Failure Analysis of Autonomous Systems with RL-Guided MCMC Sampling

Rory Lipkis
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA
rory.lipkis@nasa.gov

Adrian Agogino
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA
adrian.k.agogino@nasa.gov

ABSTRACT

Advanced autonomous systems are increasingly deployed for critical tasks but are rarely amenable to standard verification and validation techniques. Manually-refined Monte Carlo sampling is often the only recourse for the practical assessment of system behavior and the discovery of anomalies. However, this approach scales poorly when applied to systems with high-dimensional states, multiple agents, or long time horizons: rare failures cannot be effectively analyzed by direct sampling. We improve on previous work and demonstrate RL-MCMC, a Monte Carlo Markov chain approach to the efficient generation of rare system failures and the accurate estimation of failure mode log-likelihood. MCMC algorithms enable the sampling of arbitrary unnormalized distributions that lack an explicit sampling mechanism; however, they are highly sensitive to initialization and commonly suffer from convergence issues. We present a method to find ideal initializations for the MCMC sampling process with reinforcement learning, leveraging the power of modern neural network-based policy optimization to solve nontrivial and highly-constrained sequential tasks. By formulating a Markov decision process (MDP) to explicitly learn modal paths to failure, it is possible to bypass the unreliable convergence phase of the MCMC algorithm and immediately generate valid, in-distribution system failures. We assess the approach with two simple example problems and demonstrate the accuracy and stability of the likelihood estimation.

KEYWORDS

Reinforcement learning, Monte Carlo Markov chains, system validation, failure analysis

1 INTRODUCTION

Validation of a system under test (SUT) is the process of determining whether requirements specified in the design are met by the implementation. Formal methods (typically model checking and theorem proving, in their many flavors) are able to rigorously confirm or refute this correspondence when the system description is relatively small and its dynamics conform to the necessary idealizations [6, 7]. When these conditions cannot be met, statistical validation (falsification) is performed instead. This requires the tester to sample system inputs, perturbations, or transitions according to a predetermined model, categorize the observed failures, and estimate the overall reliability of the system.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2025), Avalos, Aydeniz, Müller, Mohammedalamen (eds.), May 19 – 20, 2025, Detroit, Michigan, USA, ala-workshop.github.io. 2025.

The Monte Carlo approach deteriorates when sampling occurs over higher dimensions or longer horizons. Many systems ultimately fail due to unanticipated correlations; since direct sampling explores regions in proportion to their likelihood of occurrence, the majority of computational effort is expended in the evaluation of near-nominal behavior. In addition, as the parameters of the perturbation model inform the exploration, small model errors have a vast impact on the sampling outcome. Because of these difficulties, it is common practice to manually bias the search toward suspected issues, leveraging pre-knowledge of the system to expedite failure discovery. This approach incurs the risk of missing failure modes unforeseen by the system developers and testers.

As an alternative, reinforcement learning-informed sampling (RLIS) is a two-stage framework that gathers and exploits information about the state space, allowing failure modes to be discovered with more independence and greater efficiency [10]. In the first stage, reinforcement learning is used to find a policy that encodes an approximation of the statistical modes of a failure likelihood function. This policy then forms the basis of a surrogate distribution, through which the true failure probability can be estimated by importance sampling. RLIS does not depend on any particular learning algorithm or policy representation, making it applicable to a wide variety of systems under test. It is also moderately robust under uncertainty and model error: since likelihood appears as part of an objective function, not as the property of a generative process, the learning stage is less sensitive to unmodeled effects.

However, importance sampling is notoriously sensitive; the variance of its output can be unacceptably high or even unbounded when the surrogate distribution differs significantly from the theoretical ideal [16]. Since RLIS relies on approximating the surrogate, it is susceptible to the same instability. Accurately quantifying the sampling error in practice is difficult due to the low magnitude of the typical failure likelihoods, which are estimated in absolute probability space, with $p \in [0, 1]$.

This paper vastly improves on RLIS by replacing importance sampling with Monte Carlo Markov chain (MCMC) methods. This family of algorithms enables an implicitly-specified distribution to be sampled without requiring the calculation of a normalizing constant (which is intractable in all but the simplest cases) or the derivation of an explicit generation mechanism. MCMC methods are highly sensitive to initialization, particularly when portions of the domain are infeasible, and commonly exhibit a prolonged phase of convergence during which the output is far from the mode of the distribution and is effectively invalid. As a result, they are not suited to the problem of failure *discovery*, which involves optimizing over a highly discontinuous objective.

However, we can use the RLIS failure policy to accelerate the MCMC algorithm, since it is explicitly formulated to learn the *modes* of the relevant distribution (the probability of system transitions conditioned on eventual failure). The learned policy is used as a bridge distribution to directly sample the theoretically ideal surrogate distribution using the method of *path sampling*. Notably, the failure probability estimate and its confidence bounds are formed directly in log-probability space, resulting in significantly lower variance and more reliable estimation. These developments vastly improve the accuracy and stability of the framework.

2 BACKGROUND

We consider a simulation consisting of a black box SUT and a stochastic environment, which interact over a fixed time horizon. At each point in time, the simulation is summarized by a state $s \in \mathcal{S}$; the internal state of the SUT does not need to be directly observable, but system failure must correspond to some subset of simulation states $\mathcal{F} \subset \mathcal{S}$. The environment consists of a set of external variables collected into the random variable $X_t \in \mathcal{X}$, where $X_t \sim p_t(x)$ and x may be multidimensional. In this paper, we consider only time-stationary distributions; the subscript t serves to differentiate step-wise variables from their multi-step counterparts, which are bolded for clarity. It is assumed that all randomness can be externalized and captured in the specification of the environment (i.e., the SUT is either deterministic or de-randomizable). It should also be possible to specify a rough distance-to-failure metric $d(s)$, a scalar function that is non-negative and attains a value of zero upon failure; this value serves to guide the learning and sampling processes. Note that the distance metric is specific to the observable simulation variables and does not provide insights into the behavior of the SUT.

As an example, we take as the SUT a generic aircraft collision avoidance module: a large software component containing complex interactions between program logic, models, and data, whose function is to run in real-time aboard an aircraft, monitor the surrounding aircraft, and occasionally provide the pilot with recommended maneuvers to lower the risk of collision. This sort of heterogeneous system presents a realistic, if daunting, verification challenge. The simulation might consist of various aircraft in an airspace, all running (and dutifully obeying) the collision avoidance module. The simulation state would contain the positions and velocities of the aircraft, while the environment would describe factors external to the SUT such as pilot controls (when not under recommendation), wind gusts, or sensor noise. A sensible distance metric would be the minimum pairwise distance between aircraft; failure occurs if $d = 0$. The metric provides a high-level signal, but the internal mechanisms of the SUT are unobservable and its paths to failure may be as complex and hybrid as the system itself.

For each episode of simulation, the state is initialized at some $s_0 \in \mathcal{S}$. Then, a sample $x_t \in \mathcal{X}$ is drawn from the environment and the simulation is advanced. The environment-system update step is repeated for a fixed number of time steps, or until failure. We refer to this as the *rollout* of the environment; a given sequence of environment samples $\boldsymbol{x} = [x_1, x_2, \ldots x_T] \in \mathcal{X}^T$ is a *trace.*

Since the failure region $\mathcal{F}$ may be arbitrary complex, it is defined implicitly by an indicator function $\mathbf{1}_{\mathcal{F}}(s) \in \{0, 1\}$. For a fixed initialization s_0, one can also consider the function $f(\boldsymbol{x})$, which indicates whether or not failure occurred at any point across the rollout of the trace $\boldsymbol{x}$.

3 SAMPLING FRAMEWORK

Let $\boldsymbol{X} = [X_1, X_2, \ldots X_T]$ be the random trace corresponding to a T-step rollout of the environment. Then, the result of $f(\boldsymbol{X})$, as previously defined, is a binary random variable indicating whether or not a failure occurred. The probability of failure is calculated as

$$\mu = P(f(\boldsymbol{X}) = 1) = \mathbf{E}[f(\boldsymbol{X})] = \int_{\mathcal{X}^T} f(\boldsymbol{x})p(\boldsymbol{x})\, d\boldsymbol{x}\,,$$

where $p(\boldsymbol{x}) = \prod_{t=1}^{T} p_t(x_t)$ is the joint probability distribution of the trace and $d\boldsymbol{x} = dx_1 \wedge \cdots \wedge dx_T$. Regular statistical validation is equivalent to Monte Carlo integration, in which

$$\mathbf{E}[f(\boldsymbol{X})] \approx \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}^{(i)})\,,$$

where sample traces $\boldsymbol{x}^{(i)}$ are drawn from $\boldsymbol{X}$. When failures are rare and difficult to elicit, the estimate is likely to be zero, which may not be a particularly useful result.

RLIS mitigates the elicitation problem by constructing a surrogate distribution $q(\boldsymbol{x})$ using reinforcement learning and re-weighting the samples to correct for their disproportionate contribution to the probability estimate, according to the identity

$$\mu = \int_{\mathcal{X}^T} f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})\, d\boldsymbol{x} = \mathbf{E}\left[f(\boldsymbol{X}^*)\frac{p(\boldsymbol{X}^*)}{q(\boldsymbol{X}^*)}\right]\,,$$

where $\boldsymbol{X}^* \sim q(\boldsymbol{x})$. The surrogate is formed by learning paths to failure in a setting where exploration is largely decoupled from the generative model $p(\boldsymbol{x})$, and then recentering the sampling model around these paths. A related approach is developed in [12], where Bayesian methods are used to refine a surrogate for importance sampling.

While this approach can be effective in low-dimensional settings, importance sampling has a severe limitation: the sample variance tends to increase dramatically as the surrogate distribution differs from the ideal. This variance can vastly overwhelm the estimate, in the worse case causing it to fall outside the valid interval $p \in [0, 1]$. In RLIS, the high dimensionality of the distribution (with scales with both the number of environment variables and the time horizon) tends to magnify the difference between the distributions.

The variance of an importance sampling scheme is given by

$$\begin{aligned}\sigma^2 &= \mathbf{E}\left[\left(f(\boldsymbol{X}^*)\frac{p(\boldsymbol{X}^*)}{q(\boldsymbol{X}^*)}\right)^2\right] - \mu^2 \\ &= \int_{\mathcal{X}^T} f(\boldsymbol{x})^2\frac{p(\boldsymbol{x})^2}{q(\boldsymbol{x})^2}q(\boldsymbol{x})\, d\boldsymbol{x} - \mu^2 \\ &= \int_{\mathcal{X}^T} \left(f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - \mu\right) f(\boldsymbol{x})p(\boldsymbol{x})\, d\boldsymbol{x}\,.\end{aligned}$$

The ideal variance-minimizing surrogate is thus

$$q^*(\boldsymbol{x}) = \frac{f(\boldsymbol{x})p(\boldsymbol{x})}{\mu}\,.$$

This distribution is not directly realizable, since it depends on μ, the variable we would like to estimate. However, it can be related to the fundamental problem of stress testing. Consider the conditional probability distribution function of a trace $\boldsymbol{x}$ *given* eventual failure. By Bayes' theorem, this can be written as

$$\begin{aligned} p(\boldsymbol{x} \mid f(\boldsymbol{x})) &= P(\boldsymbol{X} = \boldsymbol{x} \mid f(\boldsymbol{X}) = 1) \\ &= P(f(\boldsymbol{X}) = 1 \mid \boldsymbol{X} = \boldsymbol{x}) \frac{P(\boldsymbol{X} = \boldsymbol{x})}{P(f(\boldsymbol{X}) = 1)} \\ &= \frac{f(\boldsymbol{x})p(\boldsymbol{x})}{\mu} . \end{aligned}$$

From this perspective, it is clear that the conditional PDF is proportional to the quantity $q(\boldsymbol{x}) = f(\boldsymbol{x})p(\boldsymbol{x})$. If it were possible sample the normalized form of $q(\boldsymbol{x})$, we could perform importance sampling with an ideal surrogate and compute a perfect estimate of the failure likelihood. However, this would require first knowing the normalizing constant.

This motivates an alternative approach that abandons the mechanism of importance sampling. If $q(\boldsymbol{x})$ could instead be sampled directly in its unnormalized form, those samples would be identical in distribution to the conditional PDF. An adapted procedure could then be used to integrate over the distribution with its own samples. An elegant resolution to the problem is provided by Monte Carlo Markov chain (MCMC) methods.

4 MCMC SAMPLING

MCMC sampling involves constructing a Markov chain whose stationary distribution is proportional to a given function. Crucially, this allows an unnormalized PDF of arbitrary complexity to be sampled without ever calculating the normalization or deriving an explicit sampling mechanism. In many applications, such as Bayesian computation or the analysis of quantum systems, normalization factors are practically uncomputable.

In the symmetric random-walk Metropolis algorithm, a special case of the general MCMC formulation, the Markov chain is initialized with a value $\boldsymbol{x} \in \mathcal{X}^T$. At each iteration, a transition $\Delta\boldsymbol{x}$ is drawn from a symmetric transition distribution to form a candidate value $\boldsymbol{x}' = \boldsymbol{x} + \Delta\boldsymbol{x}$. The candidate is accepted with probability

$$P_{\text{acc}}(\boldsymbol{x} \to \boldsymbol{x}') = \min\left(1, \frac{q(\boldsymbol{x}')}{q(\boldsymbol{x})}\right) .$$

If accepted, $\boldsymbol{x}'$ becomes the next value in the chain; otherwise, $\boldsymbol{x}$ is reused. The choice of transition distribution can determine the efficacy of the process. The environment distribution $p(\boldsymbol{x})$, if appropriately symmetrized, provides a natural candidate.

Note that since $q(\boldsymbol{x})$ is only evaluated in ratio form, the constant of proportionality is irrelevant. It can be shown that under a wide range of conditions, this process eventually converges in distribution to a sampling of $q^*(\boldsymbol{x})$, the normalized distribution [14], which is here equal to the conditional distribution $p(\boldsymbol{x} \mid f(\boldsymbol{x}))$.

The period of convergence for a MCMC process is commonly referred to as *burn-in.* Since the initial value may be far from the bulk of the probability mass of $q^*(\boldsymbol{x})$, the process requires a high number of iterations before the chain begins to resemble IID samples. In the early stages of the algorithm, the chain is preoccupied with moving towards regions of higher probability. As a result, many common convergence metrics rely on autocorrelation heuristics, but the lack of a ground truth and the general difficulty of comparing high-dimensional distributions make this assessment difficult.

For the stress-testing problem, the distribution may be highly irregular, consisting of large discontinuities and plateaus. Because of the algorithm's acceptance criterion, the chain is effectively performing a noisy hill-climbing procedure. As such, it must be initialized with a valid failure trace, and it may fail to converge if the trace is too far from the mode of the distribution.

5 ADAPTIVE STRESS TESTING

An optimal MCMC initialization can be found with reinforcement learning. The adaptive stress testing (AST) framework formulates stress testing as an MDP whose solutions represent the highest likelihood failure traces [9]. This approach has proven useful in a variety of applications, including the analysis of aircraft collision avoidance software [9, 11], pedestrian avoidance procedures for autonomous vehicles [1, 5], trajectory planners for small unmanned aircraft [8], aircraft taxiing algorithms [4], and flight management systems [13].

Rather than sampling from the environment, AST explicitly optimizes over environment values to find the most likely failure events in a near black box SUT. For a T-step sequence of simulation states, the joint likelihood is given by

$$p(s_0, \ldots, s_T) = p(s_0) \prod_{t=0}^{T-1} p(x_t)$$

due to the Markov property and assumption that randomness has been externalized to the environment. For a given initialization s_0, the high-level goal of AST is then to solve the optimization problem

$$\begin{aligned} \max_{x_0,\ldots,x_{T-1}} \quad & \prod_{t=0}^{T-1} p(x_t) \\ \text{subject to} \quad & s_T \in \mathcal{F} . \end{aligned}$$

This is precisely equivalent to maximizing the conditional PDF, since within the feasible region $\mathcal{F}$,

$$q^*(\boldsymbol{x}) = \frac{f(\boldsymbol{x})p(\boldsymbol{x})}{\mu} \propto p(\boldsymbol{x}) .$$

Because the constraint may be arbitrarily complex (recall that $\mathcal{F}$ is defined implicitly in $\boldsymbol{x}$), the problem benefits from a non-classical optimization technique. In particular, the MDP formulation enables a reinforcement learning approach. Figure 1 illustrates the high-level AST architecture. At each time step, the reinforcement learner observes the state s, selects an environment instance x_t, advances the simulation to state s', and receives a reward

$$r(s, x_t, s') = \log p(x_t) + \Delta(s, s') + r_f \cdot \mathbf{1}_{\mathcal{F}}(s) ,$$

where r_f is a bonus for reaching a failure state and

$$\Delta(s, s') \propto d(s) - d(s')$$

is a potential-based reward-shaping term designed to guide the learning agent. This amounts to a softened version of the original optimization, as the constraint is replaced by a penalty; the relaxation does not change the theoretical optimum.

The scope can be expanded to learn a failure policy π, which maps simulation states to environment values that induce the likeliest path to failure. This is a standard reinforcement learning policy with actions corresponding to (adversarial) instances of the environment. It amounts to a latent representation of multiple independent failure modes, reachable through successive applications of the policy over the time horizon.

A wide variety of algorithms can be used to solve the AST MDP. Deep reinforcement learning offers an natural solution when state and environment spaces are high-dimensional. Due to the ability of neural networks to interpolate and generalize, this approach allows failure paths to be approximated between samples.

The output of AST is a trace $\boldsymbol{x}^*$ or trace-generating policy π^* that corresponds to the mode of $q^*(\boldsymbol{x})$. Because of this property, it can directly seed the MCMC process in the region of highest probability mass, mitigating the burn-in phase. One is then able to immediately generate a stream of failure traces, which can be treated as samples from the conditional distribution $p(\boldsymbol{x} \mid f(\boldsymbol{x}))$. These samples can be used to characterize the failure modes in both quantitative and qualitative analyses. The basic process is described in Algorithm 1.

Algorithm 1 RL-guided MCMC sampling

```
Input: $\boldsymbol{x}^*$, $N$                              ▷ Initialization from AST
Output: samples
 1: samples ← {}                                        ▷ Collection of samples
 2: $\boldsymbol{x} \leftarrow \boldsymbol{x}^*$
 3: $q \leftarrow p(\boldsymbol{x}^*)$
 4: for $i = 1$ to $N$ do
 5:     $\Delta\boldsymbol{x} \leftarrow$ Sample[$p(\boldsymbol{x})$]             ▷ Candidate transition
 6:     $\boldsymbol{x}' \leftarrow \boldsymbol{x} + \Delta\boldsymbol{x}$
 7:     $q' \leftarrow p(\boldsymbol{x}')$
 8:     if IsFailure($\boldsymbol{x}'$) then
 9:         $\alpha \leftarrow$ Sample[Unif(0, 1)]
10:         if $\alpha < q'/q$ then                      ▷ Acceptance criterion
11:             $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
12:             $q \leftarrow q'$
13:         end if
14:     end if
15:     append $\boldsymbol{x}$ to samples
16: end for
17: return samples
```

6 PROBABILITY ESTIMATION

The remaining task is to estimate the failure probability μ using MCMC samples. This presents a subtle challenge: since the samples are realizations of the random variable corresponding to the conditional distribution of interest, they cannot be directly used to calculate the normalizing constant of that same distribution[1]. In other words, MCMC methods allow us to compute any expectation with respect to the unnormalized distribution $q(\boldsymbol{x})$; the problem is determining what quantity yields in expectation the normalizing constant.

To compute μ, we make use of a powerful application of MCMC known as path sampling, which derives from the thermodynamic integration method of computational physics [2]. This approach requires us to specify a continuous transformation from a distribution with a known normalization to the target distribution. The desired solution is then obtained via a path integral through parameter space.

Consider the unnormalized probability distribution function

$$q_\theta(\boldsymbol{x}) = \exp\left(\frac{\beta d(\boldsymbol{x})}{\ln\theta}\right) p(\boldsymbol{x}),$$

where $d(\boldsymbol{x})$ represents the lowest value of the distance-to-failure metric d achieved across the trace $\boldsymbol{x}$. The hyperparameter $\beta > 0$ is chosen such that its product with $d(\boldsymbol{x})$ is roughly $O(1)$ across the domain. The exponential term represents an analytic continuation of the indicator function $f(\boldsymbol{x})$ such that

$$\lim_{\theta\to 0} q_\theta(\boldsymbol{x}) = p(\boldsymbol{x}) \quad \text{and} \quad \lim_{\theta\to 1} q_\theta(\boldsymbol{x}) = f(\boldsymbol{x})p(\boldsymbol{x}).$$

The presence of $d(\boldsymbol{x})$ in the expression allows the transition between the two distributions to be informed by the shape of the failure mode; it softens the discontinuities inherent to the original conditional distribution. The normalized probability distribution function is

$$q^*_\theta(\boldsymbol{x}) = \frac{q_\theta(\boldsymbol{x})}{z_\theta},$$

where $z_\theta = \int q_\theta(\boldsymbol{x})\,d\boldsymbol{x}$. By construction, $z_0 = 1$ and $z_1 = \mu$. From the definition of z_θ, it follows that

$$\frac{d}{d\theta}\ln z_\theta = \mathbf{E}_X\left[\frac{d}{d\theta}\ln q_\theta(X)\right] = -\mathbf{E}_X\left[\frac{\beta\, d(X)}{\theta\ln(\theta)^2}\right],$$

where $X \sim q^*_\theta(\boldsymbol{x})$ via MCMC sampling. This identity is integrated to yield

$$\begin{aligned}\ln\mu = \ln\left(\frac{z_1}{z_0}\right) &= \int_0^1 \frac{d}{d\theta}\ln z_\theta\, d\theta \\ &= -\mathbf{E}_{X,\Theta}\left[\frac{\beta\, d(X)}{\Theta\ln(\Theta)^2}\right],\end{aligned}$$

where the expectation is performed jointly over variables X and $\Theta \sim \mathcal{U}(0,1)$. This value is realized with the double-loop estimator

$$\widehat{\ln\mu} = \frac{1}{m}\sum_{i=1}^m \hat{u}\left(\theta^{(i)}\right),$$

$$\hat{u}(\theta) = -\frac{\beta}{\theta\ln(\theta)^2}\cdot\frac{1}{n}\sum_{j=1}^n d\left(\boldsymbol{x}_\theta^{(j)}\right),$$

where $\theta^{(i)}$ are drawn randomly from the range $(0,1)$ and $\boldsymbol{x}_\theta^{(j)}$ are the output of a MCMC sampling of $q^*_\theta(\boldsymbol{x})$. Since the estimate is formed directly as a logarithm, it is well suited to the computation of small probabilities.

Along with the estimate of the failure log-probability, the sample variance is calculable as

$$\widehat{\varsigma^2} = \frac{1}{m(m-1)}\sum_{i=1}^m\left[\hat{u}(\theta^{(i)}) - \widehat{\ln\mu}\right]^2,$$

where the $m-1$ term is the standard bias correction. The full procedure is described in Algorithm 2. The estimator is theoretically unbiased; as the number of samples increases, the distribution of the estimator should approach a normal distribution centered on

[1]In the univariate setting, the normalization could be roughly estimated via the empirical CDF, but this approach does not generalize to higher dimensions.

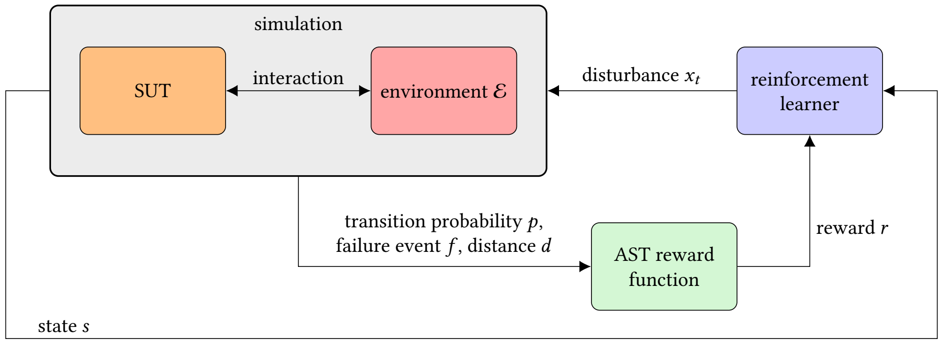

Figure 1: Adaptive stress testing (AST) architecture. An adversarial reinforcement learning agent chooses realizations of a stochastic environment to elicit the likeliest possible failure in the system under test.

Algorithm 2 Path sampling estimation

```
Input: $x^*, \beta, M, N$                              ▷ Initialization from AST
Output: $\widehat{\ln\mu}, \widehat{\varsigma^2}$
 1: $u \leftarrow \{\}$
 2: for $i = 1$ to $M$ do
 3:     $\theta \leftarrow$ Sample[Unif(0, 1)]
 4:     $d \leftarrow \{\}$
 5:     $x \leftarrow x^*$
 6:     $q \leftarrow p(x^*)$
 7:     for $j = 1$ to $N$ do
 8:         $\Delta x \leftarrow$ Sample[$p(x)$]                ▷ Candidate transition
 9:         $x' \leftarrow x + \Delta x$
10:         $q' \leftarrow \exp(\beta d(x')/\ln\theta)p(x')$      ▷ Bridge surrogate
11:         $\alpha \leftarrow$ Sample[Unif(0, 1)]
12:         if $\alpha < q'/q$ then                    ▷ Acceptance criterion
13:             $x \leftarrow x'$
14:             $q \leftarrow q'$
15:         end if
16:         append $d(x)$ to $d$
17:     end for
18:     $u \leftarrow -\beta/\big(\theta\ln(\theta)^2\big)\cdot \mathrm{mean}(d)$
19:     append $u$ to $u$
20: end for
21: return mean($u$), var($u$)/$M$
```

the true failure log-probability with variance given by the above expression [14]. This yields a concentration bound

$$P\left(\left|\ln\mu - \widehat{\ln\mu}\right| \geq \delta\right) \approx 2\left(1 - \Phi\left(\frac{\delta}{\sqrt{\widehat{\varsigma^2}}}\right)\right),$$

equivalently expressed as the confidence interval

$$P\left(\left|\ln\mu - \widehat{\ln\mu}\right| \geq \sqrt{\widehat{\varsigma^2}}\;\Phi^{-1}\Big(1-\frac{\epsilon}{2}\Big)\right) \approx \epsilon\,.$$

In experiments, we find that the sample variance decreases consistently with number of samples, while the estimator exhibits a small positive bias that appears to be proportional to the square root of the sample variance; this is consistent with the definition of an unbiased estimator, since the bias still converges to zero in the high-sample limit. More precisely, we note that the z-score formed by the estimator and its sample variance appears to converge in distribution to

$$\hat{z} = \frac{\widehat{\ln\mu} - \ln\mu}{\sqrt{\widehat{\varsigma^2}}} \sim \mathcal{N}(\ln 2, 1)\,,$$

although this effect has only be measured empirically and warrants further study. If confirmed analytically, it could be used to boost the accuracy of the confidence interval.

Because the sample variance is itself a quantity over log-probability space, it is much more accurate than in the linear setting of Monte Carlo estimation, where $\hat{\mu} \pm \hat{\sigma}$ represents an uneven and often meaningless spread over probabilities that spans multiple orders of magnitude and can easily lie outside of the interval $[0, 1]$. With the RL-MCMC method, the error bars effectively scale down for ultra-low probabilities, making them useful for rare event analysis.

7 EXPERIMENTAL RESULTS AND DISCUSSION

7.1 One-dimensional example (estimation)

We first demonstrate RL-MCMC with a very simple toy problem that admits an analytical solution. This allows us to evaluate the accuracy, efficiency, and consistency of the joint learning-estimation scheme.

In this problem, we imagine a UAV attempting to fly with constant velocity v, remaining above a given altitude until it reaches a goal located at a horizontal distance d away. The environment produces a stochastic change in altitude at each time step, according to $X_t \sim \mathcal{N}(0, \sigma^2)$. Failure occurs if the altitude drops further than a certain value h. The control system is assumed to be in a failure state and is unable to effectively stabilize the altitude. The probability of system failure is thus simply the probability that an unimpeded random walk exceeds a certain threshold, which can be calculated directly as

$$p_{\text{fail}} = 1 - \Phi\left(\frac{h}{\sigma\sqrt{\lceil d/v\rceil}}\right),$$

where Φ is the cumulative distribution function of the normal distribution. For the chosen parameters (listed in the caption of Figure 2), the failure probability is approximately 9.852×10^{-12}. Although the general path to failure could not be more obvious, the Monte Carlo approach is hopeless. On average, it would require over 10^{11} trials to reveal the existence of the failure mode, and orders of magnitude more in order to begin characterizing it; the effect is seen in Figure 2.

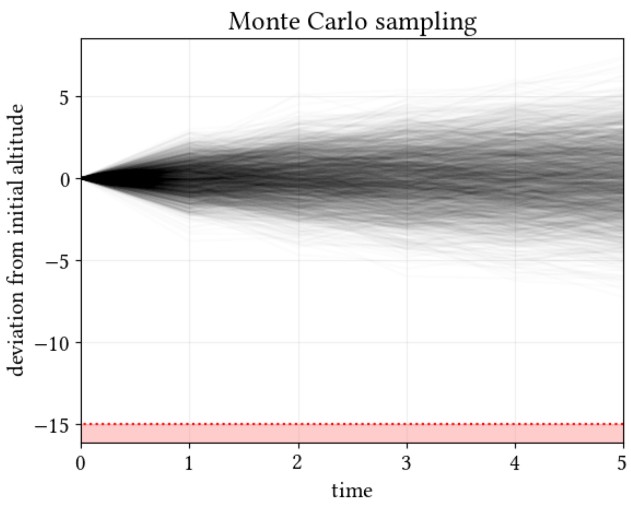

Figure 2: Random trajectories sampled from the system. Parameters are $\Delta t = 1$ second, $h = 15$ meter, $\sigma = 1$ meter, $v = 10$ meters per second, and $d = 50$ meters. While the accumulation of perturbations is evident, the Monte Carlo approach is not sufficient to reach failure; a high degree of time correlation is required.

The RL-MCMC approach looks very different. We first learn the mode of the failure distribution, here choosing the seed-action variant of AST as described in [9]. This variant treats both the system and environment as near black boxes and requires Monte Carlo tree search (MCTS) to find the likeliest paths to failure without access to direct state or action information. Instead, the action space is merely the space of random seeds used to set the system random number generator at each time step. Note that the vast majority of RL (and classical control) algorithms make active use of state information, and would be able to solve this problem trivially due to the direct relationship between actions and states. However, by using this more generic framework, we hope to make the example somewhat less trivial.

The reinforcement learning phase was run for 10^5 episodes. The 10 highest scoring paths through the tree are shown in Figure 3. By construction, these trajectories are representative of the mode of the conditional failure distribution but cannot be directly used to estimate probability of failure, since they are detached from their original probabilistic context.

Finally, the MCMC-based estimation is performed (Figure 4), following Algorithm 2, with hyperparameters set to $\beta = 1/h$, $n = 10^3$, and $m = 10^4$; this amounts to $m \cdot n = 10^7$ total trials. We obtain simultaneous estimates of

$$\widehat{\ln \mu} \approx -25.480$$

$$\sqrt{\widehat{\varsigma^2}} \approx 0.575\,.$$

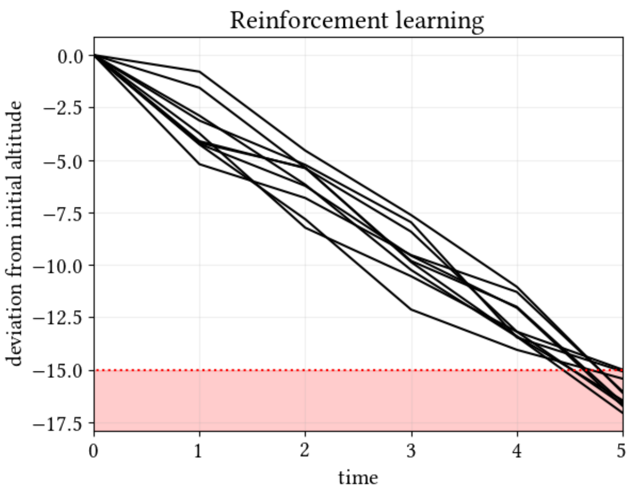

Figure 3: Highest scoring paths from the MCTS algorithm. In reinforcement learning, the exploration policy is detached from the generative process of the perturbation model and can easily find the correlated perturbations that lead to the region of failure. Since the AST framework optimizes for likelihood under the constraint of eventual failure, these trajectories represent the approximate mode of the desired conditional distribution.

Given that the true value is $\log p_{\text{fail}} \approx -25.343$, we can conclude that the RL-MCMC estimate is very close to the true value, and is capable of accurately estimating its own variance. In contrast to the Monte Carlo approach, our approach requires only $1.01 \cdot 10^7$ samples to produce a highly accurate estimate of the failure probability, representing an improvement in efficiency by several orders of magnitude.

We also characterize this estimation scheme over multiple runs. Each execution produces an estimate of the log-probability and the corresponding sample variance. As the number of samples increases, the z-score should be distributed according to a unit normal distribution. As described earlier, we observe a bias that is proportional to the square root of the sample variance, which decreases as the number of samples grows. Figure 5 shows a histogram of the results over 1000 random seeds. The mean of the z-scores is 0.695, close to the posited value of $\ln 2$, while the standard deviation is 1.023, close to the ideal value of 1.

7.2 Two-dimensional example (sampling)

To highlight the ability of RL-MCMC to learn and sample complex conditional distributions, we consider a system with an active SUT, less trivial dynamics, and a more tightly constrained route to failure. In this problem, we imagine a multirotor that must hover at the center of a horizontal square $[-a, a] \times [-a, a]$ while pointing in a fixed angle of $0°$ in the x-y plane; the vertical dimension is irrelevant. Two-dimensional Gaussian noise perturbs the position at each time step, and failure occurs if the vehicle exceeds its horizontal bounds.

Because of the fixed-angle requirement, the vehicle cannot be rotated in place to move in an arbitrary direction. An engineer

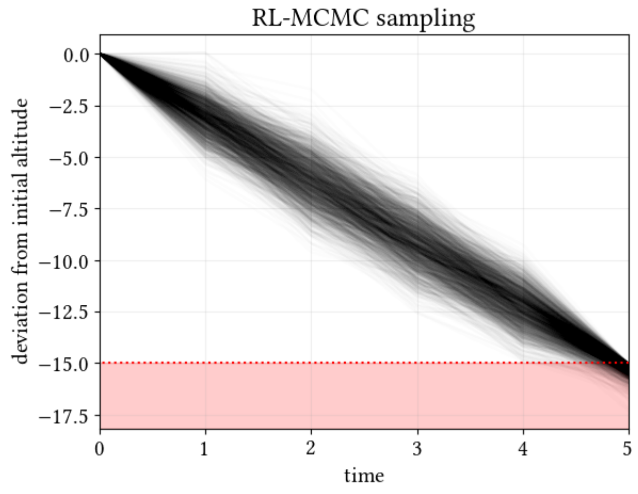

Figure 4: Random trajectories sampled directly from the conditional distribution. Since the MCMC process has been seeded at the mode of the distribution by the reinforcement learning policy, the convergence phase is eliminated. These samples can be used to characterize the distribution in greater depth.

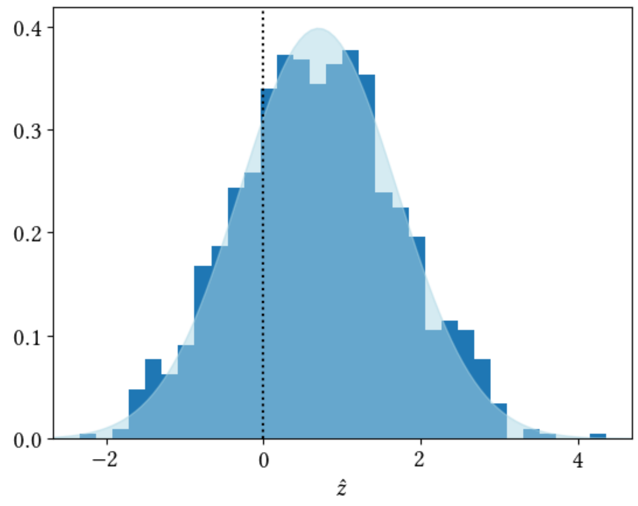

Figure 5: Distribution of z-scores for the estimated mean and sample variance. The shaded curve represents the $\mathcal{N}(\ln 2, 1)$ distribution, which we posit to be the limiting case of the statistic.

designs an ad-hoc controller to maintain the vehicle at the origin, relying on the fact that the multirotor can be rolled along either of the two non-vertical axes to produce horizontal movement. To avoid a net change in pointing angle, only one axis can be rolled at a time, and therefore these corrective positional maneuvers can only occur along the unit vectors $\hat{x}$ and $\hat{y}$, although the sign and magnitude of the impulse can vary.

The controller is implemented as follows: at each time step, if the multirotor displacement (x, y) is sufficiently off-nominal, proportional control with a gain k is applied to the higher of the two coordinates[2]. In this way, the position is continually regulated back toward the origin. To avoid jitter around the origin, the engineer specifies the nominal region as

$$||x| - |y|| < \epsilon$$

for a small value of ϵ. This is an implementation *bug*: instead of defining a simple L_1 ball around the origin, the engineer's sign error has caused the nominal region to extend along $[\pm 1, \pm 1]$ in a $O(\epsilon)$ width strip. If the position falls in this region, no correction will be applied, although the effect is subtle enough under the noise disturbance that failures remain rare (Figure 6). Note that manual coordinate-wise bounds testing would not necessarily reveal viable paths to failure.

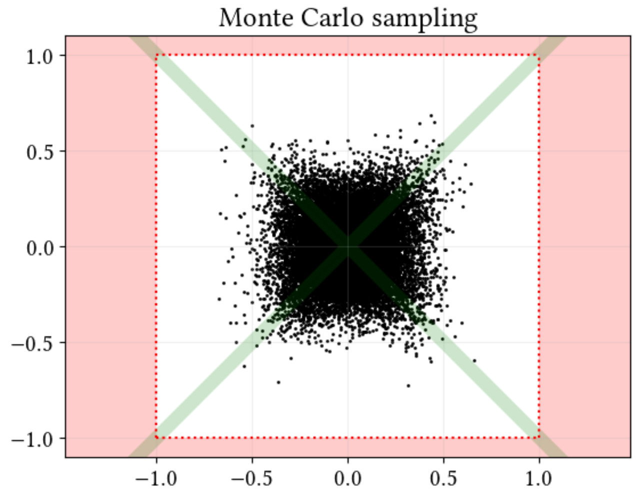

Figure 6: Random trajectories sampled from the system. Parameters are $a = 1, \sigma = 0.1, \epsilon = 0.05$, and $k = 0.25$. In most of the domain, the SUT applies restricted proportional control to quickly regulate the multirotor back to the origin. An implementation error creates corridors of the state space (shown in green) where the controller does not function correctly. Due to the stochasticity of the environment and the near-correctness of the SUT, this latent failure mode is obscured in naïve Monte Carlo testing.

The results are similar to the first example. Here, the failure model is learned with the PPO algorithm [15]; unlike MCTS, PPO makes direct use of simulation state information to learn a failure policy, which associates positions with optimal disturbances along the modal path to failure (the SUT itself remains a black box). PPO was chosen because of its common use as an RL baseline; other standard baselines, such as soft actor-critic [3] and tabular Q-learning [17] have been successfully used with the adaptive stress testing framework [11]. The learning is performed for approximately 10^4 trials (10^5 steps), and the subsequent MCMC procedure

[2]Damping from aerodynamic drag obviates the need for a full PID controller.

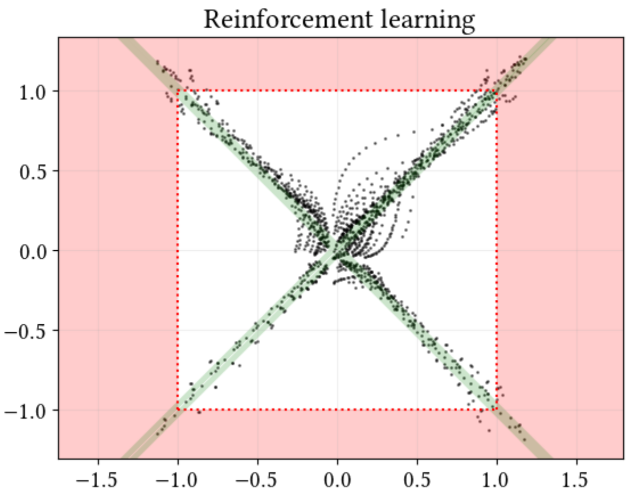

Figure 7: Samples from the PPO failure policy, with random initializations close to the origin. Deep reinforcement learning is able to quickly identify and pursue the likeliest paths to failure, although the algorithm has no knowledge of the underlying SUT implementation. As before, these trajectories are approximately modal; further training would improve the policy but the slight inexactness is not major problem for the subsequent MCMC phase.

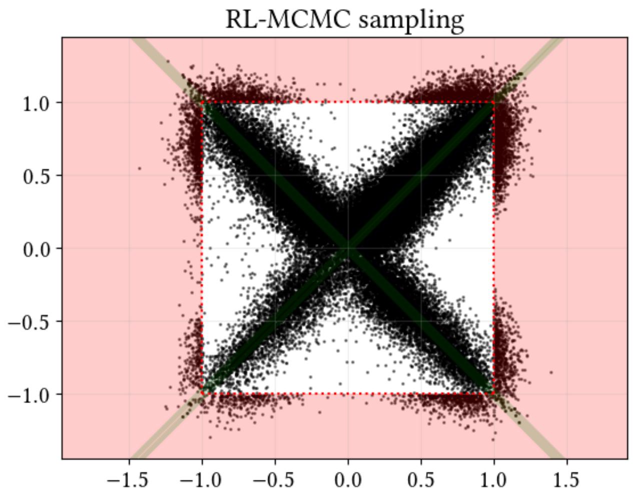

Figure 8: Random trajectories sampled directly from the conditional distribution. Some imperfections and asymmetries from the PPO policy are still visible at this stage in the sampling but are gradually smoothed out as the MCMC process continuously optimizes its distribution.

is run for 10^7 total trials, yielding a large set of failure trajectories sampled directly from the conditional distribution. These phases are shown in Figures 7 and 8, respectively. As in the first example, an equivalently-sized Monte Carlo experiment yields zero failures.

Figure 7 demonstrates why this approach is fairly tolerant of model error. If the statistical parameters of environment are altered (for instance, the perturbation magnitude), a prior Monte Carlo analysis is invalidated and subsequent results may be dramatically different. The reinforcement learning phase is somewhat invariant to this effect; once a model has been learned, the MCMC sampling can be rerun with slightly different statistical parameters. Even if the RL policy no longer represents the exact mode of the conditional distribution, the MCMC process can compensate through burn-in, optimizing its chain of samples toward the correct mode as the sampling progresses.

8 FUTURE WORK

There are several aspects of this work that are ongoing. We would like to explore the impact on the algorithm of critical hyperparameters such as β. Additionally, it will be important to better characterize the convergence properties of the path sampling method, which includes a formal analysis of the estimator bias and the development of a second-order correction factor, if possible. Finally, recent years have seen an explosion in the success of diffusion- and transformer-based policies for difficult sequential decision-making tasks. Applying these methods to the adaptive stress testing problem could further improve the ability of this method to discover and analyze errors in real-world systems. Diffusion methods may also provide a complementary approach to the statistical estimation phase, since they rely on constructing a series of gradual transformations to bridge highly dissimilar distributions.

9 CONCLUSION

We have addressed an important limitation in current and state-of-the-art approaches to statistical validation of autonomous systems. Subtle implementation errors, software bugs, and machine learning model deficiencies tend to exhibit low-incidence, high-correlated failures that are not caught by Monte Carlo testing. As a result, the development and V&V process of autonomous systems can be notoriously long-tailed. We have presented an approach that combines reinforcement learning and MCMC sampling to efficiently discover and analyze these sorts of failures. The MCMC method allows rapid generation of failures provided a valid initialization within the failure mode, while a modified path sampling procedure uses those samples to accurately estimate the associated probability, along with a valid confidence interval. We show that reinforcement learning provides the key to enable this approach at scale by finding the optimal initialization through non-statistical methods; it benefits from the ability of modern ML to solve nontrivial and high-dimensional problems. Critically, the estimates are formed directly in logarithmic space, which is well suited to representing probability, and we demonstrate that the statistics are reasonably well-behaved over irregular underlying distributions. This development improves the efficacy of automated stress testing and is a step toward generation of artifacts for system certification.

ACKNOWLEDGMENTS

This work is supported by the Systems-Wide Safety (SWS) Project under the NASA Aeronautics Research Mission Directorate (ARMD) Airspace Operations and Safety Program (AOSP).

REFERENCES

[1] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. 2019. Adaptive Stress Testing with Reward Augmentation for Autonomous Vehicle Validation. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. https://doi.org/10.1109/ITSC.2019.8917242

[2] Andrew Gelman and Xiao-Li Meng. 1998. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical science* (1998), 163–185.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[4] Kyle D. Julian, Ritchie Lee, and Mykel J. Kochenderfer. 2020. Validation of Image-Based Neural Network Controllers Through Adaptive Stress Testing. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*.

[5] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J. Kochenderfer. 2018. Adaptive Stress Testing for Autonomous Vehicles. In *IEEE Intelligent Vehicles Symposium (IV)*. IEEE.

[6] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11) (LNCS, Vol. 6806)*, G. Gopalakrishnan and S. Qadeer (Eds.). Springer, 585–591.

[7] Leslie Lamport. 2003. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.

[8] Ritchie Lee, Ole J. Mengshoel, Adrian K. Agogino, Dimitra Giannakopoulou, and Mykel J. Kochenderfer. 2019. Adaptive Stress Testing of Trajectory Planning Systems. In *AIAA SciTech, Intelligent Systems Conference (IS)*. AIAA.

[9] Ritchie Lee, Ole J. Mengshoel, Anshu Saksena, Ryan Gardner, Daniel Genin, Joshua Silbermann, Michael Owen, and Mykel J. Kochenderfer. 2020. Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning. *Journal of Artificial Intelligence Research* 69 (2020), 1165–1201.

[10] Rory Lipkis and Adrian Agogino. 2023. Discovery and Analysis of Rare High-Impact Failure Modes Using Adversarial RL-Informed Sampling. In *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 123–140.

[11] Rory Lipkis, Ritchie Lee, Joshua Silbermann, and Tyler Young. 2022. Adaptive Stress Testing of Collision Avoidance Systems for Small UASs with Deep Reinforcement Learning. In *AIAA SciTech 2022 Forum*. 1854.

[12] Robert J Moss, Mykel J Kochenderfer, Maxime Gariel, and Arthur Dubois. 2023. Bayesian Safety Validation for Black-Box Systems. In *AIAA Aviation 2023 Forum*. 3596.

[13] Robert J. Moss, Ritchie Lee, and Mykel J. Kochenderfer. 2020. Adaptive Stress Testing of Trajectory Predictions in Flight Management Systems. In *IEEE/AIAA Digital Avionics Systems Conference (DASC)*. AIAA/IEEE.

[14] Art B. Owen. 2013. *Monte Carlo Theory, Methods, and Examples*. Preprint.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv:1707.06347* (2017).

[16] Larry Wasserman. 2013. *All of Statistics: A Concise Course in Statistical Inference*. Springer.

[17] Christopher J. C. H. Watkins and Peter Dayan. 1992. Technical Note: Q-Learning. *Machine Learning* 8 (1992), 279–292.