Robust Testing for Cyber-Physical Systems using Reinforcement Learning

Xin Qin
University of Southern California
Los Angeles, USA
xinqin@usc.edu

Nikos Aréchiga
Toyota Research Institute
Los Altos, USA
nikos.arechiga@tri.global

Jyotirmoy Deshmukh
University of Southern California
Los Angeles, USA
jdeshmuk@usc.edu

Andrew Best
Toyota Research Institute
Los Altos, USA
andrew.best@tri.global

ABSTRACT

In this paper, we propose a testing framework for cyber-physical systems (CPS) that operate in uncertain environments. Testing such CPS applications requires carefully defining the environment to include all possible realistic operating scenarios that the CPS may encounter. Simultaneously, the process of testing hopes to identify operating scenarios in which the system-under-test (SUT) violates its specifications. We present a novel approach of testing based on the use of deep reinforcement learning for *robust testing* of a given SUT. In a robust testing framework, the test generation tool can provide meaningful and challenging tests even when there are small changes to the SUT. Such a method can be quite valuable in incremental design methods where small changes to the design does not necessitate expensive test generation from scratch. We demonstrate the efficacy of our method on three example systems in autonomous driving implemented within a photo-realistic autonomous driving simulator.

ACM Reference Format:
Xin Qin, Nikos Aréchiga, Jyotirmoy Deshmukh, and Andrew Best. 2023. Robust Testing for Cyber-Physical Systems using Reinforcement Learning. In *21st ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '23), September 21–22, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3610579.3611087

1 INTRODUCTION

Autonomous and semi-autonomous cyber-physical systems (CPSs) such as vehicles with advanced driver assist systems (ADAS), unmanned aerial vehicles (UAVs), and medical devices use sophisticated control and planning algorithms to safely accomplish their mission objectives. However, in order to enable autonomous operation in uncertain and previously unseen environments, such CPSs increasingly use learning-enabled components (LECs) for perception and decision-making. There have been many approaches for open-loop testing of LECs (See [10, 21, 31] for excellent surveys on this topic). Of greater relevance to this paper is work on closed-loop testing of learning-enabled CPSs. The closed-loop testing problem seeks to identify environment scenarios under which the CPS behaves in an undesired fashion.

Most techniques for closed-loop testing (including the one presented in this paper) are search-based methods; they can be divided into two classes based on the LEC that is being tested: (1) techniques to test perception components [13, 14, 36], and (2) those to test decision-making/control logic [6, 7, 23, 35]. Irrespective of the LEC being tested, a key challenge for closed-loop testing is appropriately scoping the search problem. If the environment model to generate test scenarios is allowed to be too liberal, falsifying safety conditions of the CPS becomes a trivial exercise. For example, consider the ADAS subsystem of adaptive cruise control (ACC); here, the system-under-test (SUT) or the ego car attempts to maintain a safe following distance from a lead vehicle. If the lead vehicle is allowed to travel backwards on a highway, then it is impossible to design safe ACC logic, and finding SUT violations of safety is trivial. Furthermore, it is important to have an unambiguous mathematical description of the desired behavior of the SUT. In this paper, we address both challenges through the use of the logic-based specification language of Signal Temporal Logic (STL) [26] to express both constraints on the environment as well as safety specifications for the SUT.

At a high level, our approach is similar to input-constrained falsification of STL properties [2, 17]. Falsification of STL properties is a well-studied area with many approaches (see [3, 8] for surveys). More recent work on falsification has focused on the use of *deep reinforcement learning* (RL) for falsifying STL formulas [1][37, 38]. Related work on adaptive stress testing [7] also uses deep RL, but the authors incorporate environment constraints and undesirable behavior by manually encoding them in the reward function used by the deep RL algorithm. However, none of the previous approaches have considered the *robust closed-loop testing problem*.

In closed-loop testing, the emphasis is on identifying not only the environment scenarios that cause the SUT to violate its safety specifications, but also to discover a *test policy* that can react to changes in the SUT. For example, we would like a test generation algorithm to be produce vulnerable scenarios even when there is a change to the initial conditions of the SUT's state variables, changes to the

MEMOCODE '23, September 21–22, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0318-8/23/09.
https://doi.org/10.1145/3610579.3611087

SUT's system dynamics, or minor changes to the LECs. Such an approach is particularly useful in industrial development techniques that rely on the paradigm of continuous integration and pre-merge tests. Here, changes to the software of the CPS should be small, and each should pass a suite of *pre-merge* tests before they are allowed to be merged into the main development branch. More complex tests and analytics may be run nightly or on longer timelines, but pre-merge tests are meant to be lightweight, and need to be able to run quickly to avoid hindering developer productivity. Running a full falsification procedure at pre-merge time is not feasible, and pre-recorded falsification traces are not robust to changes in the SUT.

In this paper, we show that our specific use of STL-based environment constraints and SUT specifications allows us to train *adversarial policies* that are *robust test generators*. In other words, the policy learned by our deep RL algorithm *transfers* to the modified model under certain conditions that characterize the degree of model change. Thus, our procedure has the potential to be invaluable in an incremental design framework where restarting closed-loop testing from scratch after every modification to the LEC may be expensive. Furthermore, the value function induced by the RL policy allows quantifying regions of the state space that are more sensitive to counterexamples, allowing designers to focus on those simulation-based scenarios that are likely to transfer to real-world settings.

In summary, our main contributions are:

(1) We propose a deep reinforcement learning based framework where various sources of uncertainties in the environments are modeled as (one or more) agents that behave according to a *reactive policy* that we train through simulations.
(2) We restrict the agents to respect dynamic constraints (expressed in STL) while causing an ego agent to violate its specification (also expressed in STL).
(3) We formulate an automatic *reward shaping* mechanism that guarantees that the joint behavior of the environment agents and the SUT is such that: the environment constraints are satisfied, while the SUT violates its specifications.
(4) We identify assumptions under which the learned adversarial policies are *robust*. In particular we show that if the learned adversarial policy demonstrates a violation of the SUT specification, then this policy will transfer to agents that (1) start from nearby initial configurations, and (2) have different dynamics than the original ego agent.
(5) We demonstrate the efficacy of our approach on three case-studies from the autonomous driving domain. We show that aspects such as other cars, traffic lights, pedestrians, etc. can be modeled as adversarial agents. We consider (1) an adaptive cruise control example where the leading car is modeled as an adversarial agent, (2) a controller that ensures safety during a lane merge scenario, and (3) a controller that ensures safety during a yellow light scenario.

The rest of the paper is organized as follows. In Section 2 we provide the background and problem definition. We define rewards to be used by our RL-based testing procedure in Sec. 3. We show how the adversarial agents generalize in Section 4, and provide detailed evaluation of our technique in Sec. 5. Finally, we conclude with a discussion on related work in Section 6.

2 PROBLEM STATEMENT AND BACKGROUND

We first introduce the formal description of a multi-agent system as a collection of deterministic dynamical agents.

Definition 2.1 (Deterministic Dynamical Agents). An agent $\mathcal{H}$ is a tuple $(X, A, T, X_{\text{init}}, \pi)$, where X is a set of agent states, A is the set of agent actions, T is a set of transitions of the form $(\mathbf{x}, a, \mathbf{x}')$, where $a \in A$, $X_{\text{init}} \subseteq X$ is a set of designated initial states for the agent, and finally the policy[1] π is a function mapping a state in X to an action in A.

A multi-agent system $\mathcal{S} = \{\text{ego}, \text{ado}_1, \ldots, \text{ado}_k\}$ is a set of agents, with a designated ego agent ego, and a non-empty set of adversarial agents $\text{ado}_1, \ldots, \text{ado}_k$. The state-space of the multi-agent system can be constructed as a product space of the individual agent state spaces, and the set of transitions of the multi-agent system corresponds to the synchronous product of the transitions of individual agents. The transitions of the multi-agent system when projected to individual agents are consistent with individual agent behaviors. A behavior trajectory for an agent is thus a finite or infinite sequence $(t_0, \mathbf{x}_0), (t_1, \mathbf{x}_i), \ldots$, where $\mathbf{x}_i \in X$ and $t_i \in \mathbb{R}^{\geq 0}$. We use s to denote a trajectory variable, i.e. a function mapping t_i to $\mathbf{x}_i$, i.e. $s(t_i) = \mathbf{x}_i$. In many frameworks used for simulating multi-agent systems, it is common to consider timed trajectories with a finite time horizon t_N, and a fixed, discrete time step, $\Delta = t_{i+1} - t_i, \forall i$.

Signal Temporal Logic. Signal Temporal Logic (STL) [27] is a formalism to describe properties of real-valued, dense-time trajectories. STL formulas are evaluated over behavior trajectories. An atomic STL formula is a predicate of the form $f(s) \sim c$, where s is a trajectory variable, f is a real-valued function from X to $\mathbb{R}$, $\sim$ is a comparison operator, i.e. $\sim \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{R}$. STL formulas are constructed recursively using the Boolean logical connectives such as negations ($\neg$) and conjunctions ($\wedge$) and the temporal operator (**U**). It is often convenient to define Boolean connectives such as disjunction ($\vee$), implication ($\Rightarrow$) using the usual equivalences for Boolean logic. It is also convenient to define temporal operators $\mathbf{F}_I\varphi$ as shorthand for $\top \mathbf{U}_I \varphi$, and $\mathbf{G}_I\varphi$ as shorthand for $\neg \mathbf{F}_I \neg \varphi$. Each temporal operator is indexed by the time interval[2] I of the form $[a, b]$, where $a, b \in \mathbb{R}^{\geq 0}$.

STL has both Boolean semantics that recursively define the truth value of the satisfaction of an STL formula in terms of the satisfaction of its subformulas and quantitative semantics that are used to map a trajectory and a formula to a real value known as the robust satisfaction value or simply, the *robustness*. Intuitively, the robustness is proportional to the distance between a given signal s and the set of signals satisfying the formula φ [15]. There are numerous definitions for quantitative semantics of STL, for example [22, 28]. The actual definition to be used is irrelevant to this paper as long as it is efficiently computable. We will assume that the robustness has

[1] Our framework can alternatively include stochastic dynamical agents, where T is defined as a distribution over $(X \times A \times X)$, and the control policy π is a stochastic policy representing a distribution over actions conditioned on the current state of the agent, i.e. $\pi(a \mid \mathbf{x})$. Also, states X and actions A can be finite sets, or can be dense, continuous sets.
[2] Traditional syntax of STL permits intervals that are open on either or both sides; for signals over discrete-time steps, this provision is not required. Furthermore, we exclude intervals that are not bounded above as we intend to evaluate STL formulas on finite time-length traces.

(a) Case Study I: Driving in lane with lead vehicle.

(b) Case Study II: Left vehicle merges in front.

(c) Case Study III: Yellow light running.

Figure 1: Simulation environments for case studies in the CARLA simulator[12].

been clamped to be in an interval $[\rho_{min}, \rho_{max}]$, where $\rho_{max} > 0$ and $\rho_{min} = -\rho_{max}$.

When evaluating an STL formula, each time step requires an application of the functions f that appear in the formula. We denote a signal s satisfying a formula φ at time t by $s, t \models \varphi$, and $s \models \varphi$ is by convention defined as $s, 0 \models \varphi$. In [11, 15], the authors show that $\rho(\varphi, s) \geq 0 \implies s \models \varphi$, and $\rho(\varphi, s) < 0 \implies s \not\models \varphi$. Following the notation in [11], we call the signal variables appearing in the formulas as *primary signals*, and the intermediate results that arise from function applications as *secondary signals*.

Problem Definition: Testing with Dynamically Constrained Adversaries. Given a behavior of the multi-agent system, let the projection of the behavior onto agent $\mathcal{H}$ be denoted by the signal variable $s_{\mathcal{H}}$. Formally, the problem we wish to solve can be stated as follows:

(1) Given a spec ψ_{ego} on the ego agent,
(2) Given a set of constraints φ_{ado_i} on adversarial agent $\mathcal{H}_i$,
(3) Find a multi-agent system policy that can generate behaviors such that: $\forall i : s_{\text{ado}_i} \models \varphi_{\text{ado}\,i} \wedge s_{\text{ego}} \not\models \psi_{\text{ego}}$.

In other words, we aspire to generate a compact representation for a possibly infinite number of counterexamples to the correct operation of the ego agent.

2.1 Adversarial Testing through Policy Synthesis

In contrast to falsification approaches, we assume a deterministic (or stochastic) dynamical agent model for the adversarial agents (as defined in Def. 2.1), i.e. the i^{th} adversarial agent is specified as a tuple of the form $(X_i, A_i, T_i, X_{\text{init}\,i}, \pi_i)$. We assume that initially all agents have a randomly chosen policy π_i. For adversarial agent ado_i, let $\Pi_i = A_i^{X_i}$ denote the set of all possible policies. Let $\Pi_i(\varphi_{\text{ado}}i)$ be the set of policies such that for any $\pi \in \Pi_i(\varphi_{\text{ado}}i)$, using π guarantees that the sequence of states of agent ado_i satisfies $\varphi_{\text{ado}\,i}$. Similarly, let $\Pi_i(\neg\psi_{\text{ego}})$ be the set of policies that guarantees that the sequence of states for the ego agent ego does not satisfy ψ_{ego}. The problem we wish to solve is: for each i, find a policy in $\Pi_i(\neg\psi_{\text{ego}}) \cap \Pi_i(\varphi_{\text{ado}}i)$.

One approach to solve this problem is to use a reactive synthesis approach, when specifications are provided in a logic such as LTL or ATL [4, 9, 25]. There is limited work on reactive synthesis with STL objectives [18, 32], mainly requiring encoding STL constraints as Mixed-Integer Linear Programs; this may suffer from scalability in multi-agent settings. We defer detailed comparison with reactive synthesis approaches to future work. In this paper, we propose using the framework of deep reinforcement learning (RL) for controller synthesis with a procedure for automatically inferring rewards from specifications and constraints.

2.2 Policy synthesis through Reinforcement Learning

Reinforcement learning (RL) [34] and related deep reinforcement learning (DRL) [29] are procedures to train agent policies in deterministic or stochastic environments. In our setting, given a multiagent system $\mathcal{S} = \{\text{ego}, \text{ado}_1, \ldots, \text{ado}_k\}$, we wish to synthesize a policy π_k for each adversarial agent ado_k. We can model multiple adversarial agents as a single agent whose state is an element of the Cartesian product of the state spaces of all agents, i.e., $X = X_{\text{ego}} \times X_{\text{ado}_1} \times \cdots \times X_{\text{ado}_k}$, and the action of this single agent is a tuple of actions of all *adversarial* agents, i.e. $A = A_{\text{ado}_1} \times \ldots \times A_{\text{ado}_k}$.

In each step, we assume that the agent is in state $\mathbf{x} \in X$ and interacts with the environment by taking action $a \in A$. Then, the environment (i.e., the transition relation of the adversarial and the ego agents) picks a next state $\mathbf{x}'$ s.t. $(\mathbf{x}, a, \mathbf{x}') \in T$, and a reward $R(\mathbf{x}, a)$. The reward provides reinforcement for the constrained adversarial behavior, and will be elaborated in Section 3. The goal of the RL agent is to learn a deterministic policy $\pi(\mathbf{x})$, such that the long term payoff of the agent from the initial state (i.e. the discounted sum of all rewards from that state) is maximized. As is common in RL, we define the notion of a value function V in Eq. (1); this is the expected reward over all possible actions that may be taken by the agent. In a deterministic environment, the expectation disappears.

$$V_\pi(\mathbf{x}_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R((\mathbf{x}_{t+k}), a_{t+k}) \,\middle|\, a_{t+k} = \pi(\mathbf{x}_{t+k}) \right] \quad (1)$$

RL algorithms use different strategies to find an *optimal* policy $\pi*$, that for all $\mathbf{x}$ is defined as $\pi^*(\mathbf{x}) = \arg\max_\pi V_\pi(\mathbf{x})$. We assume that the state of the agent at time t_0, i.e. $\mathbf{x}_0$ is in X_{init}.

We now briefly review a classic model-free RL algorithm called Q-learning and summarize two deep RL algorithms: Deep Q-learning and Proximal Policy Optimization (PPO). In Q-learning, we learn a state-action value function $q(\mathbf{x}, a)$, which represents the believed value of taking action a when in state $\mathbf{x}$. Note that $V(\mathbf{x}) = \max_a q(\mathbf{x}, a)$. In Q-learning, the agent maintains a table whose rows correspond to the states of the system and columns correspond to the actions. The entry $q(\mathbf{x}, a)$, encodes an approximation to the state-action value function computed by the algorithm. The table is initialized randomly. At each time step t, the agent uses the table to select an action a_t based on an ε-greedy policy, i.e. it chooses a random action with probability ε, and with probability $1 - \varepsilon$, chooses $\text{argmax}_{a \in A} q(\mathbf{x}, a)$. Next, at time step $t + 1$, the agent observes the reward received R_{t+1} as well as the new state $\mathbf{x}_{t+1}$, and it uses this information to update its beliefs about its previous behavior. During the training process, we sample the initial state of each episode with a probability distribution $\mu(\mathbf{x})$ that is nonzero at all states. After a sufficient number of iterations, all states will eventually be selected as the initial state. We also fix the policies of the adversarial agents to be ε-soft. This means that for each state $\mathbf{x}$ and every action a, $\pi(a|x) \geq \varepsilon$, where $\varepsilon > 0$ is a parameter. Random sampling of initial states and ε-soft policies ensure that the agent explores and avoids converging prematurely to local optima. We assume that during the training process the agent policies are stochastic, epsilon-soft policies, i.e. the policy represents a distribution over a set of actions conditioned on the current state. However, at the end of training, we interpret the policy as deterministic by picking the most probable action for each state.

Deep RL. Deep RL is a family of algorithms that make use of Deep Neural Networks (DNNs) to represent either the value or the policy of an agent. Deep Q-learning [29], is an extension of Q-learning where the table $q(s, a)$ is approximated by a DNN, $q(s, a, w)$, where w are the network parameters. Deep Q-learning observes states and selects actions similarly to Q-learning, but it additionally uses *experience replay*, in which the agent stores previously observed tuples of states, actions, next states, and rewards. At each time step, the agent updates its q-function with the currently observed experience as well as with a batch of experiences sampled randomly from the experience replay buffer[3]. The agent then updates its approximation network by gradient descent on the quadratic loss function $\mathcal{L} = (y_t - q(\mathbf{x}_t, a_t, w))^2$, where $y_t = R_{t+1} + \gamma \max_{a'} q(\mathbf{x}_{t+1}, a, w)$. In the case that $\mathbf{x}_t$ is a terminal state, it is common to assume that all transitions are such that $\mathbf{x}_{t+1} = \mathbf{x}_t$ and $R_t = 0$. Although these learning algorithms learn the state-action value function $q(\mathbf{x}, a)$, in the theoretical exposition that follows, we will use the *state value function* $V(\mathbf{x})$ for simplicity. The optimal state value function can be obtained from the optimal state-action value function by $V^\star(\mathbf{x}) = \max_a q^\star(\mathbf{x}, a)$. PPO [33] is a state-of-the-art *policy gradient* algorithm that performs gradient-based updates on the policy space directly while ensuring that the new policy is not too far from the old policy.

[3]Tabular Q-learning is guaranteed to converge to the optimal value function [34]. On the other hand, DQN may not converge, but it will eventually find a counterexample trace if it exists. In practice, DQN performs well and finds effective value functions, even if its convergence cannot be theoretically guaranteed.

3 LEARNING CONSTRAINED ADVERSARIAL AGENTS

Next, we describe how we construct a reward function that enables training constrained adversarial agents. The reward function needs to encode two aspects: (1) satisfying adversarial constraints, (2) violating ego specification. We assume that adversarial constraints are hierarchically ordered with priorities, inspired by the Responsibility-Sensitive Safety rules for traffic scenarios in [5].

Definition 3.1 (Constrained adversarial reward). Suppose from initial state $\mathbf{x}_0$ the agent has produced a behavior trace $s(\mathbf{x}_0)$ of duration T. We distinguish two cases.

- Case 1: All adversarial constraints are strictly satisfied, i.e. $\rho(c, s(\mathbf{x}_0)) > 0$ for each constraint $c \in \varphi_{\text{ado}}$. In this case, the reward will be the robustness of the adversarial specification.

$$R_t = \begin{cases} 0 & \text{if } t < T \\ \rho(\neg\psi_{\text{ego}}, s(\mathbf{x}_0)) & \text{if } t = T \end{cases} \tag{2}$$

- Case 2: Not all adversarial constraints are strictly satisfied. Let c be the highest priority rule that is not strictly satisfied, i.e. the highest priority rule such that $\rho(c, s(\mathbf{x}_0)) \leq 0$. Then, every constraint with priority higher than c will contribute zero, whereas every constraint with priority less than or equal to c will contribute ρ_{min}. Let M be the number of constraints with priority less than or equal to c.

$$R_t = \begin{cases} 0 & \text{if } t < T \\ -M\rho_{min} & \text{if } t = T \end{cases} \tag{3}$$

The following lemma shows that it is not possible for an adversarial agent to attain a high reward for satisfying lower priority constraints at the expense of higher priority constraints. The proof is straightforward (shown in the appendix).

Lemma 3.2 (Soundness of the reward function). *Consider two traces, s_1 and s_2. Suppose that the highest priority constraint violated by s_1 is c_1 and the highest priority constraint violated by s_2 is c_2. Suppose c_1 has lower priority than c_2. Then, the reward for trajectory 1 will be higher than the reward for trajectory 2, i.e. $R(s_1) > R(s_2)$.*

Proof. Let n_1 be the number of rules with priority less than or equal to c_1. Similarly, let n_2 the number of rules with priority less than or equal to c_2. Then, $R(s_1) = -n_1\rho_{min}$ and $R(s_2) = -n_2\rho_{min}$. Since $n_2 > n_1$ by the assumptions of the lemma, the result follows. □

Pipeline. Our training pipeline is illustrated in Figure 2. Given a scenario composed of interacting agents,

$$(E, R_e), (H_1, R_1), \ldots, (H_n, R_n),$$

the goal is to learn transition distributions $T_1, \ldots, T_n$ and policies $\pi_1, \ldots, \pi_n$ for the adversarial agents such that each adversary satisfies its rules R_k, but the ego is not able to satisfy its rules R_e. The ego agent interacts with several adversarial agents as part of a simulation. The adversarial agents are able to observe the state of the ego as well as of the other adversarial agents, and they may update their policies.

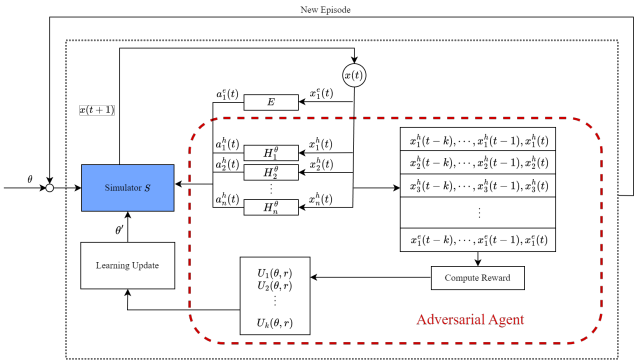


Figure 2: The ego agent E is embedded in a simulation with a collection of adversarial agents H_i^θ, which learn (possibly from a bank of past experience) to stress-test the ego by a particular reward function as derived from the constraints for the adversary and the ego specification.

4 RATIONALE FOR ROBUST TESTING

In this section we show how our RL-based testing approach makes the testing procedure itself robust by learning a closed-loop policy for testing. We first introduce some definitions that help us state the lemmas and theorems about robust testing.

Definition 4.1 (Definition 2 of [11]). Let $f_k : X \to \mathbb{R}$ be any computable function that appears in an STL formula φ. We call the vector of variables in s the *primary* signals of φ, and their images by f *secondary* signals, $\{y_k\}$.

Next, we formalize the notion of distance between signals, and Lemma 4.3 then tells us that if two signals have "nearby values" and also generate nearby secondary signals, then if one of them robustly satisfies an STL formula, the other will also satisfy that formula.

Definition 4.2 (Distance between signals). Given two signals s and s' with identical value domains S and identical time domains $\mathbb{T}$, and a metric d_S on S, the distance between s and s', denoted as $||s - s'||$ is defined as: $\sup_{t\in\mathbb{T}} d_S(s(t), s'(t))$.

Now we are ready to present our theorems about robust testing. First, we show generalizability across initial conditions in two steps: (1) In Theorem 4.4, we assume that the RL algorithm has converged to the optimal value function, and that it has used this value function to find a counterexample trace $s(\mathbf{x}_0)$. We consider a new state $\mathbf{x}'_0$, and want to bound the degradation of the robustness function of the specification $\rho(\neg\psi_{\text{ego}}, s(\mathbf{x}'_0))$. (2) In Theorem 4.5, we *relax* this strong assumption and identify conditions under which generalization can be guaranteed even with approximate convergence.

Lemma 4.3 (Theorem 1 in [11]). *If $\rho(\varphi, s, t) = \delta$, then for every signal s' s.t. every secondary signal satisfies $\left\|y_k - y'_k\right\| < \delta$, the following is true: $(s \models \varphi) \implies (s' \models \varphi)$.*

Theorem 4.4. *Suppose that the adversarial agent has converged to the optimal value function $V^\star(\mathbf{x})$, and that it has found a trace $s(\mathbf{x}_0)$ that falsifies the target specification ψ_{ego} with robustness $\rho(\neg\psi_{\text{ego}}, s(\mathbf{x}_0)) = \tau > 0$ while satisfying all of the adversarial constraints. Given a new state $\mathbf{x}'_0$ such that $\left|V^\star(\mathbf{x}_0) - V^\star(\mathbf{x}'_0)\right| < \delta$ with $\delta < \gamma^T|\rho_{min}|$, the adversary will be able to find a new trajectory $s(\mathbf{x}'_0)$ that satisfies all of the constraints. Furthermore, the robustness of the specification $\neg\psi_{\text{ego}}$ over the new trace will be at least $\tau - \delta/\gamma^T$.*

Proof. We can expand the optimal value function at state $\mathbf{x}_0$ as $V^\star(\mathbf{x}_0) = \sum_{t=0}^{T} \gamma^t R_t$, where R_t is the reward function defined in Definition 3.1. Then, the optimal value function at $\mathbf{x}_0$ is $V^\star(\mathbf{x}_0) = \gamma^T \tau$. Suppose for a contradiction that the new trajectory $s(\mathbf{x}'_0)$ violates some number M of constraints. Then, the following equations show that the two states must actually differ by a large amount, much larger than δ, leading to a contradiction. From Definition 3.1 we have

$$V^\star(\mathbf{x}'_0) = -\gamma^T M \rho_{min}, \tag{4}$$

$$\left|V^\star(\mathbf{x}'_0) - V^\star(\mathbf{x}_0)\right| \geq \gamma^T \tau + M\gamma^T \rho_{min} > \delta, \tag{5}$$

which contradicts the assumption of the theorem. As the constraints will be satisfied by $s(\mathbf{x}'_0)$, their contribution to the value function at $\mathbf{x}'_0$ will be zero. Then, for the 2^{nd} part of the theorem we can expand the optimal value function as:

$$\left|V^\star(\mathbf{x}_0) - V^\star(\mathbf{x}'_0)\right| = \left|\gamma^T \rho(\neg\psi_{\text{ego}}, s(\mathbf{x}_0)) - \gamma^T \rho(\neg\psi_{\text{ego}}, s(\mathbf{x}'_0))\right| \tag{6}$$

By assumption the above terms are $\leq \delta$, which gives us that

$$\left|\rho(\neg\psi_{\text{ego}}, s(\mathbf{x}_0)) - \rho(\neg\psi_{\text{ego}}, s(\mathbf{x}'_0)))\right| \leq \frac{\delta}{\gamma^T} \tag{7}$$

and the theorem follows. □

Note that if δ is chosen as a small enough perturbation such that $\tau - \delta/\gamma^T > 0$, then the new trace is also a trace in which the adversary causes the ego to falsify its specification.

The tabular Q-learning algorithm converges asymptotically to the optimal value function, meaning that for any ϵ, there exists a k such that at the k-th iteration, the estimate V_k differs from the optimal value function by at most α, i.e. $\forall \mathbf{x} \in X$, $\left|V^\star(\mathbf{x}) - V_k(\mathbf{x})\right| < \alpha$. Some RL algorithms have even stronger guarantees. For example, Theorem 2.3 of [16], states that running the value iteration algorithm until iterates of the value function differ by at most $\frac{\alpha(1-\gamma)}{2\gamma}$ produces a value function that converges within α of the value function $\left|V_k(\mathbf{x}_0) - V^\star(\mathbf{x}_0)\right| \leq \alpha$. While useful for theoretical results, value iteration does not scale to problems with large state spaces. The following theorem states that if the RL algorithm has found a value function that is near optimal, the agent will be able to generalize counterexamples across different initial states.

Finally, in Theorem 4.7, we show generalizability when the ego agent dynamics change. The main idea is that if the new dynamics have an ϵ-approximate bisimulation relation to the original dynamics, then we can guarantee generalizability.

Theorem 4.5. *Suppose that we have truncated an RL algorithm at iteration k. Suppose that, from the guarantees of this particular RL algorithm, we are within α of the optimal value function, i.e. for every $\mathbf{x}$, $\left|V_k(\mathbf{x}_0) - V^\star(\mathbf{x}_0)\right| \leq \alpha$. Further suppose that the adversarial agent has found a falsifying trace from state $\mathbf{x}_0$ with robustness τ, i.e. $\rho(\neg\psi_{\text{ego}}, s(\mathbf{x}_0)) = \tau$. Now consider a new state $\mathbf{x}'_0$ such that the degradation of our approximate value function is at most β, i.e.*

$\left|V_k(\mathbf{x}_0) - V_k(\mathbf{x}'_0)\right| \leq \beta$. *Then, the adversarial agent will be able to produce a new trace with robustness at least*

$$\rho(\neg\psi_{\text{ego}}, s(\mathbf{x}'_0), T) \geq \tau - \frac{2\alpha + \beta}{\gamma^T} \tag{8}$$

Proof. Note that by the triangle inequality

$$\alpha + \beta > \left|V_k(\mathbf{x}_0) - V^\star(\mathbf{x}_0)\right| + \left|V^\star(\mathbf{x}'_0) - V_k(\mathbf{x}_0)\right| \tag{9}$$

$$> \left|V_k(\mathbf{x}'_0) - V^\star(\mathbf{x}_0)\right| \tag{10}$$

Further,

$$2\alpha + \beta > \left|V_k(\mathbf{x}'_0) - V^\star(\mathbf{x}_0)\right| > \left|V^\star(\mathbf{x}'_0) - V^\star(\mathbf{x}_0)\right| \tag{11}$$

The result follows from Theorem 4.4 by substituting $\delta = 2\alpha + \beta$. □

Finally, we will show that the agent may cope with limited changes to the multiagent system. This is useful in a testing and development situation, because we would like to be able to reuse a pre-trained adversarial agent to stress-test small modifications of the ego without expensive retraining. To do this, we will define an ϵ-bisimulation relation that will allow us to formally characterize the notion of similarity between different multiagent systems.

Definition 4.6 (ϵ-approximate bisimulation, [19]). Let $\epsilon > 0$, and $\mathcal{S}_1$, $\mathcal{S}_2$ be systems with state spaces X_1, X_2 and transition relations T_1, T_2, respectively. A relation $\mathcal{R}_\epsilon \subseteq X_1 \times X_2$ is called an ϵ-approximate bisimulation relation between T_1 and T_2 if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{R}_\epsilon$,

(1) $d(x_1, x_2) \leq \epsilon$ where d is a distance metric
(2) $\forall a \in A, \forall \mathbf{x}'_1 \in T_1(\mathbf{x}_1, a), \exists \mathbf{x}'_2 \in T_2(\mathbf{x}_2, a)$ such that $(\mathbf{x}'_1, \mathbf{x}'_2) \in \mathcal{R}_\epsilon$
(3) $\forall a \in A, \forall \mathbf{x}'_2 \in T_2(\mathbf{x}_2, a), \exists \mathbf{x}'_1 \in T_1(\mathbf{x}_1, a)$ such that $(\mathbf{x}'_1, \mathbf{x}'_2) \in \mathcal{R}_\epsilon$

Theorem 4.7. *Suppose the adversarial agent has trained to convergence as part of a multiagent system $\mathcal{S}_1$, and it has found a trace that satisfies the adversarial specification with robustness τ. Consider a new multi-agent system $\mathcal{S}_2$ and suppose there exists an ϵ-approximate bisimulation relation between the two systems, including the secondary signals of the formula $\neg\psi_{\text{ego}}$. Further suppose that $\epsilon < \tau$ Then, the trajectory of the new system will also violate the ego specification while respecting the adversarial constraints.*

Proof. If there is an ϵ-approximate simulation between the primary and secondary signals of the traces of the two systems, then for a trace $s_1(\mathbf{x}_0)$ of system $\mathcal{S}_1$ starting from initial state $\mathbf{x}_0$, and a trace $s_2(\mathbf{x})$ of system $\mathcal{S}_2$ also starting from initial state $\mathbf{x}_0$, the ϵ-approximate bisimulation relation ensures that both the primary and secondary signals differ by at most epsilon, i.e. $(|s_1(\mathbf{x}_0) - s_2(\mathbf{x}_0)| \leq \epsilon) \wedge (|y_1 - y_2| \leq \epsilon)$. From Lemma 4.3, s_2 also causes the ego to falsify its specification while satisfying the adversarial specifications. □

5 CASE STUDIES

In this section we first empirically demonstrate the robustness of our adversarial testing procedure. Then, we demonstrate scalability of adversarial testing by applying it to three case studies from the autonomous driving domain.

5.1 Benchmarking Generalizability

We introduce a grid world example environment which consists of an $n \times n$ grid containing the ego agent and the adversarial agent. The objective of the ego agent is to escape adversarial agents, assuming that the game begins with the ego and the adversarial agent at $(c_{\text{ego}}, c_{\text{ado}})$. The ego agent can move k cells in any time step. The ego specification and adversarial constraints are as specified in Section 1.

The ego policy is hand-crafted: it observes the position of the adversarial agents and selects the direction (up, down, left, or right) that maximizes its distance from the adversarial agent. If the target cell lies outside the map, it chooses a fixed direction to move away. It should not be trivial for the adversarial agent to capture the ego agent; thus, for every experiment, we provide a baseline comparison with an adversarial agent that has a randomly chosen policy. A random policy has some likelihood of succeeding from a given initial configuration of the ego and adversarial agents. Thus, the ratio of the number of initial conditions from which the random adversarial agent succeeds to the total number of initial conditions being tested quantifies the *degree of difficulty* for the experiment. We denote the random adversarial agent as ado[rand].

In all the experiments in this section, we train the adversarial agent using our RL-based procedure on a training arena characterized by the vector $\lambda_{\text{train}} = (n_{\text{train}}, C, k_{\text{train}})$, i.e. a fixed grid size $(n_{\text{train}} \times n_{\text{train}})$, a set of initial positions $(c_{\text{ego}}, c_{\text{ado}}) \in C$, and (3) a fixed step size for the ego agent movement (k_{train}). We use Proximal Policy Optimization (PPO)-based deep-RL algorithm to train the adversarial agent[33]. We denote this trained agent as ado$[\lambda_{\text{train}}]$ for brevity. We frame the empirical validation of our robust adversarial testing in terms of the following research questions:

RQ1. How does the performance of ado$[\lambda_{\text{train}}]$ compare against 10 uniformly sampled ado[rand] agents on the same set of initial positions used to train ado$[\lambda_{\text{train}}]$ when all other arena parameters remain the same? [To demonstrate degree of difficulty.]
RQ2. How does the performance of ado$[\lambda_{\text{train}}]$ compare against ado[rand] in an arena of varying map sizes when all other parameters remain the same?
RQ3. How does the performance of ado$[\lambda_{\text{train}}]$ compare against ado[rand] in an arena with varying ego agent step-sizes where all other parameters remain the same?
RQ4. Does the adversarial agent generalize across initial conditions, i.e. if we pick a small subset of the initial conditions to train an adversarial policy, does the policy discover counterexamples on initial states that were not part of the training set?
RQ5. Does the adversarial agent generalize to arenas with ϵ-bisimilar dynamics?

For the first three RQs, we use $\lambda_{\text{train}} = (4, C, 2)$, where C is the set of all possible initial conditions for the agents. Table 1 shows the results for RQ1. Our trained agent easily surpasses the average performance of both 10 and 20 random adversarial agents across all initial locations in the training set C. The average number of violations found by a random adversarial agent is around 10%, while our trained adversarial agent captures the ego within the given time limit from 68% of the initial conditions. Thus, finding an adversarial agent policy that works for a majority of the initial cells is sufficiently difficult. From the results for RQ2, ado$[\lambda_{\text{train}}]$

Experiment	Parameter	Success Initial Condition Rate (%)	
		ado[λ_{train}]	ado[rand]
	Num. ado[rand]		
RQ1	10	67.92 (163/240)	9.83 (237/2400)
	20	67.92 (163/240)	10.23 (491/4800)
	Map Size		
RQ2	2×2	66.67 (8/12)	33.33 (4/12)
	$\mathbf{4 \times 4}$	67.92 (163/240)	2.91(7/240)
	5×5	70.33 (422/600)	1.0 (6/600)
	10×10	9.26 (917/9900)	0.33 (33/9900)
	Ego Step-size		
RQ3	4	72.50 (174/240)	2.92 (7/240)
	3	72.50 (174/240)	2.92 (7/240)
	2	67.92 (163/240)	2.92 (7/240)
	1	67.92 (163/240)	7.08 (17/240)
	δ	Avg. Success Initial Condition Rate	
RQ4	1	0.9735	
	0.5	0.9742	
	0.1	0.9883	
	0.01	1.0	
RQ5	ϵ	Avg. Success Initial Condition Rate	
	3	0.89	
	5	0.78	

Table 1: Empirical demonstration of the robustness of adversarial testing.

successfully causes the ego agent to violate its specification for varying map sizes, even as large as 10×10, though it was trained on a 4×4 map. In contrast a random adversarial agent is rarely successful. From the results for RQ3, we observe that the trained adversarial agent succeeds even against an ego that uses different step sizes than those on which the adversarial agent was trained.

For RQ4, we used $\lambda_{\text{train}} = (10, C_{100}, 1)$, where C_{100} is a set of 100 randomly sampled initial positions (note that the total number of initial configurations is 4950). We found 100 counterexample states during training, of which we chose 5 at random. For each of these states $\mathbf{x}$, we obtained the value of the state as maintained by the PPO algorithm, and found all states $\mathbf{x}'$ s.t. $|V(\mathbf{x}) - V(\mathbf{x}')| < \delta$. We computed the fraction of these states that also led to counterexamples. For four of the identified counterexample states, δ satisfied the conditions outlined in Theorem 4.4, and the results are shown in Table 1. As expected, smaller the value of δ, higher is the number of failing states with nearby values.

For RQ5, we used $\lambda_{\text{train}} = (10, C_{200}, 1)$, where C_{200} was a randomly chosen set of 200 initial states. We defined a refinement of the map, basically an $\epsilon \times \epsilon$ grid was imposed on each grid cell of the original map. We considered an adversarial policy that basically used the same action as that of the original coarser grid cell, while the ego agent used a refined policy. We can establish that the resulting transition system is actually ϵ-bisimilar to the original transition system. For different values of ϵ, we picked 3 sets of 300 random initial states and tested if they led to counterexamples. The average success rates are shown in Table 1. We see that an abstract adversarial policy can violate the ego spec surprisingly often.

5.2 Autonomous Driving Case studies

We apply our adversarial testing framework to three case studies from the autonomous driving domain[4]. We used the Carla driving simulator [12] as a means to stress-test a controller driving a car in three different scenarios, (1) freeway driving on a straight lane, (2) freeway driving with a car merging into the ego lane, (3) driving through a yellow light. The adversarial agent was developed in `python`, and the neural networks used in the DQN examples were developed in `pytorch` [30]. Figure 7 illustrates the main implementation steps of our testing framework. Since CARLA cannot make a car travel directly at a given speed. We let the ego and adversarial car accelerate until it reaches the target speed according to the initial condition, then teleport it to the distance specified by the initial condition, and then start training.

Adaptive Cruise Control. In this experiment, two vehicles are driving in a single lane on a highway. The lead vehicle is the adversarial agent. The follower vehicle avoids colliding into the leader using an adaptive cruise controller (ACC). The purpose of adversarial testing is to find robust adversarial policies that cause the ACC system to collide with the adversarial agent. The ACC controller modulates the throttle (α) by observing the distance (d) to the lead vehicle and attempting to maintain a minimum safe following distance d_{safe}. The ACC controller is a Proportional-Derivative (PD) controller with saturation. The PD term u is equal to $K_p(d - d_{\text{safe}}) + K_d(v_{\text{ado}} - v_{\text{ego}})$, and the controller action α is defined as $\alpha_{\max}$ if $u > \alpha_{\max}$, $\alpha_{\min}$ if $u < \alpha_{\min}$, and u otherwise. The ego specification is given in Eq. (12). Here, T is the maximum duration of a simulation episode and d_{safe} is the minimum safe following distance. The adversarial agent should cause the ego to violate its spec in less than T seconds. The adversarial constraint specifies that it should not exceed the speed limit v_{lim} and that it should maintain a minimum speed $v_{\min}$. For our experiment, we choose $v_{\min} = 0.1$, and $d_{\text{safe}} = 4.7$m. The distance d is computed between the two front bumpers. This represents a car length of $4.54m$, plus a small safety margin. The state of the adversarial agent is the tuple $d, v_{\text{ego}}, v_{\text{ado}}$. At each time step, the adversarial agent chooses an acceleration from a discretized space which contains 3 possible actions. In this experiment, we explore two different RL algorithms: Q-learning and a DQN algorithm with replay buffers [29]. The average runtime using the DQN (1.93 hours) is less than that using a Q-table (4.83 hours) and gives comparable success rates: 54.8% for the DQN agent vs. 55.79% for the agent using Q-tables. The average time to run a single episode is between 29 and 30 seconds. Fig. 4 shows 3 episodes from the same initial position for the ego and adversarial vehicles where initially the adversarial agent is not able to find an adversarial behavior, then in the later two episodes the adversarial agent is able to cause the ego to collide with it.

Lane Change Maneuvers. In this experiment, 2 vehicles are driving on a two-lane highway. The ego vehicle is controlled by a switching controller that alternates between cruising and avoiding

[4]We provide one more case study in the appendix.

Case Study	Description	STL Formula	
ACC	Ego: Avoid collision	$\mathbf{G}_{[0,T]}(d \geq d_{\text{safe}})$	(12)
	Adversarial agent: Velocity bounds	$\mathbf{G}(v_{\min} \leq v_{\text{ado}} \leq v_{\lim})$	(13)
Lane Change	Ego: Avoid collision	$\mathbf{G}_{[0,T]}(d_2 \geq d_{\text{safe}})$	(14)
	Adversarial agent: Init. pos.	$d_{\text{long}} > d_{\text{safe}}$	(15)
Yellow Light	Ego: Don't run red light	$\neg\mathbf{F}_{[0,T]}(\ell_R \wedge d_{\ell,\text{ego}} \in [-\delta, 0])$	(16)
	Adversarial agent: Speed Limits	$\mathbf{G}_{[0,T]}(v_{\text{ado}} < v_{\lim})$	(17)
	Adversarial agent: Don't run red light	$\neg\mathbf{F}_{[0,T]}(\ell_R \wedge d_{\ell,\text{ado}} \in [-\delta, 0])$	(18)

Table 2: Ego Specifications and Adversarial Rules for case studies

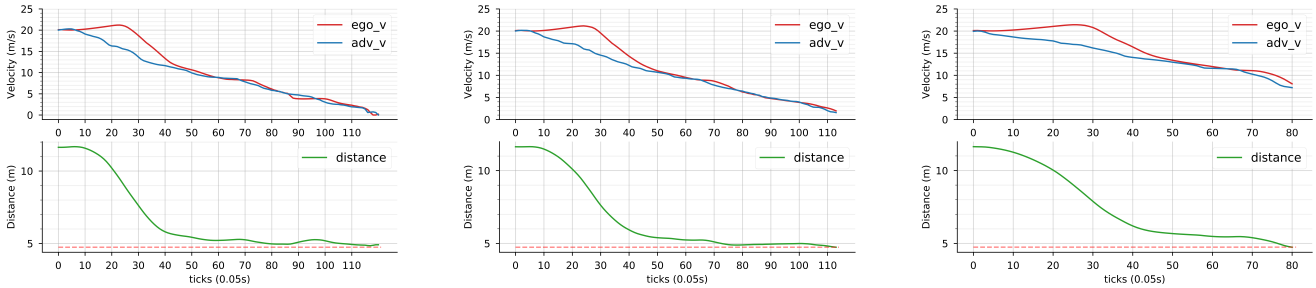


Figure 4: Traces in (a) show an early episode in the driving in the lane case study. The adversary is unable to cause a collision, and the distance between the ego vehicle and the adversarial vehicle remains above the collision threshold for the duration of the episode. Traces in (b) show a later episode in which the adversary successfully causes a collision. Traces in (c) show a different behavior that the adversary successfully learned to cause a collision.

a collision by applying a "hard" brake. The ego controller predicts future adversarial agent positions based on the current state using a look-ahead distance $d_{\text{lka}} = d_{\text{lat}} - v_{\text{ado,lat}} t_{\text{lka}}$, where d_{lat} is the lateral distance between the vehicles, $v_{\text{ado,lat}}$ is the adversarial agent's lateral velocity and t_{lka} is a fixed look-ahead time. Based on d_{lka}, it switches between two control policies: if $d_{\text{lka}} > d_{\text{safe}}$, then it continues to cruise, but if $d_{\text{lka}} \leq d_{\text{safe}}$, it applies the brakes. The adversarial agent is in the left lane and attempts to merge to the right in a way that causes the ego to collide with it. We add a constraint to ensure that the adversarial agent should always be longitudinally in front of the ego car when it tries to merge as specified in Eq. (15); here d_{long} is the longitudinal distance between the cars. Without this constraint, the adversarial agent can always induce a sideways-crash. The ego spec is given in Eq. (14). Here, $d_2 = \sqrt{d_{\text{long}}^2 + d_{\text{lat}}^2}$ is the Euclidean distance between the two cars.

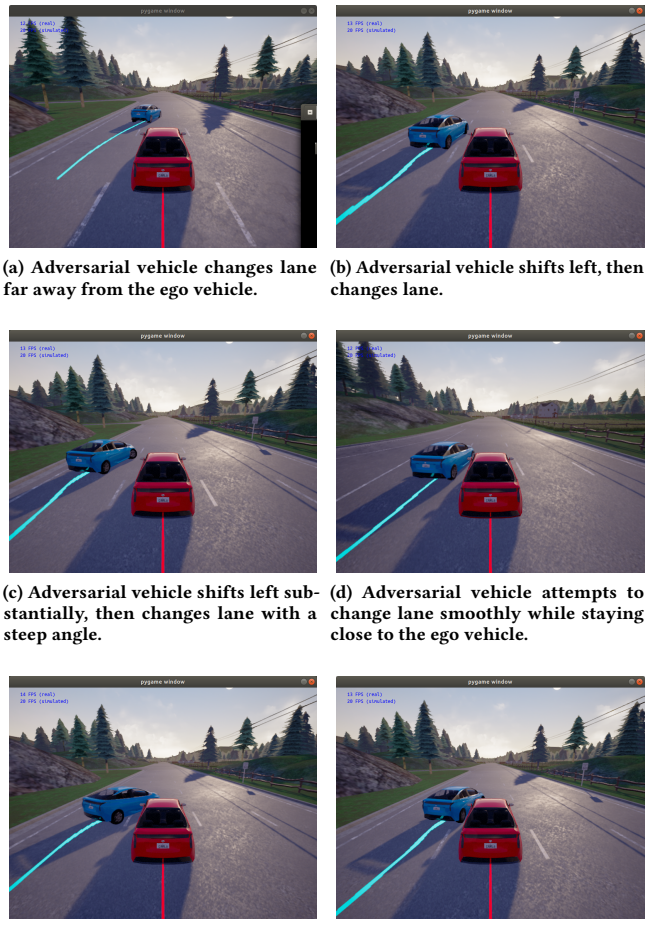
In the course of training, we observe that the behavior of the adversarial agent improves with time. Training for 106 episodes requires 2.53 hours and gives us a success rate of 71%, i.e. 71% of the episodes lead to a collision. With 206 episodes, the success rate improves to 72.35% but requires 4.71 hours of runtime. After 371 episodes, the success rate improves to 75.76% after 9.44 hours. This experiment demonstrates that even with a relatively small time budget, the constrained RL agent can learn a policy that induces failures with high probability.

Generalizability. In both case studies, we observed generalizability of the adversarial policy to different initial conditions. In the ACC case study, we found several initial states within $\delta = 6.5 \times 10^{-6}$ that were also counterexample states. Overall the states have smaller values as the episode lengths are longer and the γ^T term causes values to be small. We observed that for the failing initial state $(v_{\text{ego}} \mapsto 12, v_{\text{ado}} \mapsto 12, d \mapsto 15)$, we found failing initial states with values of both v_{ado} and d that were both smaller and larger than those in the original initial state. However, some failing initial states did not have states with nearby values that were violating. This can be attributed to the fact that the RL algorithm may not have converged to a value close to optimal. For the second case study, we found that the failing initial condition $(v_{\text{ego}} \mapsto 12, v_{\text{ado}} \mapsto 12, d_2 \mapsto 16)$ has several nearby failing initial states with a small value of δ that were not previously encountered during training.

Table 3 demonstrates the generalization statistics for the ACC case study. The leftmost column, β, is a degradation of the RL value function that we wish to consider. Given this degradation of the value function, the *bound* column is the bound predicted by Theorem 4.5. The values of the table are computed by first taking an initial condition x_0 that produced a counterexample and then sampling multiple new initial conditions x_0' so that $|V(x_0) - V(x_0')| \leq \beta$. The column labeled *num* denotes the number of such initial conditions that are sampled. Then, we run a simulation from

β	bound	μ_ρ	σ_ρ	num	success init	failure init
0.1	0.03367	0.1287	0.0064	1225	1192 (97.3%)	33
0.2	-0.08860	0.09965	0.0074	1633	1565 (95.8%)	68
0.5	-0.45539	0.04782	0.0254	1883	1565 (83.1%)	318
1	-1.0667	-0.2932	0.1973	3159	1565 (49.5%)	1594
2	-2.2893	-0.4013	0.2678	3571	1565 (43.8%)	2006

Table 3: Demonstration of Theorem 4.5. In all cases, the mean robustness degradation is bounded below as predicted by the theorem. Initial conditions with small value function degradation β are more likely to yield counterexamples, as predicted by the theory. The column success init and failure init shows the number of initial conditions that leads to a successful falsifying case or a non-successful falsifying case.



(a) Adversarial vehicle changes lane far away from the ego vehicle.

(b) Adversarial vehicle shifts left, then changes lane.

(c) Adversarial vehicle shifts left substantially, then changes lane with a steep angle.

(d) Adversarial vehicle attempts to change lane smoothly while staying close to the ego vehicle.

(e) The adversarial vehicle changes lane aggressively and hits the ego vehicle, violating traffic rules.

(f) Adversary changes lane and induces a crash without breaking the traffic rules

Figure 5: Adversarial vehicle behaviors across episodes in the lane change maneuvers case study.

x_0' with a frozen version of the adversarial agent, i.e. one that is not learning anymore. The column labeled μ_ρ denotes the mean robustness of the simulation traces, and σ_ρ denotes the standard deviation of such robustnesses. We note that in all cases, the mean robustness is larger than the lower bound predicted by the theorem, and that the table demonstrates that there is a high density of counterexamples where the value function degradation is smaller, and fewer counterexamples where the value function degradation is larger. This demonstrates that the adversarial agent has learned a generalizable policy, which correctly reflects the landscape of initial conditions that lead to counterexamples. In this table, the episode length is $T = 20$ and the discount factor is $\gamma = 0.99$.

Yellow Light. In this experiment, the ego vehicle is approaching a yellow traffic light led by an adversarial vehicle. Let the signed distance of the ego, adversarial vehicle from the light be respectively $d_{\ell,\text{ego}}$, $d_{\ell,\text{ado}}$, and Boolean variables ℓ_Y and ℓ_R be true if the light is respectively yellow and red. We use the convention that $d_{\ell,\text{ego}} > -\delta$ if the ego vehicle is approaching the light and $d_{\ell,\text{ego}} < -\delta$ if it has passed the light (resp. for adversarial vehicle). By traffic rules, a vehicle is expected to stop δ meters away from the traffic light (e.g. δ could be the width of the intersection being controlled by the light), I.e. the vehicle should stop at 0. Thus, if $d_{\ell,\text{ego}} \in [-\delta, 0]$ when the light turns red, it has run the red light. This ego specification is shown in Eq. (16). The traffic light is modeled as a non-adversarial agent, it merely changes its state based on a predetermined schedule. The goal of the adversarial vehicle is to make the ego vehicle run the red light. The rule-based constraints on the adversarial vehicle are that it may not drive backwards and it may not run the red light (shown in Eqs. (17),(18) resp.). The state of the adversarial agent includes the speed of both vehicles, and relative distance between the vehicles. At the start of an episode, $d_{\ell,\text{ado}} = 30$, and ℓ_Y is true, and ℓ_R becomes true after $\tau = 2$ seconds.

The ego controller is a switched mode controller that either uses an ACC controller or applies the maximum available deceleration $a_{\text{ego,max}}$. At time $t < \tau$, let $d(t)$ denote the distance required for the ego vehicle to come to a stop before the light turns red by applying $a_{\text{ego,max}}$. We can calculate $d(t)$ as $v_{\text{ego}} \cdot (\tau - t) + 0.5 \cdot a_{\text{ego,max}} \cdot (\tau - t)^2$. Then, at time t, the ego controller chooses to cruise if $d(t) + \delta < d_{\ell,\text{ego}}$, and brakes otherwise.

Figure 6 shows that the ego vehicle maintains an appropriate distance to the lead car, but that it starts decelerating too late and is thus caught in the intersection when light turns red. The adversarial vehicle successfully clears the intersection while the light is still yellow, consistent with its constraints. The adversarial agent we train uses the DQN RL algorithm. After 162 episodes of training, approximately 60% of its episodes find a violation of the ego specification with a runtime of 1.88 hours. After 247 episodes, the success rate increases to 64% with a runtime of 2.59 hours. This case study

demonstrates that our adversarial testing procedure succeeds even in the presence of multiple adversarial rules and an interesting ego specification.

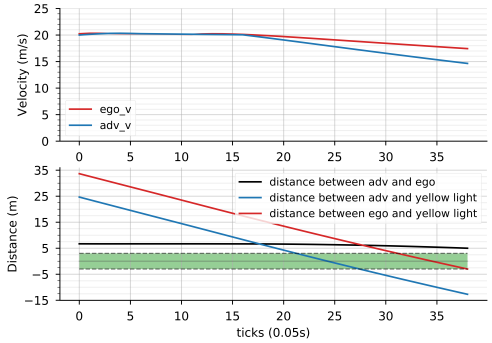


Figure 6: Yellow light case study. The green region represents the region in which the ego vehicle will run the yellow light. The adversary learns to drive the ego car into the target region.

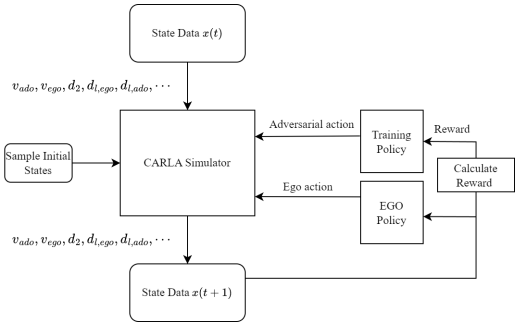


Figure 7: An illustration of our implementation structure

6 RELATED WORK AND CONCLUSIONS

Adaptive Stress Testing. The work of [6, 23] is closely related to our work. In this work, the authors also use deep RL (and related Monte Carlo Tree search) algorithms to seek behaviors of the vehicle under test that are failure scenarios. There a few key differences in our approach. In [23], reward functions (that encode failure scenarios) are hand-crafted and require manual insight to make sure that the RL algorithms converge to behaviors that are failure scenarios. Furthermore, the constraints on the adversarial environment are also explicitly specified. The approach in [6] uses a subset of RSS (Responsibility-Sensitive Safety) rules that are used to augment hand-crafted rewards to encode failure scenarios by the ego and responsible behavior by other agents in a scenario. In specifying STL constraints, we remove the step of manually crafting rewards. And the robustness value of STL makes our theorem possible.

Falsification. There is extensive related work in falsification of cyber-physical system. Most falsification techniques use fixed finitary parameterization of system input signals to define a finite-dimensional search space, and use global optimizers to search for parameter values that lead to violation of the system specification. A detailed survey of falsification techniques can be found in [8]. A control-theoretic view of falsification tools is that they learn open-loop adversarial policies for falsifying a given ego model while our approach focuses on closed-loop policies.

Falsification using RL. Also close to our work are recent approaches to use RL [38] and deep RL [1] for falsification. The key focus in work [38] is on solving the problem of automatically scaling quantitative semantics for predicates and effective handling of Boolean connectives in an STL formula. The work in [1] focuses on a smooth approximation of the robustness of STL and thoroughly benchmarks the use of different deep RL solvers for falsification.

Comparison. Compared to previous approaches, the focus of our paper is on reusability of dynamically constrained adversarial agents trained using RL techniques. We identify conditions under which a trained adversarial policy is applicable to a system with a different initial condition or different dynamics *with no retraining*. This can be of immense value in an incremental design and verification approach. The other main contribution is that instead of using a monolithic falsifier, our technique packs *multiple, dynamically constrained* falsification engines as separate agents; dynamic constraints allow us to specify hierarchical traffic rules. Also, previous approaches for falsification do not consider dynamic constraints on the environment at all, only simple bounds on the parameter space. Finally, in our approach, both specifications and constraints are combined into a single reward function which can then utilize off-the-shelf deep RL algorithms. In comparison to [1, 20, 24, 38], our encoding of STL formulas into reward functions is simplistic as it is not the main focus of this paper; we defer extensions that consider nuanced encoding of STL constraints to future work. The emphasis in [1] is to use the training process (which includes exploration) to find a (possibly non-robust) single falsifying behavior. Work [39] use fuzzing algorithms to find multiple scenarios that cause the ego to fail with coverage measures, whereas, in our work we focus on training the RL agent to obtain a robust *falsifying policy*.

Conclusions. Our work addresses the problem of automatically performing constrained stress-testing of cyberphysical systems. We use STL to specify the target against which we are testing and constraints that specify reasonableness of the testing regime. We are using STL as a lightweight, high-level programming language to loosely specify the desired behaviors of a test scenario, and leveraging RL algorithms to determine how to execute those behaviors. The learned adversarial policies are reactive, as opposed to testing schemes that rely on merely replaying pre-recorded behaviors, and under limited conditions can even provide valuable testing capability to modified versions of the system.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their feedback. The National Science Foundation supported this work through the following grants: CAREER award (SHF-2048094), CNS-1932620, CNS-2039087, FMitF-1837131, CCF-SHF-1932620, the Airbus Institute for Engineering Research, and funding by Toyota R&D and Siemens Corporate Research through the USC Center for Autonomy and AI.

REFERENCES

[1] Takumi Akazaki, Shuang Liu, Yoriyuki Yamagata, Yihai Duan, and Jianye Hao. 2018. Falsification of cyber-physical systems using deep reinforcement learning. In *International Symposium on Formal Methods*. Springer, 456–465.

[2] Ezio Bartocci, Roderick Bloem, Benedikt Maderbacher, Niveditha Manjunath, and Dejan Ničković. 2021. Adaptive testing for specification coverage in CPS models. *IFAC-PapersOnLine* 54, 5 (2021), 229–234.

[3] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. 2018. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. *Lectures on Runtime Verification: Introductory and Advanced Topics* (2018), 135–175.

[4] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. 2018. Graph games and reactive synthesis. In *Handbook of Model Checking*. Springer, 921–962.

[5] Andrea Censi, Konstantin Slutsky, Tichakorn Wongpiromsarn, Dmitry Yershov, Scott Pendleton, James Fu, and Emilio Frazzoli. 2019. Liability, Ethics, and Culture-Aware Behavior Specification using Rulebooks. In *ICRA*.

[6] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J Kochenderfer. 2019. Adaptive stress testing with reward augmentation for autonomous vehicle validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 163–168.

[7] Anthony Corso, Robert Moss, Mark Koren, Ritchie Lee, and Mykel Kochenderfer. 2021. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research* 72 (2021), 377–428.

[8] Jyotirmoy V Deshmukh and Sriram Sankaranarayanan. 2019. Formal techniques for verification and testing of cyber-physical systems. In *Design Automation of Cyber-Physical Systems*. Springer, 69–105.

[9] Rayna Dimitrova and Rupak Majumdar. 2014. Deductive control synthesis for alternating-time logics. In *2014 International Conference on Embedded Software (EMSOFT)*. IEEE, 1–10.

[10] Wenhao Ding, Chejian Xu, Mansur Arief, Haohong Lin, Bo Li, and Ding Zhao. 2023. A survey on safety-critical driving scenario generation—A methodological perspective. *IEEE Transactions on Intelligent Transportation Systems* (2023).

[11] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems*. Vol. 6246. Springer Berlin Heidelberg, Berlin, Heidelberg, 92–106.

[12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[13] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. 2019. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning* 63 (2019), 1031–1053.

[14] Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. 2019. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*. Springer, 432–442.

[15] G. E. Fainekos and G. J. Pappas. 2009. Robustness of Temporal Logic Specifications for Continuous-time signals. *Theoretical Computer Science* (2009).

[16] Norm Ferns, Prakash Panangaden, and Doina Precup. 2011. Bisimulation Metrics for Continuous Markov Decision Processes. *SIAM J. Comput.* 40, 6 (2011), 1662–1714.

[17] Thomas Ferrère, Dejan Nickovic, Alexandre Donzé, Hisahiro Ito, and James Kapinski. 2019. Interface-aware signal temporal logic. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 57–66.

[18] Shromona Ghosh, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L Sangiovanni-Vincentelli, S Shankar Sastry, and Sanjit A Seshia. 2016. Diagnosis and repair for synthesis from signal temporal logic specifications. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. 31–40.

[19] Antoine Girard and George J. Pappas. 2011. Approximate Bisimulation: A Bridge Between Computer Science and Control Theory. *European Journal of Control* 17, 5 (Jan. 2011), 568–578. https://doi.org/10.3166/ejc.17.568-578

[20] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. 2020. Reward Shaping for Reinforcement Learning with Omega-Regular Objectives. arXiv:2001.05977 [cs.LO]

[21] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.

[22] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Ničković. 2018. Quantitative Monitoring of STL with Edit Distance. *Formal Methods in System Design* 53, 1 (Aug. 2018), 83–112. https://doi.org/10.1007/s10703-018-0319-x

[23] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. 2018. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1–7.

[24] Karen Leung, Nikos Aréchiga, and Marco Pavone. 2019. Backpropagation for parametric STL. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 185–192.

[25] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M Murray. 2013. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. Automat. Control* 58, 7 (2013), 1771–1785.

[26] O. Maler and D. Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems* 3253 (2004), 152–166.

[27] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *FORMATS*. Springer, 152–166.

[28] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. In *NIPS*. http://arxiv.org/abs/1312.5602 arXiv: 1312.5602.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

[31] Nijat Rajabli, Francesco Flammini, Roberto Nardone, and Valeria Vittorini. 2020. Software verification and validation of safe autonomous cars: A systematic literature review. *IEEE Access* 9 (2020), 4797–4819.

[32] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*. 239–248.

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

[34] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement learning: an introduction* (second edition ed.). The MIT Press, Cambridge, MA.

[35] Cumhur Erkan Tuncali and Georgios Fainekos. 2019. Rapidly-exploring random trees-based test generation for autonomous vehicles. *arXiv preprint arXiv:1903.10629* (2019).

[36] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1555–1562.

[37] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. 2020. Hybrid System Falsification Under (In)Equality Constraints via Search Space Transformation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[38] Zhenya Zhang, Ichiro Hasuo, and Paolo Arcaini. 2019. Multi-Armed Bandits for Boolean Connectives in Hybrid System Falsification. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 401–420.

[39] Yuan Zhou, Yang Sun, Yun Tang, Yuqi Chen, Jun Sun, Christopher M Poskitt, Yang Liu, and Zijiang Yang. 2023. Specification-based Autonomous Driving System Testing. *IEEE Transactions on Software Engineering* (2023).