maps, our CMA-ES+Heuristic approach can still converge to small violating deviations on the safe and unsafe boundary while also finding more violations. Our heuristic helps in guiding the search and provides additional information to the algorithm when STL robustness is not enough to provide directionality. Concretely, a small similarity value would likely lead to a violation (even when the robustness value is similar) and thus results in more violations found and faster convergence to a small violation.

8 Related Work

There exists similar CPS tolerance notions from a control theory perspective such as [33, 34]. For example, Saoud et al. [33] present a resilience notion of CPS based on LTL w.r.t. a real-valued disturbance space, which forms a ball around a particular trajectory. Then, they present a method to approximate the maximum set of disturbances that maintain a desired LTL property for linear control systems. These notions target traditional controllers with a white-box assumption

of systems and controllers, whereas we employ a black-box assumption which is more practical regarding complex CPS and NN-based RL controllers.

Falsification of CPS [17] is a well-studied problem in the literature. It finds trajectories that violate a STL property by mutating the initial states or system inputs. A related application is parameter synthesis [35] that finds system parameters where the system satisfies the property. It can be seen as a dual problem to the falsification problem. Tools like Breach [22] and PSY-TaLiRo [23, 24] support both types of analysis. However, our tolerance falsification problem can be seen as solving these two problems at the same time. Our upper-layer search finds system parameters that lead to a violation of the system specification, and the lower-layer search finds initial states or system inputs that lead to a violating trajectory. Although our problem can be reduced to a CPS falsification problem with system parameters, it is not effective in solving our minimum tolerance falsification problem compared to our two-layer structure.

A two-layer optimization structure has also been applied in CPS falsification such as in [36–38]. However, these approaches still target traditional falsification of a CPS (the lower-layer in our case), whereas our approach aims to separate the problems of finding small deviation parameters from system falsification.

VerifAI [39, 40] applies a similar idea to us where they consider *abstract features* for a ML model that can lead to a property violation of a CPS. Different from us, they assume a CPS with a ML perception model (such as object detection) connecting to a traditional controller, and the abstract features are environmental parameters that would affect the performance of the ML model (e.g., brightness). In other words, they focus on deviations that affect the ML model whereas our deviation notion is more general that includes any external or internal deviation or sensor error which changes the system dynamics.

Robust RL studies the problem to improve the performance of controllers in the presence of uncertainties [2, 3]. Also, domain randomization [4–6] studies how to train a controller that works across various systems with randomized parameters. However, our work is different in that: (1) we focus on tolerance evaluation whereas they focus more on training; and (2) we focus on system specifications in STL properties, while they rely on rewards where maximizing the reward does not necessarily guarantee certain system specification.

9 Conclusion

In this paper, we have introduced a specification-based tolerance definition for CPS. This definition yields a new type of analysis problem, called *tolerance falsification*, where the goal is to find small changes to the system dynamics that result in a violation of a given STL specification. We have also presented a novel optimization-based approach to solve the problem and evaluated the effectiveness of it over our proposed CPS tolerance analysis benchmark.

Since our analysis framework is extensible, as part of future work, we plan to explore and integrate other types of evaluation functions Γ (e.g., evaluation with probabilistic guarantees [18–20]), different semantics of STL robustness

(e.g., cumulative robustness [30]), or leveraging decomposition of STL for more effective falsification of complex specifications [41]. Moreover, we currently use l -2 norm to compute the deviation distances. In the future, we also plan to explore other distance notions such as Wasserstein Distance [42–44], which computes distribution distance between system dynamics.

Acknowledgements. We are grateful to our anonymous reviewers for their comments and Georgios Fainekos for a discussion on an earlier version of this paper. This research was sponsored by InfoTech Labs, Toyota Motor North America. This work was also supported in part by the NSF awards 2144860 and 2319317, and the NSA grant H98230-23-C-0274. Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the organizations.

Data Availability Statement. The source code of our tool and all the experimental results are available at the following URL: <https://doi.org/10.5281/zenodo.12144853>.

References

1. Collins, J.J., Howard, D., Leitner, J.: Quantifying the reality gap in robotic manipulation tasks. 2019 International Conference on Robotics and Automation (ICRA), pp. 6706–6712, (2018). <https://api.semanticscholar.org/CorpusID:53208962>
2. Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., Peters, J.: Robust reinforcement learning: a review of foundations and recent advances. Machine Learning and Knowledge Extraction, vol. 4, no. 1, pp. 276–315 (2022). <https://www.mdpi.com/2504-4990/4/1/13>
3. Xu, M., et al.: Trustworthy reinforcement learning against intrinsic vulnerabilities: robustness, safety, and generalizability (2022)
4. Peng, X.B., Andrychowicz, M., Zaremba, W., Abbeel, P.: Sim-to-real transfer of robotic control with dynamics randomization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 3803–3810 (2018)
5. Sadeghi, F., Levine, S.: CAD2RL: real single-image flight without a single real image (2017)
6. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 23–30 (2017)
7. Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning, ser. ICML 1999, pp. 278–287. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (1999)
8. Booth, S., Knox, W.B., Shah, J., Niekum, S., Stone, P., Allievi, A.: The perils of trial-and-error reward design: Misedesign through overfitting and invalid task specifications. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, no. 5, pp. 5920–5929 (2023). <https://ojs.aaai.org/index.php/AAAI/article/view/25733>

9. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
10. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model Checking Probabilistic Systems, pp. 963–999. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_28
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press (2018)
12. García, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* **16**(1), 1437–1480 (2015)
13. Gu, S.:et al.: A review of safe reinforcement learning: methods, theory and applications. arXiv, vol. abs/2205.10330, (2022). <https://api.semanticscholar.org/CorpusID:248965265>
14. Yu, W., Liu, C.K., Turk, G.: Policy transfer with strategy optimization. In: International Conference on Learning Representations (2019). <https://openreview.net/forum?id=H1g6osRcFQ>
15. Bhattacharyya, S.P., Chapellat, H., Keel, L.H.: Robust Control: The Parametric Approach, 1st edn. Prentice Hall PTR, USA (1995)
16. Weinmann, A.: Uncertain Models and Robust Control. Springer, Vienna(2012)
17. Corso, A., Moss, R., Koren, M., Lee, R., Kochenderfer, M.: A survey of algorithms for black-box safety validation of cyber-physical systems. *J. Artif. Intell. Res.* **72**, 377–428 (2021)
18. Fan, C., Qin, X., Xia, Y., Zutshi, A., Deshmukh, J.: Statistical verification of autonomous systems using surrogate models and conformal inference (2021)
19. Pedrielli, G., et al.: Part-X: a family of stochastic algorithms for search-based test generation with probabilistic guarantees. *IEEE Trans. Autom. Sci. Eng.* **21**(3), 4504–4525 (2024). <https://doi.org/10.1109/TASE.2023.3297984>
20. Lindemann, L., Matni, N., Pappas, G.J.: STL robustness risk over discrete-time stochastic processes. In: 2021 60th IEEE Conference on Decision and Control (CDC), pp. 1329–1335 (2021)
21. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. *Ann. Oper. Res.* **153**, 235–256 (2007)
22. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
23. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
24. Thibeault, Q., Anderson, J., Chandratre, A., Pedrielli, G., Fainekos, G.: PSY-TaLiRo: a python toolbox for search-based test generation for cyber-physical systems. In: Lluch Lafuente, A., Mavridou, A. (eds.) FMICS 2021. LNCS, vol. 12863, pp. 223–231. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85248-1_15
25. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 312–317 (1996)
26. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)

27. Schlüter, M., Egea, J.A., Banga, J.R.: Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Comput. Oper. Res.* **36**(7), 2217–2229 (2009). <https://www.sciencedirect.com/science/article/pii/S0305054808001524>
28. Brockman, G., et al.: OpenAI gym (2016)
29. Coumans, E., Bai, Y.: Pybullet, a python module for physics simulation for games, robotics and machine learning (2016). <http://pybullet.org>
30. Haghghi, I., Mehdipour, N., Bartocci, E., Belta, C.: Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 4361–4366 (2019)
31. Mehdipour, N., Vasile, C.-I., Belta, C.: Arithmetic-geometric mean robustness for control from signal temporal logic specifications. In: 2019 American Control Conference (ACC), pp. 1690–1695 (2019)
32. Zhang, C., et al.: Tolerance of reinforcement learning controllers against deviations in cyber physical systems (2024). <https://arxiv.org/abs/2406.17066>
33. Saoud, A., Jagtap, P., Soudjani, S.: Temporal logic resilience for cyber-physical systems. In: 2023 62nd IEEE Conference on Decision and Control (CDC), pp. 2066–2071 (2023)
34. Fainekos, G.E., Pappas, G.J.: MTL robust testing and verification for LPV systems. In: 2009 American Control Conference, pp. 3748–3753 (2009)
35. Bartocci, E., et al.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) *Lectures on Runtime Verification*. LNCS, vol. 10457, pp. 135–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
36. Zhang, Z., Ernst, G., Sedwards, S., Arcaini, P., Hasuo, I.: Two-layered falsification of hybrid systems guided by monte Carlo tree search. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **37**(11), 2894–2905 (2018)
37. Zutshi, A., Deshmukh, J.V., Sankaranarayanan, S., Kapinski, J.: Multiple shooting, CEGAR-based falsification for hybrid systems. In: Proceedings of the 14th International Conference on Embedded Software, ser. EMSOFT 2014. New York, NY, USA: Association for Computing Machinery, (2014). <https://doi.org/10.1145/2656045.2656061>
38. Wang, J., Bu, L., Xing, S., Li, X.: PDF: path-oriented, derivative-free approach for safety falsification of nonlinear and nondeterministic CPS. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **41**(2), 238–251 (2022)
39. Dreossi, T., et al.: VERIFAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In: Dillig, I., Tasiran, S. (eds.) *CAV 2019*. LNCS, vol. 11561, pp. 432–442. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_25
40. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. *J. Autom. Reason.* **63**, 1031–1053 (2019)
41. Kapoor, P., Kang, E., Meira-Góes, R.: Safe planning through incremental decomposition of signal temporal logic specifications. *arXiv preprint arXiv:2403.10554* (2024)
42. Lecarpentier, E., Rachelson, E.: Non-stationary Markov decision processes, a worst-case approach using model-based reinforcement learning. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc. (2019)

43. Abdullah, M.A., et al.: Wasserstein robust reinforcement learning (2019)
44. Yang, I.: A convex optimization approach to distributionally robust Markov decision processes with Wasserstein distance. *IEEE Control Syst. Lett.* **1**(1), 164–169 (2017)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

