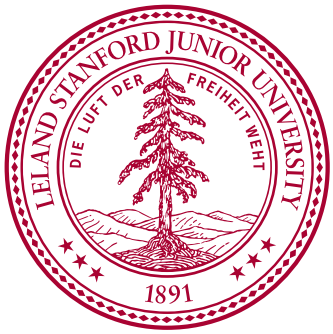
Stanford Center for AI Safety
Stanford University



Stanford Intelligent Systems Laboratory (SISL)
Navigation and Autonomous Vehicles Laboratory (NAV Lab)
Autonomous Systems Laboratory (ASL)

Department of Aeronautics and Astronautics

Autonomous Vehicle Risk Assessment

Authors:
Robert J. Moss (SISL)
Shubh Gupta (NAV Lab)
Robert Dyro (ASL)
Karen Leung (ASL)

Advisors (Stanford):
Mykel J. Kochenderfer (SISL)
Grace X. Gao (NAV Lab)
Marco Pavone (ASL)
Edward Schmerling (ASL)
Anthony Corso (SISL)

Sponsor Advisors (Allstate):
Regina Madigan
Matei Stroila
Tim Gibson

September 2020 – June 2021

Contents

Abstract

Finding failures in autonomous vehicles is especially important when trying to perform risk assessment in simulation before system deployment. When working with high-fidelity simulators and computationally expensive autonomous systems, efficiently finding failures becomes a primary objective. This work proposes an end-to-end risk assessment framework for autonomous vehicles—combining risky driving scenarios, sensor observation models, autonomous vehicle policies, efficient falsification techniques, and risk assessment methods. To build an observation model of the sensor noises in the environment, this work proposes a surrogate modeling-based approach to learn the underlying probability distributions of the noise disturbances from data. To automatically find high-likely failures in simulation, this work extends a reinforcement learning approach known as adaptive stress testing to be more data efficient. In a low-fidelity simulator, we compare two traffic flow-based driver models and introduce an intelligent behavior model learned through model predictive control to compare against a more realistic driving policy. The proposed framework is open source and has a modular design to extend to higher-fidelity simulators.

1 Introduction

To prevent loss of life and property, it is necessary to validate the safety of autonomous vehicles (AVs) before their widespread deployment into the real world. Along with validation, it is important that the risk of the AV systems can be properly quantified, assessed, and compared to other AVs. To stress the AV systems, it is common to develop simulation-based environments that attempt to mimic realistic driving scenarios and sensor observations [1]. Failures in AV systems tend to focus on collisions with other vehicles or pedestrians caused by errors in the sensor observations which lead control actions to an unsafe situation [2]–[5]. Therefore, safety validation requires identifying and analyzing failures that may occur as a result of likely sensor errors within a simulation environment. In this work, we develop an end-to-end framework for autonomous vehicle risk assessment with proposed methods for data efficient validation.

The end-to-end AV risk assessment framework is illustrated in fig. 1. First, realistic risky driving scenarios are selected; scenarios inspired by the US National Highway Traffic Safety Administration (NHTSA) pre-crash database [6]. These driving scenarios include difficult situations that have been analyzed to frequently lead to collisions. This includes four-way intersection crossings, hard braking on a highway, a pedestrian in a crosswalk, merging onto a highway, blind left turns, etc. Once scenarios are selected, the observation models that model the sensor characteristics in the environment are configured (e.g., GPS, radar, etc.) Errors in these observation models are the source of failures in the studied AV systems. Next, the particular AV systems (i.e., AV policies) are selected for stress testing. In this work, we focus on three AV policies: the intelligent driver model (IDM) [7], the Princeton driver model [8], and a behavior agent generated using model predictive control (MPC) that was develop as part of this work (see section 3.3). Using these AV policies, the objective is now to validate their safety in simulation using efficient techniques that search for failures. The *adaptive stress testing* (AST) [9] approach is the primary focus of this work, which we extend to be more data efficient using several proposed modifications. The goal is to efficiently find many high-likelihood failures to produce an empirical distribution of the cost (or severity) of

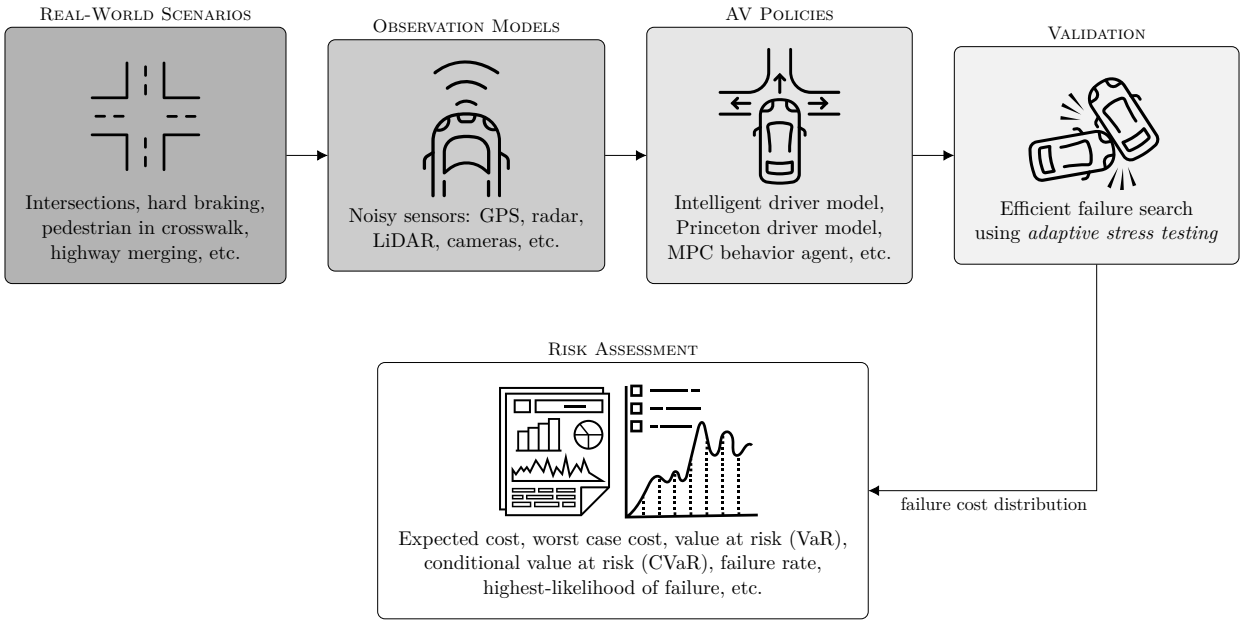


Figure 1: High-level end-to-end autonomous vehicle risk assessment framework.

failure. We use the closing velocity (i.e., closing rate) at time of collision as the cost measure of a failure. Using closing rate as the cost metric has been shown to be an effective measure of the severity of failure [10]. Finally, given this collected distribution of cost values, we propose methods to assess the AV risk using a combination of metrics output by the failure search (e.g., failure rate, highest likelihood of failure, etc.) and combine them with commonly used risk metrics from the financial and robotics industries (e.g., expected cost, worst case cost, value at risk (VaR), and conditional value at risk (CVaR)) [11].

We hold a principle that the end-to-end framework should be modular to easily add new scenarios, sensors, AV policies, validation solvers, and risk assessment metrics. This work resulted in several software packages written in the Julia programming language [12], all under the open-source `AutonomousRiskFramework`[1] tool. We extend the `POMDPStressTesting.jl`[2] [13] AST tool to include our proposed data efficient methods. This work is also based on existing AV simulation environments from the `AutomotiveSimulator.jl`[3] package, which models 2D driving scenarios. We also have preliminary work extending the framework to higher fidelity simulation environments like CARLA [14] (see appendix A.2).

We focus on AST as the primary validation method for several reasons. For safety validation through simulation, AST has been proposed in prior work and shown to be effective [1], [15]. AST uses reinforcement learning to search for the most likely sequence of disturbances in the stochastic variables within the environment that lead to a failure event. The framework applies to a broad range of autonomous systems and driving scenarios with multiple interacting agents (where agents could be other vehicles or pedestrians). The driving or motion behaviors of all agents in the simulation environment are assumed to be *black box*, or only known up to the input/output behavior. This is useful because the driving behaviors in modern autonomous systems rely on a variety of complex algorithms with multiple components [4], [5]. Section 2.1 provides a more detailed background on AST.

Although prior work on AST has been effective at discovering autonomous system failures in simulation, the existing framework requires a known probability model of the environment disturbances. This requirement is restrictive because many simulators do not provide direct access to the probability values associated with disturbances. Additionally, these disturbances are high-dimensional, such as errors present in simulated camera images or LiDAR point clouds [4], [5]. Therefore, directly searching for failure-causing disturbances is difficult due to a large and complex search space. The problem is exacerbated by the sequential nature of the search-space since sensor observations across multiple time steps must be considered. Furthermore, failure-causing errors in sensor observations are rare, and require running the simulation a large number of times to determine the probability, thus resulting in high computation costs. To address these challenges, part of this work extends the AST framework to incorporate a learned probability model of environment disturbances. Instead of high-dimensional sensor observations, our method models disturbances in the low-dimensional environment state as perceived by the perception software on the autonomous system. Using the disturbance values from running the simulation several times, we fit a model of disturbance probability as a function of the true environment state. Therefore, our framework can be used both in scenarios that contain multiple high-dimensional sensor observations and in situations where a probability model of the disturbances is unavailable.

Another drawback of current AST approaches is a potential for inefficient failure searches due to stochasticity in the solvers. When dealing with fast 2D simulators, data efficiency may not pose

[1] `https://github.com/sisl/AutonomousRiskFramework`
[2] `https://github.com/sisl/POMDPStressTesting.jl`
[3] `https://github.com/sisl/AutonomotiveSimulator.jl`

a problem to find failures. Yet when extending this work to higher-fidelity 3D simulators, efficiently finding failures becomes vital. Motivated by data efficiency, we propose several modifications to AST for more data efficient failure searches. The first modification is to use the closure rate value within the AST reward function (detailed in section 3.1.1), which satisfies a potential-based reward shaping function that has been proven to preserve the optimal policy while speeding up learning [16]. The second modification is to extend the Monte Carlo tree search (MCTS) algorithm [17] used as an AST solution method to learn a rollout policy that biases towards failures rather than a strictly random rollout of future trajectories. We learn these rollout policies using deep reinforcement learning (DRL). To use DRL, we first must have a useful state representation to be used as an input to the policies. Because we are still treating the AV system as a black box, we explore the use of *state proxies* to stand in for the true state. We investigate two state proxies: the distance to a failure (which is already required by AST) and the closure rate (which is simply derived from the distance metric). We also perform ablation studies to highlight the individual contributions of each of these data efficient approaches (section 4.2).

Finally, to compare two "simple" AV policies (namely, the IDM and the Princeton model), we also develop an optimization-based agent using *model predictive control* (MPC). This lets us create and evaluate a proxy for future and existing autonomous vehicles which rely on trajectory optimization in their low-level decision making.

1.1 Contributions

The main contributions of this work are a combination of research, development, and analysis. The primary contributions include:

- An open-source, modular, end-to-end autonomous vehicle risk assessment framework.
- Tools for modeling the probability distribution of sensor errors within the risk assessment framework to extend the framework to unknown disturbance models and improve efficiency.
- Data efficient extensions to AST including potential-based reward shaping (using closure rate), learned rollout policies for efficient MCTS, and the use of state proxies to preserve the black-box assumption while extending AST to state-based solvers.
- Optimization-based MPC agent complementing existing, simpler behavior models to represent autonomous vehicles of the future.
- Integration of the open-source CARLA simulator with the stress testing framework and preliminary analysis
- Preliminary analysis comparing AV policies using the end-to-end framework.

1.2 Related Work

Many approaches to safety validation of autonomous systems have been previously proposed in literature [18]. These approaches span across a variety of domains, such as optimization [19], path-planning [20], reinforcement learning [9], and sampling techniques [21]. Mathesen, Yaghoubi, Pedrielli, *et al.* [19] proposed a stochastic search method that mixes global and local search for generating test cases. Zutshi, Deshmukh, Sankaranarayanan, *et al.* [20] used randomized algorithms to search over segments of trajectories produced during the execution of an autonomous system. Although these approaches discover potential failures, they do not prioritize failures that are more likely to occur. Lee, Mengshoel, Saksena, *et al.* [9] formulated the problem of system failure

identification as a Markov decision process and use reinforcement learning to optimize it, thus introducing the adaptive stress testing framework. The framework uses known probability models of errors to prioritize failures based on their likelihood. However, the requirement of a known probability model of errors limits the framework's applicability to high-fidelity simulators. Huang, Lam, LeBlanc, *et al.* [21] suggested discovering failures by sampling sensing errors from a piece-wise probability model constructed using simulated or real-world data. Although such an approach is useful for sensors with low-dimensional observations, extending them for general sensors which may have high-dimensional errors is challenging due to the huge data requirement.

To search for failures in systems involving high-dimensional sensor observations, many previous works use gradient-based optimization. These approaches, based on falsification, search for inputs to a system that invalidate (falsify) system requirements while satisfying robustness constraints. Donzé and Maler [22] developed a sensitivity-based search algorithm to find parameters of a system that minimize a robustness measure. Abbas, Winn, Fainekos, *et al.* [23] used gradient descent to find trajectories that violate system safety specifications. Deng, Zheng, Zhang, *et al.* [2] analyzed multiple adversarial attack techniques that search for failures in autonomous systems that depend on camera images. However, these techniques require a known and differentiable model for perception and decision making based on the sensor observations. This constraint is restrictive because modern autonomous systems contain complex perception, estimation and decision-making algorithms which might not be fully known for stress testing.

Since failures in an autonomous system usually result from rare events, realizing these failures in a simulation may require a large number of computationally expensive evaluations. Therefore, many prior works have focused on reducing the number of required system evaluations by instead evaluating approximate models of the system quantities. Li and Xiu [24] proposed computing failure probability using a hybrid approach that samples both the original system and a constructed surrogate model. Xiao, Zhan, and Yuan [25] developed a method for reliability analysis of a system based on adaptive importance sampling and kriging models (i.e., Gaussian process regression). Pulch [26] applied strategies of generalized polynomial chaos to determine failure probabilities in periodic systems. These approaches are efficient since they replace the expensive probability evaluation requiring multiple simulations with evaluation of the constructed model. However, the accuracy of the estimated failure probability depends on the accuracy of the approximated model of system quantities, which is difficult to ensure for complex scenarios involving several interacting road agents and multiple time instances.

Autonomous vehicle risk assessment has been extensively studied in the literature to ensure AV policies are safe before deployment [27]–[30]. Koopman and Wagner [27] layout the current challenges when testing and validating AVs and indicate that machine learning-based systems can be complex to validate and that collecting data on rare edge cases where the AVs fail may be expensive and hard to scale. Our work attempts to address this concern with an adaptive framework that treats the AV policy and environment as a black box and intelligently and efficiently finds failures to better assess the AV risk. Bhavsar, Das, Paugh, *et al.* [29] perform risk analysis of AVs in mixed traffic streams by breaking down the AVs into their individual components, but they rely on previously published data to perform the assessment. One goal of our work is to automatically collect a dataset of failures in simulation in an inexpensive manner without having to break down the AVs into their subcomponents. Other work defines the AV risk as the probability of collision itself [31], where Chen, Liu, Chen, *et al.* [32] separate the probabilistic risk into high and low levels of risk based on some threshold and Wang, Huang, Jasour, *et al.* [33] define risk as the probability of entering a ring around the ego vehicle. Instead of just probability of collision, our work defines risk as a weighted combination of metrics computed over the failure cost distribution (where we use the closing rate at time of collision as the cost metric) combined with failure metrics

found during falsification. Shannon, Rizzi, Murphy, *et al.* [34] define risk using the "expected compensation costs (ECCs)" for vehicle collision modeling which considers all injuries involved and links to compensation and medical costs, lost earnings, among other metrics. Their work uses a delta-V measure (i.e., the change in velocity before and after the collision). Richards [10] also uses the delta-V measure for risk and the closing rate (or impact speed) when dealing with collisions between a vehicle and pedestrian. Their work indicates that while delta-V captures parameters of the vehicles such as weight, the closing rate at impact is effective at predicting the probability of fatality [10].

This report is broken down as follows: Section 2 provides necessary background information on AST and MCTS, section 3 describes the technical approaches and contributions of this work, section 4 details the experiments and results from ablation studies and a full end-to-end risk assessment, and section 5 concludes with a discussion about current and future work. Supplemental material is provided in the appendices.

2 Background

In this section, we present an overview of the adaptive stress testing (AST) framework proposed in [1] and [9]. Using a model of the disturbances to agent states in a simulated environment, AST finds the most likely path to a failure event in simulation. We first describe the Markov decision process (MDP) for finding failures in AST, then we detail the black-box interface required to interact with the AV system under test (SUT). We then describe the Monte Carlo tree search algorithm often used in reinforcement learning for solving the MDP. Figure 2 illustrates the AST problem formulation.

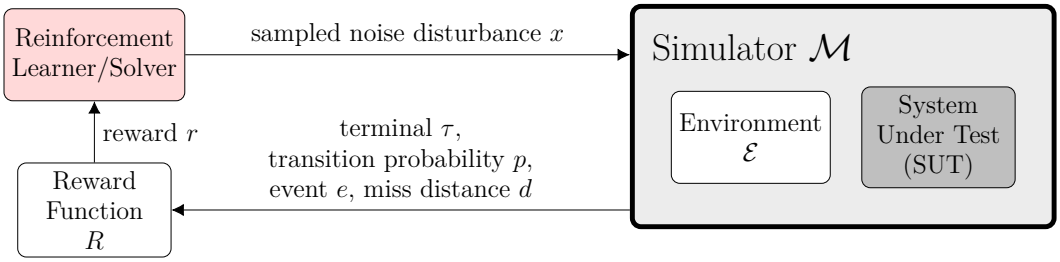


Figure 2: Adaptive stress testing formulation.

2.1 Adaptive Stress Testing MDP

AST formulates the search for the most likely path to a failure event as an MDP. An MDP is characterized by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is a set of states; $\mathcal{A}$ is a set of actions; $P(s' \mid s, a)$ is probability of transition from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ when action $a \in \mathcal{A}$ is chosen; $R(s, a)$ is the reward function; $\gamma \in [0, 1]$ is the discount factor for discounting future reward values.

Simulator. In AST, the simulator $\mathcal{M}$ consists of the environment $\mathcal{E}$, the system under test (SUT), and any interacting agents. The SUT interacts with the environment $\mathcal{E}$ and the agents over discrete time instances $t \in \{0, \ldots, T\}$.

States. Any given state $s \in \mathcal{S}$ in the MDP is the state of the simulation environment $\mathcal{E}$ comprised of positions and velocities of the SUT and all agents. A path (or trajectory) in the MDP is represented as a sequence of states $s_{0:t} = \{s_0, \ldots, s_t\}$ up to time t.

Actions. An MDP action in AST denotes choosing a value of environmental disturbance $x_t \in N$ from a specified probabilistic disturbance model N. The disturbance x_t captures the environment stochasticity from all factors such as wind conditions or uncertainty in motions of environment actors such as pedestrians. At each time t, the environment generates an observation $o_t \in \mathcal{O}$ based on state s_t and disturbance x_t.

Transition Probability. Based on the generated observation o_t, the MDP then transitions to the next state s_{t+1} by executing driving behaviors of all agents in the environment $\mathcal{E}$. Since all environment stochastic variables are modeled in x_t, the transition $(s_t, x_t) \to s_{t+1}$ is deterministic within the simulator such that $P(s_{t+1} \mid s_t, x_t) = 1$ for some (s_t, s_{t+1}, x_t).

Reward Function. The reward function directs the search towards two objectives: 1) finding a failure event, and 2) maximizing the path likelihood. The reward function is given by

$$R(s, x) = \begin{cases} R_T & \text{if } s \text{ is a failure event} \\ -d(s) & \text{if } s \text{ is a non-failure terminal event} \\ \log p(x) & \text{otherwise,} \end{cases} \tag{1}$$

where R_T is the reward for finding a failure (we use $R_T = 0$ in this work), $d(s)$ is a user-defined measure of miss distance to the failure, and $p(x)$ is computed from the disturbance model N. The reward function is designed to maximize the sum of the log-likelihoods, which is equivalent to maximizing the product of the likelihoods.

2.2 AST Black-Box Interface

To apply AST to an autonomous system, the AST black-box interface must be defined. The interface wraps around the simulator $\mathcal{M}$, which includes the environment and SUT. It defines how we initialize, evaluate (i.e., step), and determine miss distance from the system. Table 1 outlines the interface functions and their inputs and outputs [35]. We then describe the implemented AST interface functions used for the AV risk problem.

Table 1: Adaptive Stress Testing Interface [35]

Function	**Input $\mapsto$ Output***
Initialize	$\mathcal{M} \mapsto \emptyset$
Evaluate	$\langle \mathcal{M}, x \rangle \mapsto \langle p, d, e \rangle$
Transition	$\langle \mathcal{M}, x \rangle \mapsto p \in \mathbb{R}$
MissDistance	$\mathcal{M} \mapsto d \in \mathbb{R}$
IsEvent	$\mathcal{M} \mapsto e \in \mathbb{B}$
IsTerminal	$\mathcal{M} \mapsto \tau \in \mathbb{B}$

* $\mathbb{R}$ indicates all real values and $\mathbb{B}$ indicates a boolean value.

Each of the following interface functions takes the simulation structure $\mathcal{M}$ as input, and may modify the structure in place.

- Initialize: To initialize the simulator, we reset simulation variables. This includes the simulation time $t = 0$, the underlying adversarial MDP that controls the ego and adversarial vehicles, the SUT state from the adversarial MDP, the current noise disturbances, and the

previous distance value used to calculate the closure rate. The simulation structure is modified in place and nothing is returned (indicated by the empty set $\emptyset$).

- Evaluate: To evaluate the simulator and SUT, we combine three calls to the below functions (Transition, MissDistance, and IsEvent) into a single function call. Here, we also update the previous distance value used in the closure rate calculation. We include the sampled noise disturbance x as input, and return the transition probability p, the miss distance d, and the failure event indication e as output.
  - Transition: To transition (or step) the simulator, which includes the SUT, we first increment the simulation time t by our dt (in our case $dt = 1$). Then we take the sampled noise disturbance x given as input and apply the disturbance to the agents in the environment (this could be applied to the ego vehicle only or to all agents). Then the underlying adversarial MDP that controls the SUT (and all other agents) is propagated forward one step, taking the current MDP state and disturbances as input and returns the next state. Finally, we return the log-likelihood p of the sampled noise disturbance and MDP state as output (which we ultimately want to maximize given failure).
  - MissDistance: The miss distance d is calculated as the distance between the centers of the ego vehicle and the other agent (either another vehicle or a pedestrian). Then d is returned as output.
  - IsEvent: To indicate that a failure event occurred or not, we determine if there was a collision between the agents using the parallel axis theorem [36]. A boolean e is return indicating if a collision occurred.
- IsTerminal: Finally, we provide an indication that the simulation has terminated. We define termination if both vehicles are out of the frame, a collision event occurred, or the maximum simulation time T has been reached. A boolean τ indicating termination is returned as output.

2.2.1 Monte Carlo Tree Search

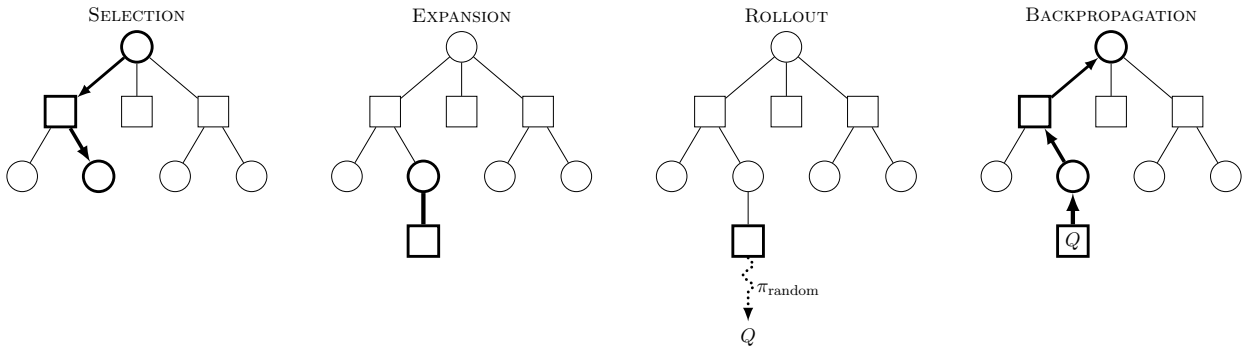


Figure 3: The four steps of the Monte Carlo tree search algorithm using a random rollout policy π [35].

The Monte Carlo tree search (MCTS) algorithm [17] is a stochastic tree-based search algorithm often used in planning and reinforcement learning (RL) [37]. MCTS iterates through simulations and uses rollouts of a random policy to estimate the state-action Q-value of each node in the tree. Figure 3 illustrates the four steps of each simulation: Selection, Expansion, Rollout, and Backpropagation. MCTS is an "anytime" algorithm, often used as an online RL solver, which

means it can be stopped after any iteration and the best solution (so far) can be returned. Yet after many more iterations, the solution will improve. The random rollout policy is executed during the Rollout step to provide an unbiased estimate of the future Q-value from that particular state node. This can be very useful for exploration, yet may pose as a bottleneck when we want to bias towards some portion of the state space.

3 Approach

This section details the technical approaches proposed by this work. The approaches range from data efficient falsification methods for better failure searches, sensor observation modeling for wider application to complex environmental models, and an intelligent behavior AV agent learned using model predictive control for a more realistic comparison to other lane-following AV policies.

3.1 Data Efficient Falsification

To assess risk, we must first find failures. A motivating objective of this work is to find failures efficiently, especially when the SUT may be computationally expensive to evaluate. There are several proposed methods we explore to more efficiently find failures using AST. The first method is a modification to the AST reward function itself to speed up learning, which means it can be applied regardless of the chosen solver. The second method is a modification of the MCTS solver to use a precomputed learned rollout policy to bias towards failures, instead of a random rollout policy. This section describes the details of these two data efficient falsification methods.

3.1.1 Potential-Based Reward Shaping

Reward shaping is a technique used in reinforcement learning to help guide an agent to their goal through clever modifications to the reward function [16]. Ng, Harada, and Russell [16] showed that if the reward function is modified using a potential-based reward shaping function F that satisfies certain non-cyclic properties, then the optimal policy is invariant under this reward modification. Let $F(s, s')$ be the *reward shaping function* $F : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}$ we employ. The reward shaping function becomes a *potential-based* shaping function when we use a real-valued function $\phi : \mathcal{S} \mapsto \mathbb{R}$ that *necessarily* satisfies that there are no positive reward cycles (which could lead the agent to divert from the goal to collect more intermediate reward, thus "distracting" the agent). The reward shaping function F then becomes the difference of potentials between consecutive states:

$$F(s, s') = \gamma\phi(s') - \phi(s) \tag{2}$$

Note, in this work (and most other AST research [9], [15]), we use a discount factor of $\gamma = 1$ to not penalize failures later in the trajectory.

In this work, the potential function ϕ is defined as the negative distance to failure $\phi(s) = -d(s)$. As shown by Ng, Harada, and Russell [16], using a "distance-to-goal" potential function does not introduce cycles and thus satisfies the necessary condition to preserve the optimal policy. Therefore, we define the reward shaping function $F(s, s')$ as:

$$\begin{aligned} F(s, s') &= \phi(s') - \phi(s) && \text{(potential-based reward shaping function definition)} \\ &= d(s) - d(s') && \text{(substitute } \phi(s) = -d(s)\text{)} \\ &= d_\Delta(s, s') && \text{(closure rate, i.e. distance-rate)} \end{aligned}$$

The shaping function is added to the existing reward function R to get the modified (or augmented) reward function R':

$$R'(s, x, s') = R(s, a) + F(s, s') \tag{3}$$

Adding the potential-based reward shaping function F has been shown to reduce learning time for an agent to reach their goal (in our case the goal is a failure/collision) [16]. Knowing that the optimal policy is the same under R and R', we can expect to find failures faster in simulation while achieving similar (or increased) failure rates using R' as we would under R.

The AST reward function described in eq. (1) can be written as

$$R(s, x) = \log p(x) + R_E \mathbb{1}(s_T \in E) - d\mathbb{1}(s_T \notin E) \tag{4}$$

where s_T is the terminal state, E represents the set of all failure events, and $\mathbb{1}(\cdot)$ is the indicator function (which evaluates to 1 if the input is true and 0 otherwise). Using this form of the reward function, the augmented reward function R' can then be written as

$$R'(s, x, s') = \log p(x) + R_E \mathbb{1}(s_T \in E) - d\mathbb{1}(s_T \notin E) + d_\Delta(s, s') \tag{5}$$

which we can use without changing the existing AST black-box assumption because d_Δ is a function of two distances, which we already have access to through the Distance interface function.

3.1.2 Learned Rollout Policies

To increase the efficiency of finding failures, we investigate the use of a pretrained policy from a separate failure search to be used as the replacement rollout policy that MCTS employs. Figure 4 illustrates the learned rollout stage of MCTS, where we use a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that biases towards failure which was previously learned using *proximal policy optimization* (PPO) [38]. The input to the learned policy is the state of the environment, but we investigate the use of a *state proxy* to approximate the true state with known information while preserving the AST black-box assumption. The output of the learned policy is an action, i.e., a sample of the noise disturbances to apply to each agent in the environment. Figure 5 illustrates the phased approach starting with the offline learning phase, then using the learned policy for a more efficient MCTS phase, and finally using the collected cost distribution for risk assessment.

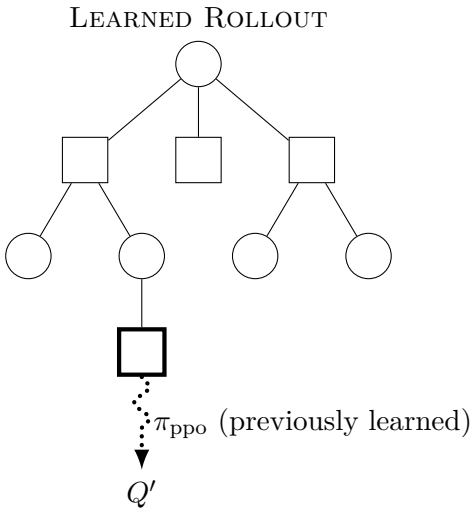


Figure 4: The MCTS rollout phase using a learned rollout policy output by PPO.

By replacing the random rollout policy that is currently used by standard MCTS, we expect to bias the search towards failures to ultimately find more failures that we can use for a wider spread of cost evaluation of the AV policies under test. This also allows us to explore the space of failures without increasing the number of iterations we run MCTS. Algorithm 1 details the learned rollout algorithm used during the third stage of MCTS, which recursively samples a next action from the learned policy (instead of random) and simulates or "hallucinates" the future trajectories down a particular tree path. By replacing the random rollout with a rollout biased towards failures, we can greatly increase the failure rate of MCTS and use an offline policy to increase the performance of an online solution method.

Algorithm 1 Simulate rollouts using a previously learned policy.

function LEARNEDROLLOUT(s, d, π)
  if $d = 0 \vee$ ISTERMINAL(s)
    return 0
  $x \sim \pi_{\text{learned}}(x \mid s)$ ▷ sample action from learned policy
  $(s', r) \sim G(s, x)$ ▷ calls EVALUATE(s) and $R'(s, x, s')$
  return $r + \gamma$ LEARNEDROLLOUT($s', d - 1, \pi$)

The use of any offline learning algorithm can be selected, where we chose PPO for its simplicity in implementation and its empirical performance in the literature. It is important to note that we need some representation of a state value to be able to use the state-based RL solvers like PPO. If we have access to the true state of the environment, then we can use that. But to keep the AST black-box assumption, and to allow this work to be easily extended to other black-box AV policies and simulators, we are interested in other measures we can use as proxies for the true state.

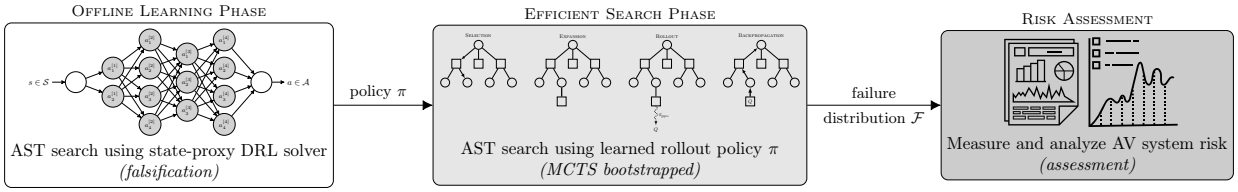


Figure 5: Phased efficient falsification for risk assessment.

3.1.3 State Proxy

The true state of the simulation environment consists of $s = \{x_1, y_1, \theta_1, v_1, x_2, y_2, \theta_2, v_2\}$, which are the xy-positions, the heading θ, and the velocity v of each agent. This information may not be readily available when moving this work to other simulators (e.g., CARLA), so we investigate the use of a *state-proxy* to replace the true state with known information we have already required from AST, namely the distance and rate metrics. Using the state-proxies, we can maintain the black-box nature of the problem by using either the distance from the DISTANCE AST interface function or the closure rate which is derived from the distance. Again, state-proxies also enable the use of deep reinforcement learning solution methods (*proximal policy optimization* (PPO) [38], *Trust-region policy optimization* (TRPO) [39], *deep deterministic policy gradient* (DDPG) [40], etc.) that require state as input to learned neural network-based policies.

3.2 Sensor Observation Modeling

In this section, we build on the AST framework to determine the most likely paths to failure events in a simulation under sensor observation errors. The proposed framework is applicable to scenarios where the probability distribution of the disturbances in the environment state due to the sensor observation errors may be unknown. The simulation environment consists of the system under test (SUT) as well as other agents (vehicles, pedestrians, etc.). At each time instant, the simulator generates observations o from a set of sensors. From the observations, the SUT uses onboard

perception algorithms to perceive the state $\tilde{s}$ of the surrounding environment and determines the control action c to take based on the driving policy. These control actions are then input to the simulator to advance to the next time instant.

We modify the optimization objective in AST to incorporate disturbances in perceiving the environment from sensor observations while simultaneously estimating the probability of the disturbances. Naïve approaches to evaluate the probability of disturbances require executing the computationally expensive simulation several times at each time instant. Instead, our modified AST objective evaluates the probability using a learned model η of disturbances. We learn the disturbance model in a two-phased approach:

- The first phase is the simulation phase that links the perception and driving algorithms on the autonomous vehicle with sensor observations from the simulator. By repeatedly executing the simulation loop, we first record several samples of the true environment state s and the noise in the perceived environment state $\tilde{s} - s$, where the state consists of quantities used by the controls system such as positions and velocities of all agents. Using the recorded states and noise, we then fit a surrogate probability model η (section 3.2.1), or the disturbance model, that models the probability distribution of the noise in $\tilde{s}$ conditional on the true state s.

- The second phase is the stress-testing phase that uses the constructed disturbance model η in AST optimization to search for the most likely sequence of disturbances that leads to a failure event. We use the Monte Carlo tree search algorithm (section 2.2.1) for optimization using the log-likelihood of disturbances from the probability model. Finally, we update the recorded set of states with the new environment states encountered during the search and repeat the two phases for either a predefined number of iterations or until convergence.

Figure 6 illustrates the proposed framework.

Our approach offers several advantages over existing stress testing techniques. Similar to the current work on AST, we treat the simulator as a black-box and do not assume any prior knowledge about the system transitions. Furthermore, we relax the previous requirement of a known probabilistic model of disturbances, and instead construct the model using sensor observations and perception algorithms within the simulation. As a result, our approach can be used with a broad range of high-fidelity simulators where a probability model of disturbances is unavailable. Generating sensor observations and executing the perception algorithm within the simulation is computationally expensive. Therefore, our approach separates the data-collection phase—in which a disturbance model is constructed by parallelly querying the simulator a fixed number of times—and the stress-testing phase for identifying failures using the constructed model. Unlike existing falsification approaches for high-dimensional sensor observations, we search over the lower dimensional space of disturbances applied to the quantities determined by the perception algorithm. Hence, our approach is scalable to environments with multiple interacting agents as well as observations from multiple sensors.

3.2.1 Surrogate Modeling

In most modern autonomous driving simulators, the probability associated with a disturbance x in the environment state is unknown or difficult to evaluate. For instance, the distribution of noise in sensor observations such as camera images have complex profiles that do not have a readily available closed-form expression. Furthermore, even if the source probability distribution of the simulated sensor noise is known (e.g., Gaussian), the resulting probability distribution of the perceived environment state $\tilde{s}$ and hence the disturbance x may be difficult to evaluate without

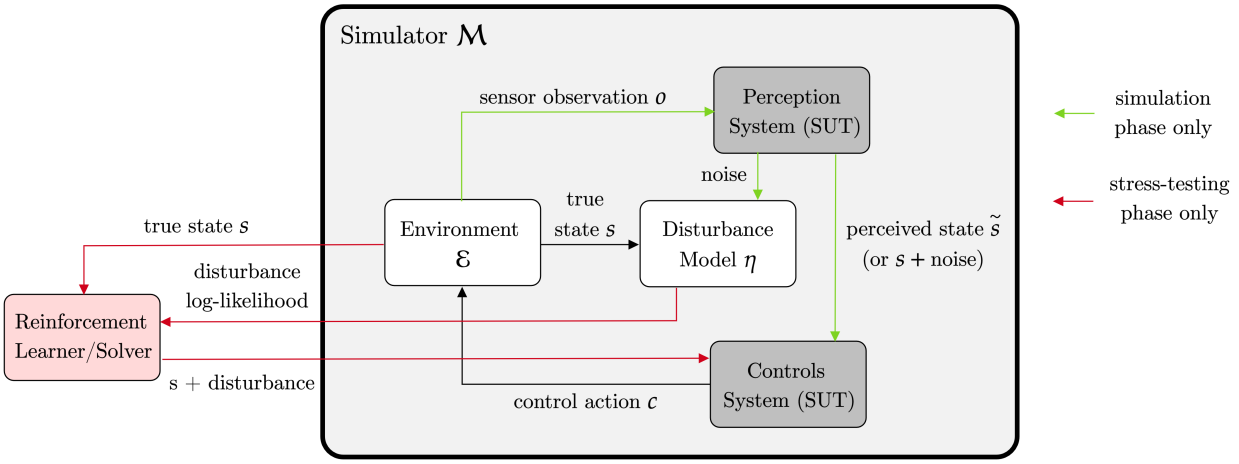


Figure 6: The proposed stress testing framework including sensor observation errors. The environment interacts with the autonomous vehicle software through sensor observations. In the simulation phase (black and green arrows), the perception system processes the sensor observations to determine a perceived state of the environment, which may differ from the true state within the environment due to environment noise. The control system then computes control actions to take based on the perceived state to advance the simulation. Through samples of the environment state and noise recorded from repeated evaluations of the simulation loop, a probability model of disturbances is constructed. Next, the disturbance model is used in the stress testing phase (black and red arrows) to determine the disturbance log-likelihood without sensor observations and guide the search for the most likely failure path.

assuming constraints on the functional relation between the noise x given an environment state s. To address these challenges, we construct a surrogate model $\eta(x, s) \approx \mathcal{P}(x \mid s)$ of the conditional probability distribution of x for an environment state s that can be evaluated easily.

In this work, we use a Mixture Density Network (MDN) [41] as our surrogate probability model

$$\eta(x, s) = \sum_{k=1}^{K} \pi_k(s) \mathcal{N}(x \mid \mu_k(s), \Sigma_k(s)),$$

where K is the number of mixture components, $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$ are the learned parameters of the probability distribution modeled via neural networks, and $\mathcal{N}(\cdot)$ is the multivariate Gaussian distribution. Furthermore, we have the constraints on the mixing coefficients π_k such that $\sum_{k=1}^{K} \pi_k(s) = 1$ and $\pi_k(s) \geq 0$. As a preprocessing step, we extract features from the state s based on position and velocity of the SUT as well as relative positions and velocities of all environment agents from the SUT. We feed the extracted features as inputs to the MDN and train by minimizing the negative log-likelihood over the recorded data from the simulation phase:

$$\mathcal{L}(x, s) = -\log(\eta(x, s)).$$

Since η captures the conditional probability distribution based on the environment state s, it can model variations in the disturbance probability based on the states of environment agents, for instance, capturing the increased/decreased noise in detecting as well as positioning an obstacle that is far from/close to the vehicle camera.

3.3 MPC Behavior Agent

Model Predictive Control (MPC) is an optimal control algorithm which selects actions taken by an agent by solving a finite horizon optimization program simulating agent's and the world's behavior into the future. Although in theory such an optimal computed plan can then by executed to its end, in practice, in order to reject disturbance, only the first action is executed and the plan is recomputed at the next time step. Because the Model Predictive optimization is performed over both agent's action u, future states z and the state of the world s, a model of the agent and the evolution of the world (most generally in response to these actions, but possibly statically) is required. In practice, the model of agent's and the world's states in response to the agent's actions (which are optimized) need not be perfect, the optimal plan is recomputed at every step, to reject disturbance, which include modeling errors. Typically MPC is computed for discrete dynamics (with discrete time steps), which results in a computational limit on the horizon considered into the future. Realistically, the choice of the horizon is an algorithmic hyperparameter, but in practice a horizon encompassing the next second or two of driving is used. The finite horizon approximation can be made exact if the optimal cost-to-go is known for the last state in the plan, but, in general, knowing that implies having solved the optimal control problem for all possible states which voids the need for MPC.

In this work we consider linear second order integrator dynamics, in the lane frame of reference, for the longitudinal and, optionally, lateral dynamics of the vehicle. We justify this assumption by noting that it is trivial to construct an autonomous car controller which rejects the centrifugal force of a curving road, rendering the dynamics as we formulate them to be linear. Linear dynamics in frame of reference of the lane significantly simplify the computational complexity of the MPC optimization because linear constraints (encoding dynamics) typically do not increase the computational difficulty of solving an optimization problem for a given driving objective.

Given the linear autonomous vehicle dynamics in the lane frame of reference

$$z^{(k+1)} = Az^{(k)} + Bu^{(k)}$$

with

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

and autonomous car dynamical state z

$$z^{(k)} = \begin{bmatrix} p^{(k)}_{\text{longitudinal}} & v^{(k)}_{\text{longitudinal}} & p^{(k)}_{\text{lateral}} & v^{(k)}_{\text{lateral}} \end{bmatrix}^T$$

where p denotes position and v the speed.

We formulate the following MPC optimization problem, resolved at every step

$$\begin{aligned} \text{minimize}_u \quad & c\big(u^{(k)}, z^{(k)}\big) \\ \text{subject to} \quad & z^{(0)} = z_0 \\ & z^{(k+1)} = Az^{(k)} + Bu^{(k)} \\ & z^{(k)} \in \mathbb{Z} \;\; u^{(k)} \in \mathbb{U} \quad \forall k \in [0, \ldots, N_{\text{horizon}}] \end{aligned}$$

where $c\big(\cdot, \cdot\big)$ is the objective associated with the state and actions of the vehicle and $\mathbb{Z}, \mathbb{U}$ are feasible sets for the state and actions of the vehicle (e.g., the state z should deviate less than 1 m from the lane centerline and the action's absolute value u should not exceed 10 $^{m}/_{s^2}$).

The specific behavior of the MPC agent is highly dependent on the choice of the objective function $c(\cdot, \cdot)$, because, within the realm of dynamically feasible trajectories, there exists an objective function for every possible behavior. For example, whether the car attempts to avoid obstacles or actively seeks to crash into them is dependent on the sign of the term quantifying that in the objective function. The behavior is then specified in terms of desirability of outcome and not, like in simpler, direct driving models, through direct dependence of actions on the state of the world. MPC behavior specification via the objective enables more behaviors that take into account the future explicitly and is much closer to how humans and autonomous vehicles act in reality—by planning and not just reacting.

3.3.1 Human-Interpretable Objective Specification via Signal Temporal Logic

Signal Temporal Logic (STL) allows to quantitatively express several human-interpretable notions of outcomes and allows for temporal dependencies between them. The language of STL enables the specification of desired outcomes expressed in human language (like English), for example, *never crash into another car* and *do not pass until the oncoming lane is clear* in a principled way in the objective function. For a formal definition of STL we refer the reader to Section C in the Appendix.

3.4 Risk Assessment Method

To quantitatively measure the risk of an AV policy, we base our assessment on several risk-based metrics from the literature and failure-based metrics output from the AST search. By defining some measure of *cost* to the system when it fails, we can calculate risk metrics over the distribution of cost. In this work, we use the *closure rate* (i.e., relative speed at time of collision) as the cost measure Z, sometimes called the severity of failure. It has been shown that this closure rate value is a useful indicator of vehicle risk in predicting occupant fatalities [10].

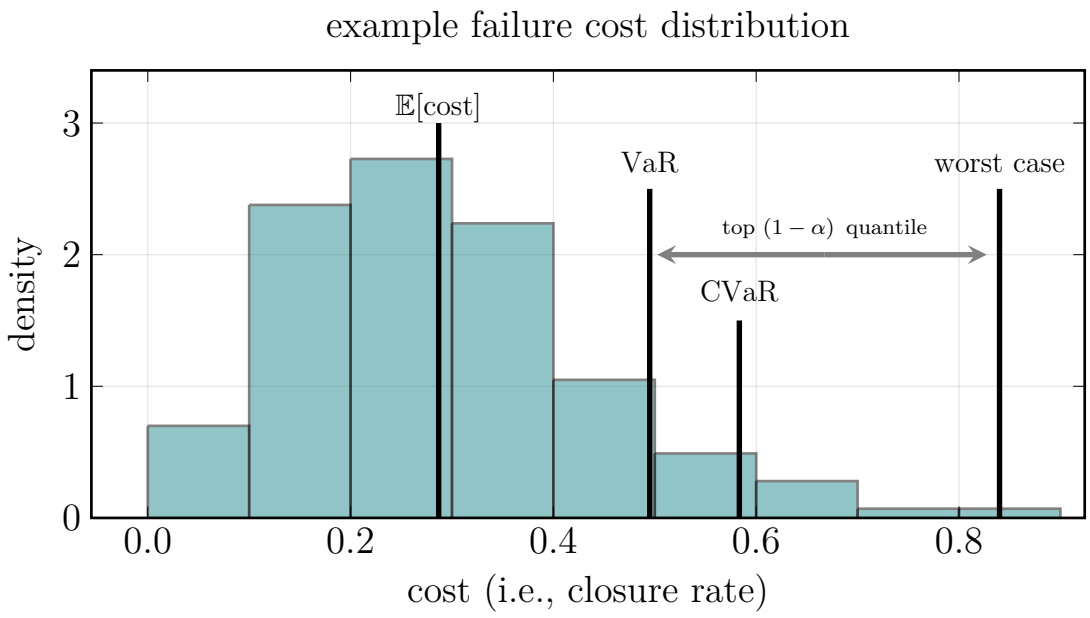


Figure 7: Distribution of cost values with expected cost, VaR, CVaR, and worst case cost illustrated.

Cost metrics. The robotics and financial communities [11] have previously used *value at risk* (VaR) as their primary risk assessment metric. Given some risk tolerance threshold α (where we are willing to accept α fraction of the highest cost), we define VaR as the value at that particular α threshold:

$$\text{VaR}_\alpha(Z) := \min\{z \mid \mathcal{P}(Z > z) \leq \alpha\} \qquad \text{(lowest cost given probability threshold)}$$

Figure 7 illustrates the cost metrics we use in this work. Along with VaR, we also record the expected cost of failure and the worst case cost of failure. Recently, the robotics community suggested that *conditional value at risk* (CVaR) is a well-formed risk metric for autonomous robotics applications, including autonomous vehicles [11]. CVaR is defined as the expected cost under the conditional distribution set by the risk tolerance threshold α:

$$\begin{aligned} \text{CVaR}_\alpha(Z) &:= \frac{1}{\alpha}\int_{1-\alpha}^{1} \text{VaR}_{1-\tau}(Z)d\tau && \text{(expected cost under conditional distribution)} \\ &:= \mathbb{E}_{\mathcal{P}'}[Z] && \text{(where } \mathcal{P}'(z) := \mathbb{P}\{z \mid \mathcal{P}(Z > z) \leq \alpha\}\text{)} \end{aligned}$$

Shown in fig. 7 and further analyzed by Majumdar and Pavone [11], we can see that CVaR captures more information about the tail of the failure cost distribution which is deemed the "acceptable" failure region given α.

Failure metrics. Along with the cost metrics, we use three failure-based metrics output from the AST failure search as another measure of AV policy sensitivity to failures. We use the failure rate (i.e., the number of failures over number of simulations or the biased probability of failure) as an indicator of how often the AV policy fails in simulation. We use the first failure episode (or index) as a way to measure the "ease" of finding failures. Finally, we use the highest log-likelihood of failure as an indicator of the likelihood of failure within the simulation environment.

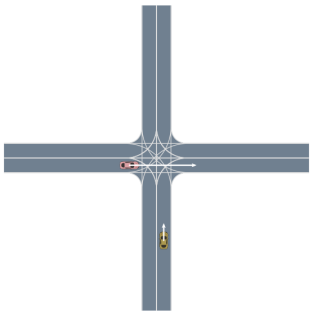
By combining the cost metrics and the failure metrics, we can produce an overall measure of risk by plotting the values over a polar plot and calculating the area under the curve. If necessary, we can apply different weights to each metric using a vector $\mathbf{w}$, which can be chosen by the user or policy maker when balancing how much each of the metrics count towards the overall AV risk. In full, we have the following seven metrics used for risk assessment:

- $\mathbb{E}[Z]$ (mean cost of failure)
- VaR (Value at Risk)
- CVaR (Conditional Value at Risk)
- Worst case cost
- Failure rate (i.e., biased p(fail))
- First failure episode (i.e., "ease" of finding failures)
- Highest log-likelihood of failure

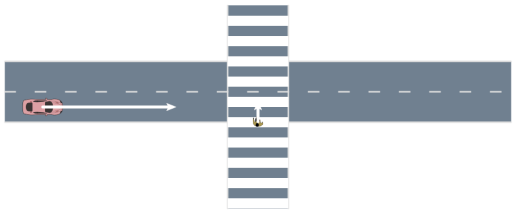
The last three failure metrics are output by the AST search, and the other cost metrics are agnostic to the method of falsification (and their cost value Z can be changed from severity of failure to monetary value or other cost measures).

4 Experiments and Results

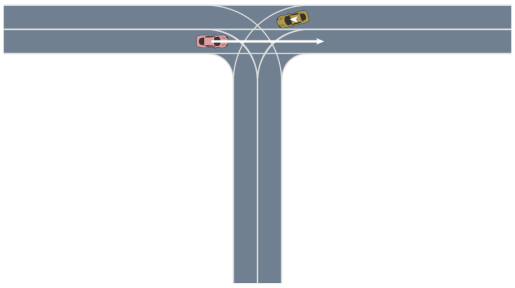
Experiments were preformed to first test the individual components proposed for increased data efficient falsification. These ablation studies first show how well a random search solver compares to MCTS using the IDM as a baseline. The next ablation study tests the effect of the potential-based reward shaping function proposed in section 3.1.1. We also test the effects of the learned rollout policy on increased failure rate for a better coverage of the distribution of cost (which we use to calculate risk). We study different uses of state proxies in place of the true state to both extend this to state-based deep RL solvers and to preserve the black-box assumption within AST. Experiments to test the performance of the observation modeling described in section 3.2 were also preformed. Lastly, we run an end-to-end risk assessment across six NHTSA pre-crash inspired scenarios comparing three AV policies and quantifying their risk using several weighted metrics.



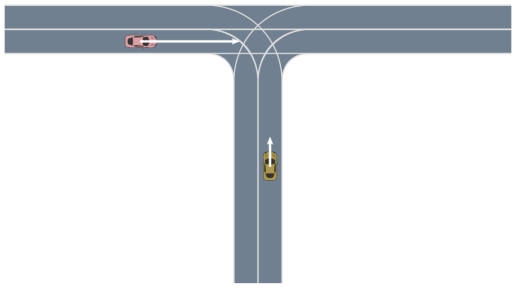
(a) Crossing intersection.



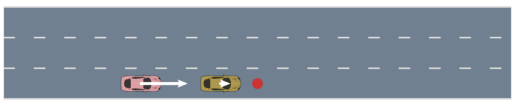
(b) Pedestrian crosswalk.



(c) Head-on T-intersection turn.



(d) Left T-intersection turn.



(e) Highway stopping.

(f) Highway merging.

Figure 8: Driving scenarios studied in this work, inspired by the NHTSA pre-crash scenario database [6]. The red vehicle is the ego vehicle and white arrows indicate the current velocities.

4.1 Scenarios

Figure 8 illustrates the six driving scenarios that have been shown to be stressing cases where collisions tend to occur [42]. The six driving scenarios were designed based on the NHTSA pre-

crash scenario database [6] using the `AutomotiveSimulator.jl`[4] package. The scenarios provide a useful mix of different normal driving situations where some scenarios result in failures easily (as shown in table 2) and others are more rare.

4.2 Ablation Studies

Ablation studies are primarily used in the field of machine learning to investigate the contributions of many individual components to the overall performance of some method. In this section, we describe and present results from experiments run to facilitate ablation studies.

Random Solver. First, we want to test how easy it is to find failures in the six scenarios we study. We compare the MCTS solver to a naive random search solver (i.e., naive Monte Carlo) as a baseline for failure rate. Table 2 highlights these results. Note that we run each algorithm across 5 different RNG seeds to more effectively compare across different random runs. Notice that MCTS has significantly higher failure rate compared to random, which is what we would expect: we want these scenarios to have rare failures. MCTS makes a major difference in maximum log-likelihood of a failure trajectory (last column) when the random failure rate is extremely rare; looking at the crossing intersection and highway merging scenarios as examples. This highlights that AST can find more failures than random, and then maximizes the likelihood of the failure trajectories as a secondary optimization objective. Note that unless otherwise specified, these experiments are using the IDM with the learned observation model and the potential-based reward shaping function, but are not using the learned rollout policies (as that will be studied in a separate section).

Table 2: Random solver vs. MCTS results.

Scenario	Algorithm*	Failure Rate (%)	First Failure Episode†	max log(p)
Crossing intersection	Random	$0.54_{\pm 0.305}\%$	$84.0_{\pm 75.72}$	$-740.13_{\pm 679.47}$
	MCTS	$\mathbf{34.09_{\pm 24.12}\%}$	$\mathbf{58.6_{\pm 32.54}}$	$\mathbf{-206.82_{\pm 329.59}}$
Head-on T-turn	Random	$3.18_{\pm 0.44}\%$	$70.8_{\pm 56.41}$	$\mathbf{-23.77_{\pm 10.70}}$
	MCTS	$\mathbf{17.08_{\pm 21.72}\%}$	$\mathbf{35.2_{\pm 37.67}}$	$-32.57_{\pm 29.29}$
Left T-turn	Random	$10.6_{\pm 0.47}\%$	$15.2_{\pm 10.06}$	$-102.32_{\pm 34.87}$
	MCTS	$\mathbf{74.9_{\pm 4.33}\%}$	$\mathbf{7.6_{\pm 12.03}}$	$\mathbf{-44.76_{\pm 36.95}}$
Highway stopping	Random	$2.16_{\pm 0.31}\%$	$36.0_{\pm 17.61}$	$\mathbf{1.20_{\pm 1.15}}$
	MCTS	$\mathbf{8.60_{\pm 8.40}\%}$	$\mathbf{29.8_{\pm 19.65}}$	$1.18_{\pm 0.94}$
Highway merging	Random	$0.16_{\pm 0.11}\%$	$\mathbf{207.3_{\pm 213.41}}$	$-10080.16_{\pm 17398.66}$
	MCTS	$\mathbf{10.05_{\pm 15.91}\%}$	$292.3_{\pm 170.75}$	$\mathbf{-128.49_{\pm 191.29}}$
Pedestrian crosswalk	Random	$18.86_{\pm 2.28}\%$	$\mathbf{5.8_{\pm 3.56}}$	$-1.59_{\pm 1.64}$
	MCTS	$\mathbf{69.73_{\pm 11.43}\%}$	$6.4_{\pm 7.70}$	$\mathbf{-0.01_{\pm 0.52}}$

* Ran across 5 different RNG seeds using the IDM as the AV policy.
† Ran over 1000 episodes.

Potential-based reward shaping (rate). Next, we want to test the effectiveness of the potential-based reward shaping function described in section 3.1.1. Table 3 details the experimental results. Notice that including the rate in the reward function (i.e., those rows with the reward $R'(s, x, s')$ from eq. (3)) makes a small improvement to most of the failure metrics. The experiments were run

[4] `https://github.com/sisl/AutomotiveSimulator.jl`

across all scenarios, with 5 different RNG seeds for each trial (but shared RNG seeds across different reward functions for a better comparison). The failure metrics either stayed the exact same or slightly increased, with the exception in the pedestrian crosswalk scenario where the failure rate is lower but with a larger standard deviation. Further analysis will help understand the impact of using R', but these preliminary studies help indicate that including a potential-based term in the reward function can generally help performance in falsification and most likely failure analysis.

Table 3: Potential-based reward shaping results.

Scenario	Reward*	Failure Rate (%)	First Failure Episode†	$\max \log(p)$
Crossing intersection	$R(s,x)$	$34.09_{\pm 24.12}$	$58.6_{\pm 32.54}$	$-206.82_{\pm 329.59}$
	$R'(s,x,s')$	$34.09_{\pm 24.12}$	$58.6_{\pm 32.54}$	$-206.82_{\pm 329.59}$
Head-on T-turn	$R(s,x)$	$17.04_{\pm 21.76}$	$35.2_{\pm 37.67}$	$-32.57_{\pm 29.29}$
	$R'(s,x,s')$	$\mathbf{17.08_{\pm 21.72}}$	$35.2_{\pm 37.67}$	$-32.57_{\pm 29.29}$
Left T-turn	$R(s,x)$	$74.74_{\pm 4.29}$	$7.6_{\pm 12.03}$	$-56.67_{\pm 38.35}$
	$R'(s,x,s')$	$\mathbf{74.94_{\pm 4.33}}$	$7.6_{\pm 12.03}$	$\mathbf{-44.76_{\pm 36.95}}$
Highway stopping	$R(s,x)$	$7.49_{\pm 13.18}$	$29.8_{\pm 19.65}$	$0.72_{\pm 1.13}$
	$R'(s,x,s')$	$\mathbf{8.60_{\pm 8.39}}$	$29.8_{\pm 19.65}$	$\mathbf{1.18_{\pm 0.94}}$
Highway merging	$R(s,x)$	$9.33_{\pm 16.29}$	$383.6_{\pm 241.60}$	$-191.31_{\pm 180.70}$
	$R'(s,x,s')$	$\mathbf{10.05_{\pm 15.91}}$	$\mathbf{292.3_{\pm 170.75}}$	$\mathbf{-128.49_{\pm 191.29}}$
Pedestrian crosswalk	$R(s,x)$	$\mathbf{74.31_{\pm 4.41}}$	$6.4_{\pm 7.70}$	$-0.54_{\pm 0.64}$
	$R'(s,x,s')$	$69.73_{\pm 11.43}$	$6.4_{\pm 7.70}$	$\mathbf{-0.01_{\pm 0.52}}$

* Ran using MCTS across 5 different RNG seeds using the IDM as the AV policy.
† Ran over 1000 episodes.

Learned rollout policy. We study the effect of the phased approach to increase failure rate using a pretrained learned rollout policy through experimentation. This experiment also tests the different state-proxy configurations (true state, distance proxy, or rate proxy) to see how they compare when using PPO to learn the rollout policy. From table 4, the baseline MCTS run uses the standard random rollout policy and achieves a mean failure rate of about 7.5%. When we use a neural network-based solver like PPO that takes a state (or state-proxy) as input, we get about a 97−99% failure rate across the state-proxy configurations. Using that learned policy as the rollout policy for MCTS in the efficient search phase dramatically increases the failure rate from about 7.5% to about 98−100%. Notice that the maximum likelihood of the failure is much higher in the

Table 4: Learned rollout policy results.

Phase	Solver*	State-Proxy	Failure Rate (%)	First Failure Episode†	$\max \log(p)$
Baseline	MCTS	—	$7.49_{\pm 13.18}$	$29.8_{\pm 19.65}$	$0.72_{\pm 1.13}$
Offline Learning	PPO	true	99.00	4.0	−655.50
Efficient Search	MCTS	true	$98.20_{\pm 1.78}$	$1.0_{\pm 0.0}$	$0.44_{\pm 1.59}$
Offline Learning	PPO	distance	97.30	19.0	−154.60
Efficient Search	MCTS	distance	$99.10_{\pm 0.51}$	$1.0_{\pm 0.0}$	$0.27_{\pm 1.38}$
Offline Learning	PPO	rate	98.10	14.0	−791.24
Efficient Search	MCTS	rate	$100.00_{\pm 0.0}$	$1.0_{\pm 0.0}$	$-0.14_{\pm 1.10}$

* Ran across 5 different RNG seeds using the highway stopping scenario.
† Ran over 1000 episodes.

MCTS case compared to simply solving using PPO. This means that we can learn a policy that leads to failures using some state-based algorithm and transfer that policy to the MCTS setting to increase failure rate and find failures with higher likelihood.

Learned disturbance models. We study the performance of the developed disturbance model fitting technique in corresponding to the probability distribution of the actual noise in the environment state variables. To measure the fitting performance, we treat the agent state variables (position, velocity) as observed quantities with a known noise model. We experiment with different noise profiles as well as with different agents (ego vehicle, non-ego vehicle) by comparing the known noise model with the disturbance model that is fit from environment samples collected during the simulation phase. We compare the probability distributions using the KL divergence (fit from noise), symmetric KL divergence, and the cosine distance between sample probabilities. Throughout the experiment, we use a bi-modal Gaussian mixture as the disturbance model, the architecture of which is kept fixed for all experiments. For the baseline, we use Gaussian distributions with randomly generated parameters about the true noise parameters and take the average of the metrics across 5 random seeds. We observe that the learned disturbance model is significantly closer (several orders of magnitude) to the true noise model than the baseline in terms of the symmetric KL divergence and the cosine distance. Therefore, the probability of disturbance evaluated from the learned disturbance model is a good proxy for the noise probability if the actual probability distribution is not known.

Table 5: Fitting performance of the learned disturbance model.

Agents	State	Noise*	KL-divergence	Symmetric KL-divergence	Cosine distance†
Baseline	x, y	$\mathcal{N}(0,1)$	$-0.12_{\pm 0.60}$	$21.91_{\pm 8.72}$	$0.31_{\pm 0.22}$
Non-ego only	x, y	$\mathcal{N}(0,1)$	$-0.04_{\pm 0.22}$	$0.16_{\pm 0.15}$	$0.01_{\pm 0.007}$
Non-ego only	x, y	$\mathcal{N}(0,5)$	$0.29_{\pm 0.20}$	$0.09_{\pm 0.07}$	$0.00_{\pm 3e^{-4}}$
Baseline	x, y	$\mathcal{N}(0,1)$	$0.00_{\pm 1e^{-3}}$	$2.38_{\pm 3.27}$	$0.48_{\pm 0.44}$
Ego, Non-ego	x, y	$\mathcal{N}(0,1)$	$0.06_{\pm 0.11}$	$0.01_{\pm 0.08}$	$0.01_{\pm 0.01}$
Ego, Non-ego	x, y	$\mathcal{N}(0,5)$	$0.00_{\pm 2e^{-3}}$	$0.00_{\pm 5e^{-4}}$	$0.00_{\pm 4e^{-4}}$
Baseline	x, y, v	$\mathcal{N}(0,1)$	$0.00_{\pm 3e^{-5}}$	$0.57_{\pm 0.22}$	$0.74_{\pm 0.37}$
Ego, Non-ego	x, y, v	$\mathcal{N}(0,1)$	$0.01_{\pm 0.01}$	$0.00_{\pm 6e^{-3}}$	$0.02_{\pm 0.03}$
Ego, Non-ego	x, y, v	$\mathcal{N}(0,5)$	$0.00_{\pm 2e^{-5}}$	$0.00_{\pm 9e^{-5}}$	$0.00_{\pm 1e^{-4}}$

* Corresponds to each state variable. Ran across 5 different RNG seeds using the highway stopping scenario.
† Computed from 100 samples.

4.3 End-to-End Risk Assessment Study

The ultimate goal of this work is to automate the risk assessment process. This section goes through an example end-to-end risk assessment of three AV policies: the IDM, the Princeton model, and the MPC behavior model from section 3.3. We employ proposed data efficient falsification approaches (i.e., reward shaping and the phased learned rollout approach), and learn a sensor observation model for each of the AVs. The end-to-end study runs across all six driving scenarios and collects both cost and failure metrics. We compute the overall risk measure as the area under the polar curves shown in fig. 10. Each metric can be weighted individually (defaults to all ones) and for this experiment we set the weights for the three failure metrics to be $w_i = 10$ to provide more emphasis on failure rate (i.e., biased probability of failure), first failure episode (i.e., "how easy was it to find failures?"), and the maximum likelihood of failures (i.e., "how likely are the failures?"). Table 6

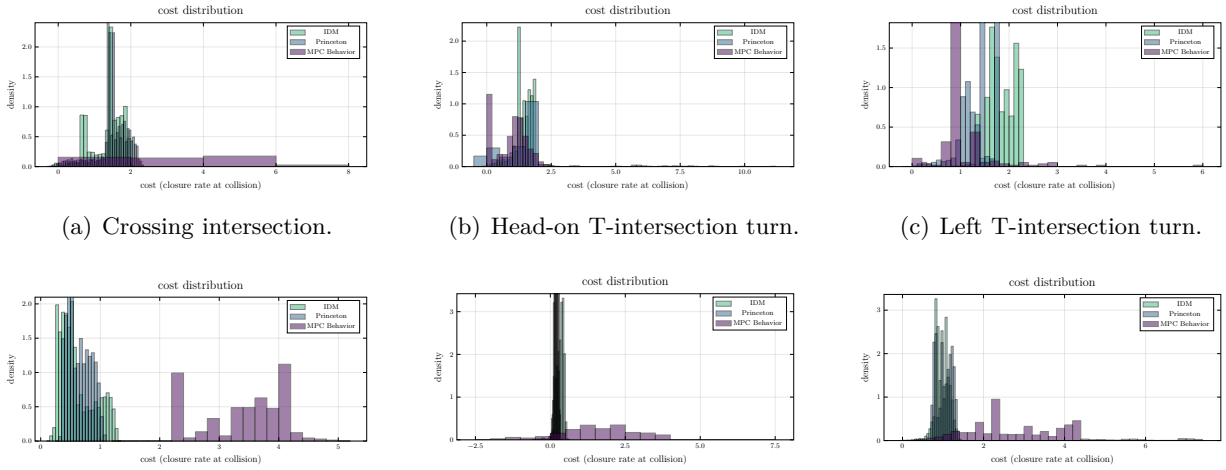
Table 6: End-to-end risk assessment results.

Scenario	AV Policy	$\mathbb{E}[Z]$	VaR	CVaR	Worst-Case	Failure Rate	FFE*	$\max \log(p)$	Risk AUC
Crossing intersection	IDM	$1.35_{\pm 0.27}$	$1.71_{\pm 0.12}$	$1.83_{\pm 0.06}$	$2.0_{\pm 0.02}$	$99.30_{\pm 0.92}$	$1.0_{\pm 0.0}$	$-256.6_{\pm 186.33}$	26.22
	Princeton	$1.48_{\pm 0.05}$	$1.8_{\pm 0.09}$	$1.99_{\pm 0.06}$	$2.22_{\pm 0.08}$	$76.62_{\pm 7.14}$	$25.80_{\pm 18.73}$	$-307.19_{\pm 200.34}$	24.51
	MPC Behavior	$3.26_{\pm 0.53}$	$4.86_{\pm 0.75}$	$5.44_{\pm 0.82}$	$6.35_{\pm 0.98}$	$1.54_{\pm 0.34}$	$42.80_{\pm 42.60}$	$-6812.82_{\pm 5896.56}$	29.87
Head-on T-turn	IDM	$1.37_{\pm 0.25}$	$1.61_{\pm 0.23}$	$1.66_{\pm 0.18}$	$1.73_{\pm 0.14}$	$27.41_{\pm 29.71}$	$110.67_{\pm 38.42}$	$-15.94_{\pm 16.69}$	14.66
	Princeton	$0.97_{\pm 0.77}$	$1.0_{\pm 0.85}$	$1.13_{\pm 0.74}$	$1.33_{\pm 0.7}$	$18.18_{\pm 17.19}$	$494.25_{\pm 281.13}$	$-591.53_{\pm 1280.35}$	17.46
	MPC Behavior	$2.46_{\pm 1.74}$	$4.13_{\pm 2.82}$	$5.28_{\pm 3.36}$	$7.85_{\pm 3.13}$	$17.20_{\pm 18.88}$	$27.60_{\pm 31.29}$	$-88.26_{\pm 122.12}$	27.62
Left T-turn	IDM	$1.85_{\pm 0.05}$	$2.16_{\pm 0.01}$	$2.22_{\pm 0.01}$	$2.3_{\pm 0.01}$	$99.66_{\pm 0.21}$	$1.0_{\pm 0.0}$	$-344.35_{\pm 174.27}$	27.57
	Princeton	$1.27_{\pm 0.28}$	$1.39_{\pm 0.26}$	$1.41_{\pm 0.25}$	$1.45_{\pm 0.23}$	$40.88_{\pm 25.70}$	$93.80_{\pm 20.17}$	$-103397.66_{\pm 152910.3}$	19.68
	MPC Behavior	$1.25_{\pm 0.44}$	$1.55_{\pm 0.8}$	$2.23_{\pm 1.34}$	$3.37_{\pm 1.82}$	$5.74_{\pm 11.06}$	$21.00_{\pm 12.92}$	$-3566.96_{\pm 3553.93}$	19.34
Highway stopping	IDM	$0.61_{\pm 0.06}$	$0.93_{\pm 0.01}$	$1.1_{\pm 0.01}$	$1.29_{\pm 0.01}$	$100.0_{\pm 0.0}$	$1.0_{\pm 0.0}$	$0.64_{\pm 1.51}$	72.69
	Princeton	$0.64_{\pm 0.02}$	$0.84_{\pm 0.01}$	$0.92_{\pm 0.0}$	$1.07_{\pm 0.0}$	$100.0_{\pm 0.0}$	$1.0_{\pm 0.0}$	$-2.43_{\pm 4.72}$	48.75
	MPC Behavior	$3.35_{\pm 0.51}$	$3.8_{\pm 0.15}$	$4.03_{\pm 0.11}$	$4.86_{\pm 0.19}$	$73.71_{\pm 18.30}$	$1.4_{\pm 0.55}$	$-106.25_{\pm 156.37}$	36.22
Highway merging	IDM	$0.3_{\pm 0.04}$	$0.42_{\pm 0.02}$	$0.46_{\pm 0.01}$	$0.57_{\pm 0.04}$	$99.96_{\pm 0.05}$	$1.0_{\pm 0.0}$	$-311.45_{\pm 248.84}$	21.65
	Princeton	$0.19_{\pm 0.04}$	$0.24_{\pm 0.03}$	$0.26_{\pm 0.02}$	$0.33_{\pm 0.03}$	$94.03_{\pm 4.5}$	$1.2_{\pm 0.45}$	$-192.2_{\pm 278.92}$	20.40
	MPC Behavior	$1.54_{\pm 0.57}$	$2.27_{\pm 1.02}$	$2.74_{\pm 0.96}$	$4.38_{\pm 1.83}$	$9.78_{\pm 9.03}$	$6.8_{\pm 7.5}$	$-4928.2_{\pm 5487.19}$	24.83
Pedestrian crosswalk	IDM	$0.97_{\pm 0.04}$	$1.11_{\pm 0.03}$	$1.19_{\pm 0.01}$	$1.34_{\pm 0.01}$	$99.86_{\pm 0.13}$	$1.0_{\pm 0.0}$	$-4.60_{\pm 3.54}$	25.71
	Princeton	$1.04_{\pm 0.05}$	$1.22_{\pm 0.02}$	$1.28_{\pm 0.02}$	$1.4_{\pm 0.02}$	$100.0_{\pm 0.0}$	$1.0_{\pm 0.0}$	$-10.39_{\pm 3.65}$	24.50
	MPC Behavior	$2.92_{\pm 0.28}$	$3.92_{\pm 0.27}$	$4.63_{\pm 0.43}$	$6.8_{\pm 0.61}$	$90.77_{\pm 3.77}$	$1.0_{\pm 0.0}$	$-2.73_{\pm 1.54}$	38.88
Overall	IDM	$1.05_{\pm 0.55}$	$1.3_{\pm 0.6}$	$1.39_{\pm 0.6}$	$1.52_{\pm 0.59}$	$87.70_{\pm 29.56}$	$12.75_{\pm 36.09}$	$-155.38_{\pm 202.86}$	74.25
	Princeton	$0.93_{\pm 0.51}$	$1.09_{\pm 0.58}$	$1.17_{\pm 0.6}$	$1.3_{\pm 0.63}$	$71.62_{\pm 34.14}$	$89.34_{\pm 192.09}$	$-17416.9_{\pm 68955.41}$	48.53
	MPC Behavior	$2.46_{\pm 1.12}$	$3.42_{\pm 1.66}$	$4.06_{\pm 1.89}$	$5.6_{\pm 2.21}$	$33.12_{\pm 37.70}$	$16.77_{\pm 25.63}$	$-2584.2_{\pm 4263.2}$	39.01

* First failure episode.

details the results from running 5 different RNG seeds and highlights the "best" AV policy for each metric in green. Note that unlike the results from the ablation studies, a lower failure rate and lower maximum log-likelihood is better (i.e., less risky), while a higher "ease" of failure metric is better (i.e., harder to fail). The last row in table 6 combines the metrics across all of the scenarios to get an overall measure of risk.

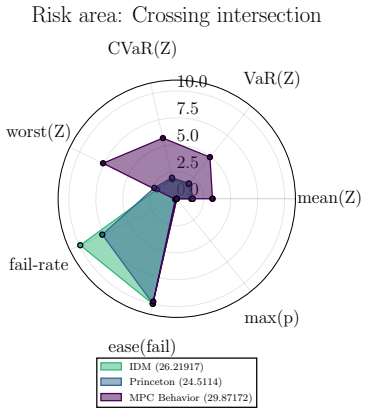
Breaking down the risk into the individual scenarios highlights where certain AV policies may have difficulties. Using this type of assessment during AV development can also provide an opportunity to find and resolve any system problems before deployment. The cost distributions for each AV are illustrated in fig. 9 to show where the AVs fail across different closure rates (i.e., costs). Evident from these distributions is an increased cost for the MPC behavior policy which may be attributed to the lack of a constraint on the maximum acceleration (where the IDM has an acceleration maximum and the Princeton model has a velocity maximum). There is room to



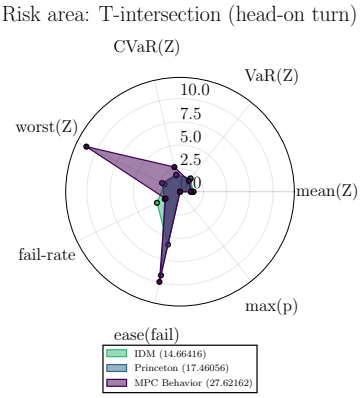
(a) Crossing intersection. (b) Head-on T-intersection turn. (c) Left T-intersection turn.

(d) Highway stopping. (e) Highway merging. (f) Pedestrian crosswalk.

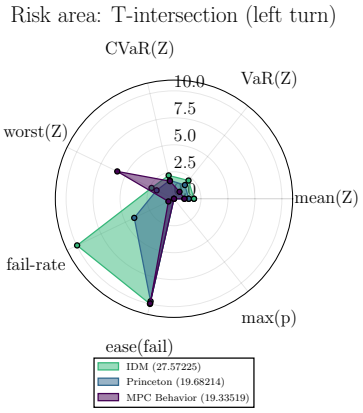
Figure 9: Cost distributions of each AV policy over the six driving scenarios.



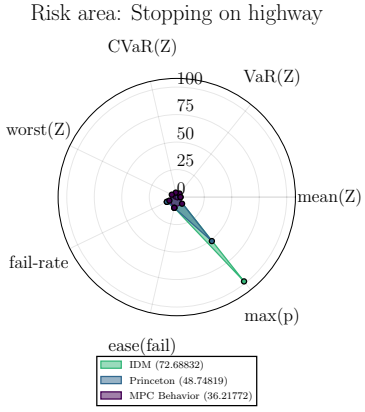
(a) Crossing intersection.



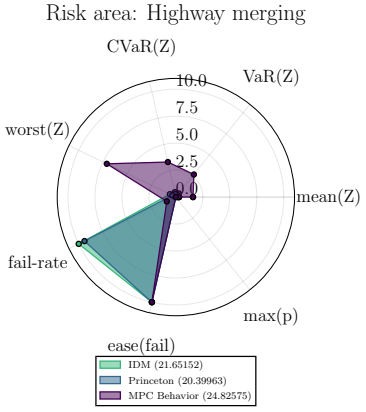
(b) Head-on T-intersection turn.



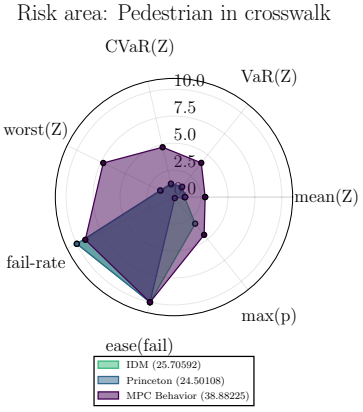
(c) Left T-intersection turn.



(d) Highway stopping.



(e) Highway merging.



(f) Pedestrian crosswalk.

Figure 10: Polar plots of the total risk for each driving scenario. A weight of 10 is applied to each failure metric.

tune the MPC agent through hyperparameters based on these results, which also emphasizes the use of this type of automated assessment during development. We left the MPC agent "un-tuned" for an example where one AV policy (the MPC agent) has higher cost metrics but lower failure metrics—thus, providing a useful example of balancing metrics using the risk weights. The wider distribution for the MPC agent can be seen in the cost distributions and in the higher cost metrics in table 6. Balancing the cost of a failure with how often failures occur is the reasoning behind the weighted risk measure. Selecting weights of 10 for the failure metrics puts more emphasis on how likely the system is to fail, and then incorporating the cost metrics when it *does* fail. Polar plots that qualitatively represent the AV risk are shown in fig. 10 (using the previously described weights). The risk area under the curves (AUC) is shown in the figure legends where larger values means riskier. Dominance of one AV curve over another is a quick indication that the dominating policy is riskier. The high cost for the MPC agent (in purple) is visible in the polar plots and the high failure statistics are also evident in the IDM and Princeton policies (in green and blue, respectively). Because we want higher to represent riskier in each of these risk metrics, we define the "ease of failing" as $(N - F)/N$, where F is the first failure episode and N is the total number of episodes. We also use the likelihood (rather than log-likelihood) in the polar plots to ensure all metrics are non-negative.

Focusing on the risk over all scenarios, we can also see the higher cost trend when using the

MPC behavior agent with the IDM and Princeton models exhibiting similar distributions in the aggregate (shown in fig. 11). Looking at the polar plots over all scenarios, fig. 12(a) illustrates the risk with the adjusted failure weights and fig. 12(b) illustrates the risk with uniform weights (i.e., all set to one). Figure 12(b) clearly highlights the higher cost metrics when using the MPC agent and higher failure metrics when using the IDM or Princeton models, illustrating the importance of having control over the weighting scheme.

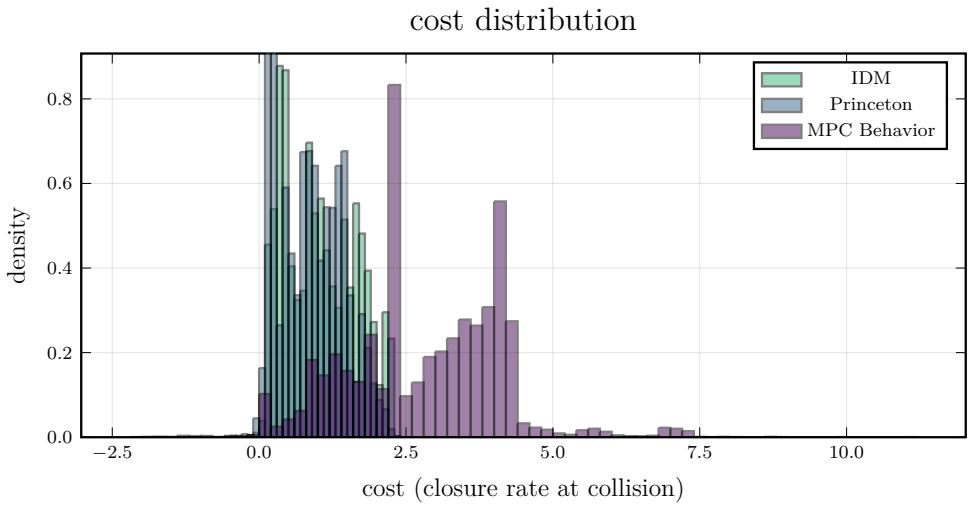
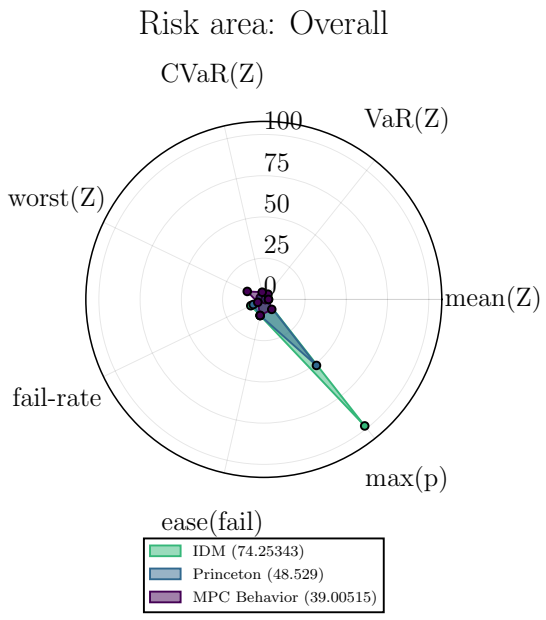
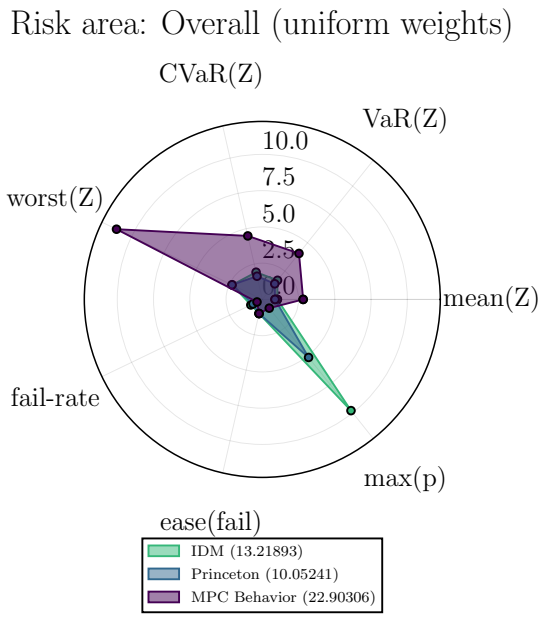


Figure 11: Cost distribution over all six driving scenarios.



(a) Risk using weights of 10 applied to mean(Z), max(p), and ease(fail).



(b) Risk using uniform weights (all ones).

Figure 12: Risk area over all six driving scenarios using separate weights.

5 Conclusion and Future Work

This work presents an end-to-end autonomous vehicle risk assessment framework to automate the validation process. The intention is to compare different AV policies across stressing driving scenarios and quantify their risk across several important metrics. We use established risk metrics over a cost distribution combined with failure metrics found through adaptive stress testing. Balancing these metrics using weights provides developers or policy makers with control over the relative importance of each risk metric in the total measured risk. The risk framework is open-source with a modular design to allow for "plug-in-play" components ranging from driving scenarios, sensor observation models, AV policies, validation methods, and risk assessment metrics. We introduce a method to learn surrogates of complex or unknown probability distributions of the sensor observation models to extend the black-box assumption around the simulation environment itself, not just the AV system under test. We propose several modifications to adaptive stress testing for more efficient failure searches; including a rate component in the reward function and employing a two-stage learned rollout method for Monte Carlo tree search. Comparing against two lead-car following driver models (namely, the IDM and Princeton model), we have also developed a driving behavior agent using model predictive control to compare against more realistic AV policies. Ablation studies were performed to test the individual components we propose and their relative affect on performance. An example end-to-end risk assessment study compared the three AV policies across six driving scenarios to compute individual risk for each scenario and an overall risk across all scenarios. The modularity of the developed framework can be extended to higher-fidelity simulators and more realistic AV policies, hence the research focus on data efficient falsification and sensor observation modeling in lieu of more computationally expensive simulators and systems.

Future work will extend this framework from low-fidelity simulators to high-fidelity 3D simulators like CARLA [43] and to more realistic AV policies like Comma AI's OpenPilot.[5] Further extending the data efficiency of adaptive stress testing can be explored; either through modifications to the problem formulation itself or to existing reinforcement learning algorithms. Further analysis of the potential-based reward shaping approach and how the magnitude of the rate value may help speed up learning would also be useful. Normalization schemes applied to each risk metric could also be investigated so the metrics can be weighted and scaled relative to other metrics. Finally, other cost models besides closure rate could be applied in simulation to further assess AV risk under a different definition of cost (e.g., property damage).

[5] https://github.com/commaai/openpilot

A Software Tools

This section describes the open-source software tools produced as part of this research.

A.1 AutonomousRiskFramework

The end-to-end autonomous vehicle risk assessment framework is collected into the `AutonomousRiskFramework`[6] package. This high-level collection of packages serves as the main repository for documentation and example notebooks. The following four main Julia packages have been developed to support this work and are included in the `AutonomousRiskFramework` repository.

A.1.1 RiskSimulator.jl

The `RiskSimulator.jl` package is designed to set up and run the risk assessment process, linking all of the packages together. The package is built off of `AutomotiveSimulator.jl`[7], `AdversarialDriving.jl`[8], and `POMDPStressTesting.jl`[9] [13] to build a falsification-based simulator around different driving scenarios and AV policies. The driving scenarios studied in this work are included in this package, along with all of the code used in the experiments. Plotting tools for risk assessment and analysis are included, along with code to compute the cost-based risk metrics and AUC measures used in this work.

A.1.2 ObservationModels.jl

The `ObservationModels.jl` package contains tools for fitting a probabilistic model of the disturbances in the simulation environment. The package relies on the interface from `AutomotiveSimulator.jl` and `AdversarialDriving.jl` to simulate and collect several scenes containing the noisy environment state. Using the collected scenes, the package contains methods and examples to construct a disturbance model using a Mixture Density Network or using moment matching. Methods for simulating sensor observations (e.g., GPS pseudorange and range-bearing measurements) with associated noise models are included in the package, along with examples for visualizing disturbances and landmark objects in the environment (such as buildings).

A.1.3 IntelligentDriving.jl

The `IntelligentDriving.jl` package allows constructing an MPC driver agent within the large AST framework by specifying an arbitrary objective function that the agent should attempt to minimize. The agent then minimizes a finite horizon optimal control plan at every time step and executes the first action from the plan—attempting to drive optimally according to the specified objective function.

In the interest of computational speed, a separate automatic differentiation package is provided (`SpAutoDiff.jl`) which computes the sparse first and second order derivatives of objectives specified using its atoms. A wide range of functions (not necessarily convex) are implemented.

Although `IntelligentDriving.jl` does not allow specifying state and action constraints explicitly, exact satisfaction of these can be guaranteed by specifying the constraints within the objective function via exact penalty functions [44].

[6] `https://github.com/sisl/AutonomousRiskFramework`
[7] `https://github.com/sisl/AutomotiveSimulator.jl`
[8] `https://github.com/sisl/AdversarialDriving.jl`
[9] `https://github.com/sisl/POMDPStressTesting.jl`

A.1.4 STLCG.jl

The `STLCG.jl` package allows specifying the human-interpretable Signal Temporal Logic (STL) atoms and computing their robustness score. This can then be used either within `AutonomousRiskFramework` to specify the undesirable occurrences within the scene (indicated by a low or negative robustness score) or as an objective for a MPC agent specified via `IntelligentDriving.jl`. Because `IntelligentDriving.jl` relies on general nonlinear derivative based optimization (and some STL atoms generate sparse derivative information) the STL formulas can be relaxed, replacing low-level `max` and `min` operators with their smooth approximation defined conveniently via a guaranteed overapproximator in the form of the `logsumexp` operator with a tunable smoothness parameter (where large values of the parameter indicate an almost exact approximation).

A.2 CARLA Integration

The objective of the CARLA integration is to leverage the high-fidelity sensor observations and driving models available in the CARLA simulator for stress testing. By directly searching for failures within CARLA, AST can assess the risk associated with realistic driving scenarios and with the algorithms for perception and control. Furthermore, the integration leverages the `ScenarioRunner` package in CARLA to simulate various NHTSA-inspired pre-crash scenarios, thereby improving the risk assessment. The integration between the CARLA Python interface and AST Julia interface is enabled using `PyCall.jl` and `PyJulia` packages. For efficiency, the interface is implemented in two phases; the first phase executes the stress testing in CARLA using a specified model of the disturbances and the second phase generates and records the sensor observations corresponding to a specified trajectory. The first phase does not necessarily require sensor observations to be simulated or the environment to be rendered, and therefore can be executed at 10 times the real-time speed. The input to the second phase are the trajectories encountered by AST during the failure search, for which several sensor observations are generated asynchronously and noisy environment states are computed. In the next steps of this integration, these sensor observations and environment

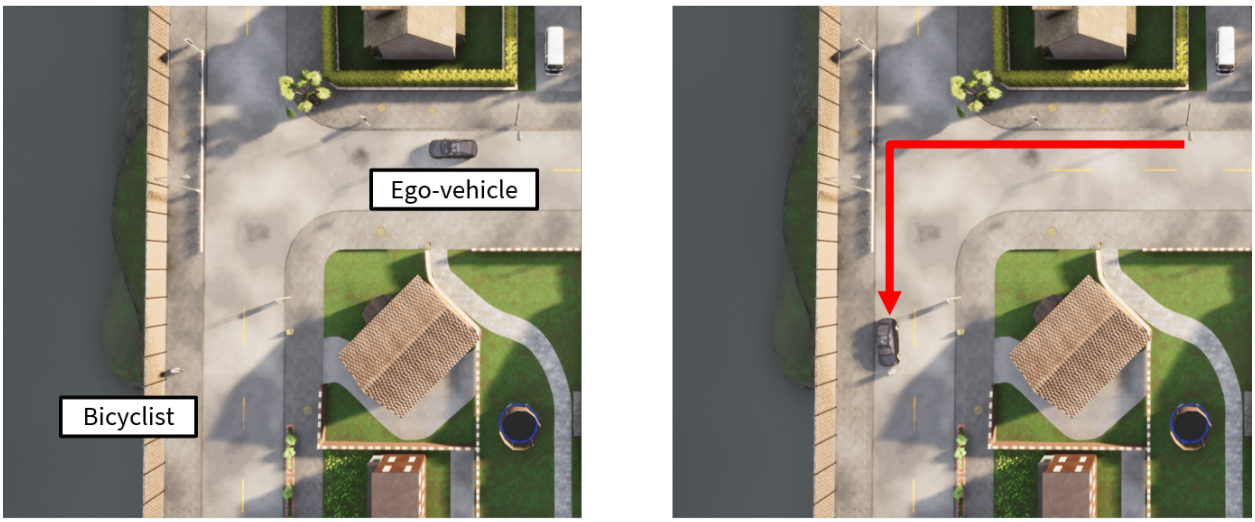


Figure 13: An example failure discovered in the CARLA simulator. The ego vehicle uses the `BaseAgent` class for choosing the control actions and observes the noisy position and velocity of the bicyclist. AST finds a likely sequence of disturbances added to the position and velocity in the simulation that causes the ego vehicle to hit the bicyclist.

states will be utilized in constructing a new disturbance model using the `ObservationModels.jl` package, and repeat the first phase with the updated disturbance model to improve the accuracy of the assessed risk. Figure 13 shows an example of a failure discovered in CARLA using AST.

B Demonstration: End-to-End Risk Assessment

This section illustrates an example end-to-end risk assessment when using the `AutonomousRiskFramework`. To load the framework code into a Julia session, we run:

```
using RiskSimulator
```

Next we choose which AV policy we want to stress test. In this case, we use the IDM as our SUT.

```
system = IntelligentDriverModel()
```

We select a scenario from the `SCENARIO` enumeration, where we chose the highway merging scenario here as an example.

```
scenario = get_scenario(MERGING)
```

We then set up the AST planner, passing in the specific SUT and scenario.

```
planner = setup_ast(sut=system, scenario=scenario)
```

Now we can run the failure search, which will automatically collect necessary metrics.

```
search!(planner)
```

To access the failure and risk metrics, we can pull out the necessary metrics for further analysis.

```
α = 0.2 # risk tolerance
fail_metrics = failure_metrics(planner)
cost_metrics = risk_assessment(planner, α)
```

Finally, we can plot metrics (not shown) and calculate the overall risk (with optional weights $\mathbf{w}$).

```
risk = overall_area(planner, weights=w, α=α)
```

C Signal Temporal Logic (STL)

Signal Temporal Logic (STL) defines a continuous satisfaction robustness score for Linear Temporal Logic (LTL) formulas. STL and LTL operate on the idea of a signal. Below is a summary based on an excellent description of STL in [45].

Definition 1 (Signal)**.** *A signal* $s_{t_0} = (x_0, t_0), (x_1, t_1), \ldots, (x_T, t_T)$ *is an ordered* $(t_{i-1} < t_i)$ *finite sequence of states* $x_i \in \mathbb{R}^n$ *and their associated times* $t_i \in \mathbb{R}$*. For ease of notation,* s *(i.e., when the subscript on* s_t *is dropped) denotes the entire signal.*

STL allows specifying requirements in a human-interpretable way over autonomous agents trajectories by capturing things like

- ***Always*** *avoid collisions with other vehicles.*
- *Do not leave your lane, to overtake,* ***until*** *the oncoming lane is free from traffic.*
- ***Eventually*** *reach goal A* ***or*** *goal B* ***and always*** *avoid collisions with other vehicles.*

STL formulas are defined recursively according to the following grammar [45]:

$$\phi ::= \quad \top \mid \mu_c \mid \neg\phi \mid \phi \wedge \psi \mid \phi \mathcal{U}_{[a,b]}\, \psi. \tag{6}$$

The symbols are: $\neg$ (negation/not), $\wedge$ (conjunction/and), logical connectives, and $\mathcal{U}$ (until) is a temporal operator. The time range for temporal operator is denoted in the subscript and, when omitted, is the positive ray $[0, \infty)$. Other commonly used logical connectives are $\vee$ (disjunction/or) and $\implies$ (implies) and temporal operators $\Diamond$ (eventually) and $\Box$ (always) and can be defined as follows:

$$\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi), \qquad \phi \Rightarrow \psi = \neg\phi \vee \psi, \qquad \Diamond_{[a,b]}\, \phi = \top\, \mathcal{U}_{[a,b]}\, \phi, \qquad \Box_{[a,b]}\, \phi = \neg\Diamond_{[a,b]}(\neg\phi).$$

Formally, the Boolean (true or false) value w.r.t. to a signal s_t is defined as follows:

$$\begin{aligned}
s_t \models \mu_c \quad &\Leftrightarrow \quad \mu(x_t) > c \\
s_t \models \neg\phi \quad &\Leftrightarrow \quad \neg(s_t \models \phi) \\
s_t \models \phi \wedge \psi \quad &\Leftrightarrow \quad (s_t \models \phi) \wedge (s_t \models \psi) \\
s_t \models \phi \vee \psi \quad &\Leftrightarrow \quad (s_t \models \phi) \vee (s_t \models \psi) \\
s_t \models \phi \Rightarrow \psi \quad &\Leftrightarrow \quad \neg(s_t \models \phi) \vee (s_t \models \psi) \\
s_t \models \Diamond_{[a,b]}\phi \quad &\Leftrightarrow \quad \exists t' \in [t+a, t+b] \;\text{ s.t. } s_{t'} \models \phi \\
s_t \models \Box_{[a,b]}\phi \quad &\Leftrightarrow \quad \forall t' \in [t+a, t+b] \;\; s.t.\; s_{t'} \models \phi \\
s_t \models \phi\, \mathcal{U}_{[a,b]}\, \psi \quad &\Leftrightarrow \quad \exists t' \in [t+a, t+b] \;\; s.t.\; (s_{t'} \models \psi) \;\wedge\; (s_t \models \Box_{[0,t']}\phi)
\end{aligned}$$

STL differs from LTL in that it defines a continuous robustness score ρ, a measure of "how much" a given formula is satisfied (with positive values indicating Boolean value of true and negative values indicating Boolean false). This takes the following form:

$$\begin{aligned}
\rho(s_t, \top) &= \rho_{\max} \qquad \text{where } \rho_{\max} > 0 \\
\rho(s_t, \mu_c) &= \mu(x_t) - c \\
\rho(s_t, \neg\phi) &= -\rho(s_t, \phi) \\
\rho(s_t, \phi \wedge \psi) &= \min(\rho(s_t, \phi), \rho(s_t, \psi)) \\
\rho(s_t, \phi \vee \psi) &= \max(\rho(s_t, \phi), \rho(s_t, \psi)) \\
\rho(s_t, \phi \Rightarrow \psi) &= \max(-\rho(s_t, \phi), \rho(s_t, \psi)) \\
\rho(s_t, \Diamond_{[a,b]}\phi) &= \max_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi) \\
\rho(s_t, \Box_{[a,b]}\phi) &= \min_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi) \\
\rho(s_t, \phi\, \mathcal{U}_{[a,b]}\, \psi) &= \max_{t' \in [t+a, t+b]} (\min(\rho(s_{t'}, \psi), \min_{t'' \in [0, t']} \rho(s_{t''}, \phi)))
\end{aligned}$$

D Environment Noise

Table 7 details the probability distributions for the noise disturbances added to both the ego and other vehicle (or pedestrian).

Table 7: Environment distributions of noise disturbances.

Scenario	Noise Distribution: xy-position	Noise Distribution: velocity
Crossing intersection	$\mathcal{N}(0, 10)$	$\mathcal{N}(0, 2)$
Head-on T-turn	$\mathcal{N}(0, 10)$	$\mathcal{N}(0, 2)$
Left T-turn	$\mathcal{N}(0, 10)$	$\mathcal{N}(0, 2)$
Highway stopping	$\mathcal{N}(0, 2)$	$\mathcal{N}(0, 1e^{-4})$
Highway merging	$\mathcal{N}(0, 3)$	$\mathcal{N}(0, 1)$
Pedestrian crosswalk	$\mathcal{N}(0, 2)$	$\mathcal{N}(0, 0.1)$

E Proposal: Flagship Research Project

For reference, this section details the original proposal for the flagship project.

E.1 Background

An approach known as adaptive stress testing (AST) has recently been used to find the most likely failures in aircraft collision avoidance systems [46], [47] and autonomous vehicles [48], [49]. AST was designed for safety-critical black-box systems, and thus requires minimal interaction with the system under test. Lee, Kochenderfer, Mengshoel, *et al.* [46] applied AST to aircraft collision avoidance systems using a modified Monte Carlo tree search algorithm to control the random number generator seed used by the system's simulation environment. By only controlling the seed of an otherwise intractable problem, AST found likely failure events in a system where failures are extremely rare. Koren, Alsaif, Lee, *et al.* [48] applied AST to white-box autonomous vehicles using deep reinforcement learning as the solver and extended their approach to black-box autonomous vehicles using recurrent neural networks [49]. Koren and Kochenderfer [50] also propose a method to learn a heuristic reward using the go-explore algorithm, which otherwise can be difficult or infeasible to construct from domain knowledge.

To better understand failure events, approaches have been proposed to find human-interpretable failures [51] and to cluster high-level failures types into categories [52]. Corso and Kochenderfer [51] use a grammar based on signal temporal logic (STL), which is commonly used for falsification, to describe the failures and use genetic programming to optimize for high likely failure events. Their approach results in human-interpretable logical statements that describe the failures (e.g. "between 0 and 5 seconds the blinker is always on"). Lee, Kochenderfer, Mengshoel, *et al.* [52] propose a similar grammar-based approach for interpretability and also provide a categorization of failure events based on common behaviors described by the grammar. Their work aims to provide more useful descriptions of failure events to be addressed by the engineers and designers of the system under test. Other applications of AST include pairwise differential stress testing [47] and domain-relevant reward augmentation [53]. To stress test two different systems, Lee, Mengshoel, Saksena, *et al.* [47] propose a pairwise approach for differential adaptive stress testing (DAST). Their approach uses reinforcement learning to encourage failures in one system and not the other to maximize the difference in their outcomes. DAST is useful when assessing a system relative to another baseline system and can also be used for regression testing. Corso, Du, Driggs-Campbell,

et al. [53] encode domain relevant information into the AST reward function to encourage finding a larger and more expressive set of failures (to avoid exploiting repeatedly discovered failures).

One critical category of failure events is sensing and perception failures. An autonomous vehicle is equipped with an array of sensors for perception, such as cameras, GPS units, inertial navigation units, LiDARs, radars and transceivers for communication with other vehicles on the road. To validate image-based neural network controllers, Julian, Lee, and Kochenderfer [54] use AST to apply disturbances to input images to find failures in an aircraft taxiing system. Their work addresses the validation of the multi-step properties in neural network controllers. Shetty and Gao [55] proposed an image-based cross-view geolocalization method for pose estimation with the aid of georeferenced satellite imagery. The approach consists of two Siamese neural networks that extract relevant features despite large differences in viewpoints, and integrates the crossview geolocalization output with visual odometry through a Kalman filter. In addition to images, there is also a large body of existing work by on LiDAR faults [56], [57], GPS faults [58], [59] and faults of sensor fusion [60].

E.2 Research Proposal

E.2.1 Overview

To perform autonomous vehicle risk assessment over black-box systems, we propose a framework for defining safety requirements through temporal logic specifications and using adaptive stress testing to validate each system under test. With this approach, we can define the failure events to search for (e.g., collisions) and the events that led to a failure (e.g., sensor failures) as human-interpretable temporal logic specifications. To better model real-world risk, our safety validation framework will account for realistic sensor uncertainties in the simulation, as well as interactions with other agents on the road (e.g., human-driven vehicles). Given the computational tractability of the risk assessment problem, we propose speed ups to adaptive stress testing by combining reinforcement learning techniques with local optimization methods.

E.2.2 Scope

The extent of this work is to build a cohesive risk assessment framework that is agnostic to the black-box autonomous vehicles under test. We will use off-the-shelf intelligent driver models [61] as our black-box autonomous vehicle systems and high-fidelity simulation environments like CARLA [62] for realistic integration of physical perception sensors. The design of the modular framework can easily include additional sensor types, agent behavioral models (e.g., Trajectron++ [63]), autonomous vehicle systems, and driving scenarios into the simulation.

E.2.3 Goals

Our objective is to provide a realistic framework for autonomous vehicle risk assessment. We aim to use adopted techniques from the safety validation literature, including adaptive stress testing, temporal logic specification robustness, agent behavioral models, and sensor uncertainty models. We will also design and develop algorithms to improve the computational tractability of the risk assessment.

E.3 Task Descriptions

Task 1: AST Modeling Framework

To expand on our existing research, we would like to incorporate STL robustness measurements into the AST reward function. For system requirements described using STL, robustness is a measure of how close the requirement is to being violated [64]. Current approaches in the falsification literature use robustness to guide their search [18]. To assess autonomous vehicle risk, system requirements can be specified in STL and we can use AST to find likely failures. These system requirements can be common across AV platforms, thus allowing us to appropriately compare any number of black-box systems. Combining the interpretability work described above with STL robustness, we could automate the process of finding measurable failures that are also human-interpretable.

Task 2: Uncertainty Models

We will aid AST with real-world perception as well agent behavioral uncertainties. Existing work such as [51] makes certain assumptions about the sensing noise profile (e.g. uniformly distributed in $[0, 1]$). We will extend the existing work with a real-world sensing profile, which includes not only nominal cases that form a multi-modal distribution, but also outlier characteristics. The more realistic noise profile will enable more accurate risk assessment, and thus higher level of safety. Another key source of uncertainty is represented by the presence of sentient agents (e.g., human-driven vehicles, pedestrian, etc.) on the road. We will again leverage STL, along with data-driven techniques (e.g., [63]), to develop high-quality simulation agents for the purposes of AST and risk assessments. This task will leverage our recent breakthroughs on imbuing logical structure into high-capacity, learning-based representations of human behaviors through a tool referred to as `stlcg` [45] .

Task 3: Computational Tractability

We would also like to propose to speed up the AST process. One proposed approach is extending AST to use a two-layered approach similar to Zhang, Ernst, Sedwards, *et al.* [65]. This approach splits the failure event search and the most likely failure event search into phases. The two-layered approach uses optimization techniques to search a local action-space and the result is fed to the higher-level reinforcement learner. Furthermore, we will apply prior knowledge about sensing and perception uncertainties in the autonomous vehicle's physical state to search for the most likely failure event first. For example, if an autonomous vehicle is driving west during sunset time, image failures due to sun glare may be the main failure cause and should be searched first. The goal is to help speed up falsification (i.e. finding failures) so the reinforcement learner can explore the failure-space for likely cases. Applying this technique to assess autonomous vehicle safety would ideally speed up the process and assess risk across a more diverse set of failures.

E.4 Timeline

	October	November	December
Task 1	Design and development of modular framework	Incorporate STL specifications	Incorporate robustness calculations
Task 2	Implement simple sensor models	Implement outlier detection	Devise STL-based simulation agents
Task 3	Design two-layer approach into framework	Implement local optimization method(s)	Test optimization methods
	January	February	March
Task 1	Incorporate STL specification interpretability	Test STL interpretability	Perform massive risk assessment tests
Task 2	Model GPS multipath effects and outlier measurement using a high-fidelity GPS simulator	Model vision-based perception and sensing with CARLA	Augment CARLA with realistic GPS sensing measurements and STL-based simulation agents
Task 3	Include IDM into framework	Interface with CARLA	Test end-to-end risk assessment
	April	May	June
Task 1	STL interpretability data collection	Analysis of STL failures	Formalize work into paper
Task 2	Expand to multi-modal sensing profiles including both vision and GPS	Port `stlcg` to Julia	Formalize work into paper
Task 3	Two-layered approach data collection	Analysis of RL/ optimization methods	Formalize work into paper

E.5 Deliverables

We anticipate the following deliverables:

1. A peer-reviewed conference paper per task submitted no later than July 15, 2021.
2. A final report on all tasks by July 15, 2021.
3. All general-purpose software will be made open source under a non-restrictive license (e.g., MIT or BSD) by July 15, 2021.

E.6 Researchers

1. Mykel Kochenderfer and Robert Moss, Stanford Intelligent Systems Laboratory
2. Grace Gao and Shubh Gupta, Stanford Navigation and Autonomous Vehicles Laboratory
3. Marco Pavone and Robert Dyro, Stanford Autonomous Systems Laboratory

References

[1] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive Stress Testing for Autonomous Vehicles," *arXiv:1902.01909 [cs, stat]*, Feb. 2019, arXiv: 1902.01909. [Online]. Available: `http://arxiv.org/abs/1902.01909` (visited on 02/18/2021).

[2] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An analysis of adversarial attacks and defenses on autonomous driving models," in *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, IEEE, 2020, pp. 1–10.

[3] C. Yan, W. Xu, and J. Liu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle," *Def Con*, vol. 24, no. 8, p. 109, 2016.

[4] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial sensor attack on lidar-based perception in autonomous driving," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2267–2281.

[5] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components," *arXiv:1804.06760 [cs]*, Jan. 2019, arXiv: 1804.06760. [Online]. Available: `http://arxiv.org/abs/1804.06760` (visited on 02/15/2021).

[6] W. G. Najm, J. D. Smith, M. Yanagisawa, *et al.*, "Pre-crash scenario typology for crash avoidance research," United States. National Highway Traffic Safety Administration, Tech. Rep., 2007.

[7] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, no. 2, p. 1805, 2000.

[8] G. F. Newell, "Nonlinear effects in the dynamics of car following," *Operations research*, vol. 9, no. 2, pp. 209–229, 1961.

[9] R. Lee, O. J. Mengshoel, A. Saksena, R. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, "Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning," *arXiv:1811.02188 [cs]*, Dec. 2020, arXiv: 1811.02188. [Online]. Available: `http://arxiv.org/abs/1811.02188` (visited on 02/15/2021).

[10] D. Richards, "Relationship between speed and risk of fatal injury: Pedestrians and car occupants," 2010.

[11] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *Robotics Research*, Springer, 2020, pp. 75–84.

[12] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[13] R. J. Moss, "POMDPStressTesting.jl: Adaptive stress testing for black-box systems," *Journal of Open Source Software*, vol. 6, no. 60, p. 2749, 2021.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on robot learning*, PMLR, 2017, pp. 1–16.

[15] R. J. Moss, R. Lee, N. Visser, J. Hochwarth, J. G. Lopez, and M. J. Kochenderfer, "Adaptive stress testing of trajectory predictions in flight management systems," *Digital Avionics Systems Conference (DASC)*, 2020.

[16] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning (ICML)*, vol. 99, 1999, pp. 278–287.

[17] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*, Springer, 2006, pp. 72–83.

[18] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, *A survey of algorithms for black-box safety validation*, 2020. arXiv: 2005.02979.

[19] L. Mathesen, S. Yaghoubi, G. Pedrielli, and G. Fainekos, "Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart," English (US), in *2019 IEEE 15th International Conference on Automation Science and Engineering, CASE 2019*, IEEE Computer Society, Aug. 2019, pp. 991–997. [Online]. Available: https://asu.pure.elsevier.com/en/publications/falsification-of-cyber-physical-systems-with-robustness-uncertain (visited on 02/15/2021).

[20] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski, "Multiple shooting, cegar-based falsification for hybrid systems," in *Proceedings of the 14th International Conference on Embedded Software*, 2014, pp. 1–10.

[21] Z. Huang, H. Lam, D. J. LeBlanc, and D. Zhao, "Accelerated evaluation of automated vehicles using piecewise mixture models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 9, pp. 2845–2855, 2017, Publisher: IEEE.

[22] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic over Real-Valued Signals," en, in *Formal Modeling and Analysis of Timed Systems*, K. Chatterjee and T. A. Henzinger, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2010, pp. 92–106, ISBN: 978-3-642-15297-9.

[23] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius, *Functional Gradient Descent Method for Metric Temporal Logic Specifications*.

[24] J. Li and D. Xiu, "Evaluation of failure probability via surrogate models," *Journal of Computational Physics*, vol. 229, no. 23, pp. 8966–8980, 2010, Publisher: Elsevier.

[25] N.-C. Xiao, H. Zhan, and K. Yuan, "A new reliability method for small failure probability problems by combining the adaptive importance sampling and surrogate models," en, *Computer Methods in Applied Mechanics and Engineering*, vol. 372, p. 113 336, Dec. 2020, ISSN: 0045-7825. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045782520305211 (visited on 02/15/2021).

[26] R. Pulch, "Polynomial chaos for the computation of failure probabilities in periodic problems," in *Scientific Computing in Electrical Engineering SCEE 2008*, Springer, 2010, pp. 191–198.

[27] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[28] A. Wardzinski, "Dynamic risk assessment in autonomous vehicles motion planning," in *2008 1st International Conference on Information Technology*, IEEE, 2008, pp. 1–4.

[29] P. Bhavsar, P. Das, M. Paugh, K. Dey, and M. Chowdhury, "Risk analysis of autonomous vehicles in mixed traffic streams," *Transportation Research Record*, vol. 2625, no. 1, pp. 51–61, 2017.

[30] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, pp. 1–14, 2014.

[31] M. Strickland, G. Fainekos, and H. B. Amor, "Deep predictive models for collision risk assessment in autonomous driving," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 4685–4692.

[32] C. Chen, X. Liu, H.-H. Chen, M. Li, and L. Zhao, "A rear-end collision risk evaluation and control scheme using a bayesian network model," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 264–284, 2018.

[33] A. Wang, X. Huang, A. Jasour, and B. Williams, "Fast risk assessment for autonomous vehicles using learned models of agent futures," *arXiv preprint arXiv:2005.13458*, 2020.

[34] D. Shannon, L. Rizzi, F. Murphy, and M. Mullins, "Exploring the price of motor vehicle collisions–a compensation cost approach," *Transportation research interdisciplinary perspectives*, vol. 4, p. 100 097, 2020.

[35] R. J. Moss, "Algorithms for efficient validation of black-box systems," M.S. thesis, Stanford University, 2021.

[36] A. E. Haas, *Introduction to Theoretical Physics Vols. I & II*. D. Van Nostrand, 1925.

[37] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897.

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv*, no. 1509.02971, 2015.

[41] C. M. Bishop, "Mixture density networks," 1994.

[42] F. M. Favarò, N. Nader, S. O. Eurich, M. Tripp, and N. Varadaraju, "Examining accident reports involving autonomous vehicles in California," *PLoS one*, vol. 12, no. 9, e0184952, 2017.

[43] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, PMLR, 2017, pp. 1–16.

[44] S. Han and O. L. Mangasarian, "Exact penalty functions in nonlinear programming," *Mathematical programming*, vol. 17, no. 1, pp. 251–269, 1979.

[45] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through STL specifications: Infusing logical structure into gradient-based methods," in *Workshop on Algorithmic Foundations of Robotics*, 2020.

[46] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *Digital Avionics Systems Conference (DASC)*, IEEE, 2015, pp. 6C2–1.

[47] R. Lee, O. J. Mengshoel, A. Saksena, R. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, *Adaptive stress testing: Finding likely failure events with reinforcement learning*, 2018. arXiv: 1811.02188.

[48] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1–7.

[49] M. Koren and M. J. Kochenderfer, "Efficient autonomy validation in simulation with adaptive stress testing," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2019, pp. 4178–4183.

[50] ——, "Adaptive stress testing without domain heuristics using go-explore," *arXiv*, no. 2004.04292, 2020.

[51] A. Corso and M. J. Kochenderfer, "Interpretable safety validation for autonomous vehicles," *arXiv*, no. 2004.06805, 2020.

[52] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, and J. Silbermann, "Interpretable categorization of heterogeneous time series data," in *Proceedings of the 2018 SIAM International Conference on Data Mining*, SIAM, 2018, pp. 216–224.

[53] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validation," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2019, pp. 163–168.

[54] K. D. Julian, R. Lee, and M. J. Kochenderfer, "Validation of image-based neural network controllers through adaptive stress testing," *arXiv*, no. 2003.02381, 2020.

[55] A. Shetty and G. X. Gao, "UAV pose estimation using cross-view geolocalization with satellite imagery," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1827–1833.

[56] A. V. Kanhere and G. X. Gao, "LiDAR SLAM utilizing normal distribution transform and measurement consensus," in *Proceedings of ION GNSS+ Conference*, Institute of Navigation, 2019, pp. 1–1.

[57] ——, "Integrity for gps-lidar fusion utilizing a raim framework," in *Proceedings of ION GNSS+ Conference*, Institute of Navigation, 2018, pp. 3145–3155.

[58] S. Gupta and G. X. Gao, "Particle RAIM for integrity monitoring," in *Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019), Miami, Florida*, 2019, pp. 811–826.

[59] S. Bhamidipati and G. X. Gao, "Multiple GPS fault detection and isolation using a graph-SLAM framework," in *31st International Technical Meeting of the Satellite Division of the Institute of Navigation, ION GNSS+*, Institute of Navigation, 2018, pp. 2672–2681.

[60] ——, "Integrity Monitoring of Graph-SLAM Using GPS and Fish-eye Camera," in *Journal of the Institute of Navigation*, 2020.

[61] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, no. 2, pp. 1805–1824, Aug. 2000, ISSN: 1095-3787. [Online]. Available: `http://dx.doi.org/10.1103/PhysRevE.62.1805`.

[62] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[63] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *European Conf. on Computer Vision*, In Press, 2020.

[64] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[65] Z. Zhang, G. Ernst, S. Sedwards, P. Arcaini, and I. Hasuo, "Two-layered falsification of hybrid systems guided by Monte Carlo tree search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2894–2905, 2018.