Rigorous Agent Evaluation: An Adversarial Approach to Uncover Catastrophic Failures

Jonathan Uesato*, **Ananya Kumar***†, **Csaba Szepesvari***, **Tom Erez, Avraham Ruderman**
Keith Anderson, Krishmamurthy (Dj) Dvijotham, Nicolas Heess, Pushmeet Kohli
DeepMind, London, UK
{juesato, ananyak, szepi, pushmeet}@google.com

Abstract

This paper addresses the problem of evaluating learning systems in safety critical domains such as autonomous driving, where failures can have catastrophic consequences. We focus on two problems: searching for scenarios when learned agents fail and assessing their probability of failure. The standard method for agent evaluation in reinforcement learning, Vanilla Monte Carlo, can severely underestimate agent failure probabilities, leading to the deployment of unsafe agents. In our experiments, we observe this even after allocating equal compute to training and evaluation. To address this shortcoming, we draw upon the rare event probability estimation literature and propose an *adversarial evaluation* approach. Our approach focuses evaluation on difficult scenarios that are selected adversarially. The key challenge is in identifying these adversarial situations – since failures are rare there is little signal to drive optimization. To solve this we propose a *continuation approach* that learns failure modes in related but less robust agents. We demonstrate the efficacy of adversarial evaluation on two standard RL domains (humanoid control and simulated driving). Experimental results show that our methods can find catastrophic failures and estimate failures rates of agents multiple orders of magnitude faster (hours instead of days) than standard evaluation schemes.

1 Introduction

How can we ensure machine learning systems do not make catastrophic mistakes? While machine learning systems have shown impressive results across a variety of domains, they may also fail badly on particular inputs, often in unexpected ways (Szegedy et al., 2013). As we start deploying these systems, it is important that we can reliably evaluate the risk of failure. This is particularly important for safety critical domains like autonomous driving where the negative consequences of a single mistake can overwhelm the positive benefits accrued during typical operation of the system.

Limitations of random testing. In typical RL formulations, the rewards and penalties are tightly bounded. For example an agent might get a reward of +1 for good behavior, and -1 for bad behavior. In such cases, evaluating an agent is easy – we run the agent for a small number of episodes and output the empirical mean reward. However, in safety critical environments, the penalty for failing can be much higher than the reward for success. For concreteness, consider a self-driving car company that decides the cost of an accident outweighs the benefit of 100 million miles of safe driving. A car with failure probability greater than 1e-8 has negative expected return – it would be better to not deploy such a car. However, to achieve reasonable confidence that the car crashes with probability below $1e$–8, we need to test-drive the car for at least $1e8$ miles, which may be prohibitively expensive.

*Equal Contribution
†Now at Stanford, ananya@cs.stanford.edu

Preprint. Work in progress.

Our Contributions. To overcome the above-mentioned problem, we develop an *adversarial evaluation* approach. The overarching idea is to screen out situations that are unlikely to be problematic, and focus evaluation on difficult situations. The challenge arises in identifying these situations – since failures are rare, there is little signal to drive optimization. To address this problem, we introduce a *continuation approach* to learning an *adversarial value function* (AVF), which estimates the probability the agent fails given some initial conditions. The idea is to leverage data from less robust agents, which fail more frequently, to provide a stronger learning signal. In our implementation, this also allows the algorithm to reuse data gathered for training the agent, saving time during evaluation.

We look at two settings where the AVF can be used. In the simplest setting, *failure search*, the problem is to efficiently find initial conditions that cause failures. This could be used to identify and debug potentially unsafe policies. Additionally, efficient adversaries can be used for adversarial training, by folding the states causing failures back into the training algorithm (Madry et al., 2017). The second setting, *risk estimation*, is the problem of efficiently estimating the failure probability of an agent, which also has a simple application to efficiently selecting the most reliable agent from a finite set (section 4.3). Empirically, we demonstrate large improvements in efficiency through adversarial testing on two domains, simulated driving and humanoid locomotion.

We hope our work emphasizes the importance of moving beyond the traditional RL setting where rewards and penalties are tightly bounded. If we want to deploy RL systems in the wild we need to pay careful attention to catastrophic failures, which, as our experiments show, can reveal flaws in agents we might otherwise think are safe. Additionally, we hope to motivate simulated reinforcement learning environments as a valuable testbed for researchers interested in adversarial examples. In traditional adversarial examples, moving beyond norm ball specification has been difficult, because human evaluation may be necessary to determine ground truth labels (see Brown et al. (2018); Song et al. (2018)). RL could be a valuable testbed, since ground truth is provided by the simulator. Unlike previous work on adversarial examples in RL (Huang et al., 2017; Lin et al., 2017), our method also works when adversaries generate inputs within the distribution on which the agent is trained.

2 Problem Formulation

Recall that we are interested in reliability assessment of a trained agent. We assume that the reliability assessment procedure can run an episode with the trained agent given some initial condition $x \in \mathcal{X}$, the outcome of which is a random failure indicator $C = c(x, Z)$ where $Z \sim P_Z$ for some probability distribution P_Z over some set $\mathcal{Z}$ and where $c : \mathcal{X} \times \mathcal{Z} \to \{0, 1\}$. C is binary and $C = 1$ indicates a "catastrophic failure". Z collects all sources of randomness due to the agent and the environment, but is neither observed, nor controllable by the evaluator. We are interested in the agent's performance on the environment distribution over initial conditions $X \sim P_X$, which we assume can be sampled quickly. Evaluating c requires performing an environment rollout or running the agent in the real-world so we assume that evaluating c on the pair (x, Z) is costly. We focus on two related problems:

Failure Search: The objective of *failure search* is to find a catastrophic failure of the agent as quickly as possible. Algorithms that search for failures, which we will call *adversaries*, can specify initial conditions x, observe the outcome C of running the agent on x and return as soon as $C = 1$.

Risk Estimation: The *failure probability* of the trained agent is $p = \mathbb{E}[c(X, Z)]$, where $X \sim P_X$ and $Z \sim P_Z$ independently. We say an estimate $\hat{P}$ is a ρ-approximation of p if it belongs to the interval $[p/\rho, p\rho]$. The objective is to produce a ρ-approximation as quickly as possible.

3 Approach

Our proposed solutions use historical data to estimate the *adversarial value function* (AVF) $f_* : \mathcal{X} \to [0, 1]$ that returns the probability of failure given an initial condition.

$$f_*(x) = \mathbb{P}(c(x, Z) = 1), \qquad x \in \mathcal{X}\,,$$

where $Z \sim P_Z$. We learn f, an approximation to f^*. f will be chosen so that the cost of evaluating f is negligible compared to running an episode. The high level idea is to use f to save on the cost of experimentation. We describe how we use f, and then discuss our approach to learning f.

Failure Search: If we had access to $f = f^*$, we would simply optimize over f. To compensate for the mismatch between f and f^* we use a simple procedure to choose a diverse set of samples. We

Domain	AVF Cost	VMC Cost	PR Cost	Acceleration Factor
Driving	3/5/11	200/1000/2700	NA	65/198/250
Humanoid	19/33/56	60K/110K/180K	9K/10K/220K	2100/3100/3800

Table 1: **Failure search.** We compare the number of episodes each adversary took to find an adversarial problem instance. Each column reports the min, median, and max over evaluations of 5 separate agents. In the median case, the AVF adversary finds adversarial inputs with 198x fewer episodes than random testing on Driving and 3100x fewer on Humanoid (Acceleration Factor).

sample n initial conditions from P_X, pick the initial condition x from this set where f is the largest, and run an experiment from x. We repeat this process with new sampled initial conditions until we find a catastrophic failure. The pseudocode is included in Appendix A.

Risk Estimation: We use importance sampling (e.g. Bucklew 2004) with a learned proposal distribution. Intuitively, we focus our evaluation efforts on more difficult scenarios – if initial condition x is more likely to lead to failure we pick x with higher probability. Since we do not have access to the density $p_X(x)$ on initial conditions, we use rejection sampling to sample from the proposal distribution. We can then use importance sampling correction terms to ensure our estimates are unbiased. If we had access to $f = f^*$, we show (in appendix B) that the variance minimizing proposal distribution[3] is to sample from $q_f(x) \propto f^{1/2}(x)p_X(x)$. Since $f \neq f^*$, we sample from $q_f(x) \propto f^{\alpha}(x)p_X(x)$, where α is tunable. We provide more details and pseudocode in appendix B.

Learning the AVF: One approach to learning f is to run the trained agent on many initial conditions x, and observe whether it failed or not. We can then estimate f, which given an x predicts the probability of failure, using a neural network. This is similar to running one step of the cross entropy method. The issue with this is we get no optimization signal since the trained agent fails very rarely. For example, our final Humanoid agent fails once every 150k episodes after we train it for 300k episodes. The key to our *continuation* approach is to learn f from a family of related agents that fail more frequently. In our instantiation of the idea, we learn f from agents that were previously seen during training, thereby also reusing training data. Since the agent changes during training, we give f the time-step of the agent as an additional input. We give model architecture details in appendix D.1.

4 Experiments

We run experiments on two standard RL domains. In the *Driving* domain, the agent controls a car in the TORCS simulator (Wymann et al., 2000). Initial conditions are defined by the shape of the track. A failure is any collision with a wall. We use an on-policy actor-critic agent as in Espeholt et al. (2018). In the *Humanoid* domain, the agent controls a 21-DoF humanoid body in the MuJoCo simulator (Todorov et al., 2012; Tassa et al., 2018). The initial condition of an experiment is defined by a standing joint configuration, which is sampled from a fixed distribution. A failure is a state where the humanoid has fallen. We use a D4PG agent (Barth-Maron et al., 2018). See appendix C for more details on our agents and environments and appendix D for more details on our experiments.

4.1 Failure Search

We compare our AVF adversary with two baselines on the number of episodes required to find an initial condition that leads to a failure. The baselines evaluated are 1. the naive VMC adversary which evaluates the agent on initial conditions supplied by the environment until observing a failure, 2. a prioritized replay (PR) adversary that evaluates the agent on initial conditions which led to failures during training, most recent first. Additional details and ablation studies on all adversaries are provided in appendix D. The results are summarized in table 1.

The AVF adversary is multiple orders of magnitude more efficient than the VMC adversary. On the Humanoid domain, the VMC adversary required 77 hours to find a single failure, compared to 6

[3]Our optimal proposal distribution differs from the standard optimal form in the rare event estimation literature because our environments have unobserved stochasticity given by z.

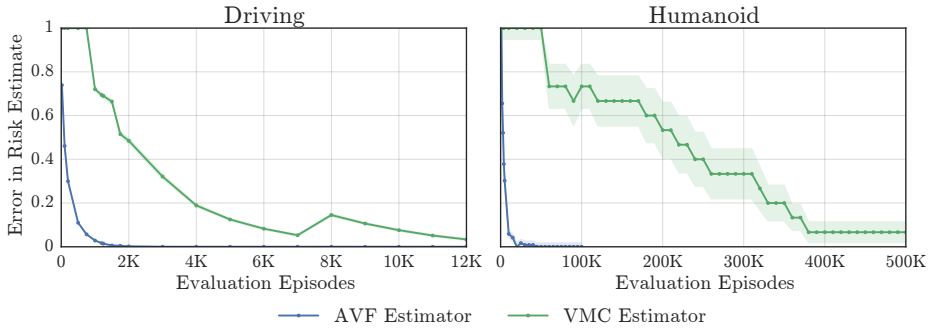

Figure 1: The figure shows the reliability of the AVF and VMC estimators. The x axis shows the number of evaluation episodes, while the y axis shows the probability that $\hat{P}$ does not belong to the interval $(p/3, 3p)$. This probability is obtained by repeating the evaluation multiple times and plotting the fraction of these runs when the estimate is outside of the target interval. The AVF estimator reliably approaches ground truth dramatically faster than the VMC estimator. We show error bars of 2 standard errors, which are difficult to see on the Driving domain.

minutes for the AVF adversary, amounting to a 61-fold speedup after including AVF training time of 70 minutes. Note that for the VMC adversary to have over a 95% chance of detecting even a single failure, we would require over $300,000$ episodes, exceeding the cost of training, which used less than $300,000$ episodes. In practice, evaluation is often run for much less time than training, so the naive approach would often lead to the mistaken impression that such failure modes do not exist.

We also compared our method against a stronger baseline, the PR adversary. The PR adversary focuses evaluation on more difficult situations – initial conditions that led to failure during training. One limitation is that the PR adversary may *never* detect failures, even if they exist. In particular, the agent may have learned to handle all training problem instances, without eliminating all possible failures. This occurred in one out of five runs of Humanoid and all runs of Driving. Even in the median case for Humanoid, our AVF adversary needed 300x fewer episodes than the PR adversary.

4.2 Risk Estimation

We compare our AVF estimator to the standard VMC estimator for risk estimation. For each estimator we report the fraction of runs when the obtained estimates fail to fall in the $(p/3, 3p)$ interval, where p is the failure probability to be estimated. This provides a fairly accurate estimate of the probability of $\hat{P}$ failing to be a $\rho = 3$-approximation of p. Results are summarized in fig. 1. In appendix D.3 we include plots for other choices of ρ, to show that the results are not very sensitive to ρ.

On Driving, the AVF estimator needs only 750 episodes to achieve a 3-approximation with 95% confidence, while the VMC estimator needs $11,000$ episodes. Similarly, on the Humanoid domain, the AVF estimator requires $15,000$ episodes to achieve a 3-approximation with 95% confidence, while the VMC estimator requires $5.1e5$ episodes, a 34-fold improvement.

4.3 Model Selection

The improved efficiency of the AVF estimator has many benefits. One application is to identify the most reliable agent from a finite set. For this illustration, we consider 50 Humanoid agents spaced evenly over the course of training. The goal is to identify the most reliable agent among these. One approach is to estimate the failure probability of each of the 50 agents using a fixed amount of compute for each agent[4]. Then we can select the agent with the lowest estimated failure rate. We could estimate failure rates with either the VMC estimator or the AVF estimator. In the median case, over 5 runs, the VMC based method finds an agent that fails once every 30k episodes, while the AVF based method finds an agent that fails once every 100k episodes. Further, all 5 runs of the AVF based method found more robust agents than all 5 runs of the VMC based method. See Appendix D.4 for more details. Note that the last agent during training failed once every 50k episodes.

[4]This is a crude approach to a bandit problem, but is what people often do when plotting evaluation curves.

5 Conclusion and Future Work

In this work, we argued that standard approaches to evaluating RL agents are highly inefficient in detecting rare, catastrophic failures, which can create a false sense of safety. We believe the approach and results here strongly demonstrate that adversarial testing can play an important role in assessing and improving agents, but are only scratching the surface. We hope that future work on adversarial examples goes beyond norm ball perturbations, and we believe RL or other simulated domains can be useful to this end. We also believe that it is important to move beyond tightly bounded rewards and penalties in RL, and also focus avoiding catastrophic failures. We hope this work lays the groundwork for future research into evaluating and developing robust, deployable agents.

Acknowledgements

We would like to thank Percy Liang, Tengyu Ma, Murray Shanahan, Jackson Broshear, Alhussein Fawzi and our anonymous reviewers for their very useful feedback in preparing this manuscript.

References

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

Brown, T. B., Carlini, N., Zhang, C., Olsson, C., Christiano, P., and Goodfellow, I. (2018). Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352*.

Bucklew, J. A. (2004). *Introduction to Rare Event Simulation*. Springer New York.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lin, Y.-C., Hong, Z.-W., Liao, Y.-H., Shih, M.-L., Liu, M.-Y., and Sun, M. (2017). Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Pritzel, A., Uria, B., Srinivasan, S., Puigdomenech, A., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural episodic control. *arXiv preprint arXiv:1703.01988*.

Song, Y., Shu, R., Kushman, N., and Ermon, S. (2018). Generative adversarial examples. *arXiv preprint arXiv:1805.07894*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE.

Vecerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. A. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR, abs/1707.08817*.

Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., and Sumner, A. (2000). Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4:6.

A Pseudocode for AVF Adversary

Algorithm 1 AVF-guided Search (AVF adversary)

Input: Sample size n
repeat
  Collect random initial conditions $S = \{X_1, \ldots, X_n\}$ where $X_i \sim P_X$, $i \in [n]$
  Select $X = \text{argmax}_{x \in S} f(x)$
  Run an experiment to generate outcome $C = c(X, Z)$
until $C = 1$

B Detailed Explanation of Risk Estimation Approach

Our failure probability estimation method uses importance sampling (IS) (e.g., Section 4.2, Bucklew 2004) where the distribution P_X is replaced by a *proposal distribution* Q. For $t \in [n]$, the proposal distribution is used to generate random initial condition $X_t \sim Q$, then an experiment is performed to generate $C_t = c(X_t, Z_t)$. The failure probability p is estimated using the sample mean

$$\hat{P} = \frac{1}{n} \sum_{t=1}^{n} W_t \, c(X_t, Z_t) \,,$$

where $W_t = \frac{p_X(X_t)}{q(X_t)}$ is the *importance weight* of the t-th sample. Here, p_X denotes the density of P_X and q denotes the density of Q.[5] Let $U_t = W_t c(X_t, Z_t)$. As is well-known, under the condition that $p_X(x) f^*(x) = 0$ whenever $q(x) = 0$, we have that $\mathbb{E}[U_t] = p$, and hence $\mathbb{E}[\hat{P}] = p$. Given that $\hat{P}$ is unbiased for any proposal distribution, one natural objective is to choose a proposal distribution which minimizes the variance of the estimator $\hat{P}$.

Proposition B.1. *For $f : \mathcal{X} \to [0, 1]$ such that $P_X(f^{1/2}) := \int f^{1/2}(x') P_X(dx') > 0$ let Q_f be defined as the distribution over $\mathcal{X}$ whose density q_f is*

$$q_f(x) = \frac{f^{1/2}(x) p_X(x)}{P_X(f^{1/2})} \,.$$

Then, the variance minimizing proposal distribution Q^* is Q_{f_*}.

Proof. Since $U_1, \ldots, U_n$ are independent and identically distributed, $\text{Var}[\hat{P}] = \text{Var}[U_t]/n$ for any $t \in [n]$. Hence, the variance of $\hat{P}$ is minimized when the variance of U_t is minimized. Since $\mathbb{E}[U_t] = p$, this variance is minimized when $\mathbb{E}[U_t^2] = \mathbb{E}[W_t^2 \, c(X_t, Z_t)] = \mathbb{E}[(W_t f_*^{1/2}(X_t))^2]$ is minimized, where we used the definition of f_* and that c is binary-valued. However, this is nothing but the second moment of the importance sampling estimator that uses the proposal distribution Q to estimate $\int P_X(dx) f_*^{1/2}(x)$. Using Cauchy-Schwarz for integrals, this is minimized by the proposal distribution whose density with respect to P_X is proportional to $f_*^{1/2}$, leading to the desired claim. Note that from the definition of Q_f it follows that $\frac{p_X(x)}{q_f(x)} = f^{-1/2}(x) P_X(f^{1/2})$ when $q_f(x) > 0$. □

The above result motivates us to suggest using the distribution Q_f as the proposal distribution of an IS estimator. Note that the choice of f (as long as it is bounded away from zero) only influences the efficiency of the method, but not its correctness. It remains to specify how to sample from Q_f and how to calculate the importance weights. For sampling from Q_f, we propose to use the *rejection sampling method* (Section 1.2.2, Bucklew 2004) with the proposal distribution chosen to be P_X: First, $X \sim P_X$ is chosen, which is accepted by probability $f^{1/2}(X)$. This is repeated until a sample is accepted:

Proposition B.2. *A rejection sampling procedure that accepts a random instance $X \sim P_X$ with probability $f^{1/2}(X)$ produces a sample from Q_f.*

[5] Here, we implicitly assume that these measures have a density with respect to a common measure. As it turns out, this is not a limiting assumption, but this goes beyond the scope of the present article.

Proof. Let U be uniformly distributed on the $[0,1]$ interval and independent of X. Clearly, the probability that a sample produced by rejection sampling falls into a set $A \subset \mathcal{X}$ is $\mathbb{P}(X \in A | U \leq f^{1/2}(X))$. Now,

$$\mathbb{P}(X \in A | U \leq f^{1/2}(X)) = \frac{\mathbb{P}(X \in A, U \leq f^{1/2}(X))}{\mathbb{P}(U \leq f^{1/2}(X))} = \frac{\int_A \mathbb{P}(U \leq f^{1/2}(x)) P_X(x)}{\mathbb{P}(U \leq f^{1/2}(X))}$$
$$= \frac{\int_A f^{1/2}(x) P_X(x)}{\mathbb{P}(U \leq f^{1/2}(X))} .$$

Since $\mathbb{P}(X \in \cdot | U \leq f^{1/2}(X))$ is a probability measure on $\mathcal{X}$, it follows that the unconditional acceptance probability satisfies $\mathbb{P}(U \leq f^{1/2}(X)) = \int_{\mathcal{X}} f^{1/2}(x) P_X(x)$, thus showing that the distribution of accepted points is Q_f as required. □

To increase the robustness of the sampling procedure against errors introduced by $f \neq f_*$, we introduce a "hyperparameter" $\alpha > 0$ so that q_f is redefined to be proportional to f^α. Note that $\alpha = 1/2$ is what our previous result suggests. However, if f is overestimated, a larger value of α may work better, while if f is underestimated, a smaller value of α may work better.

The pseudocode of the full procedure is given as algorithm 2.

Algorithm 2 AVF-guided risk estimator (AVF estimator)

Input: AVF f, budget $T \in \mathbb{N}$ and tuning parameter $\alpha > 0$
Returns: Failure probability estimate $\hat{P}$
Initialize $S \leftarrow 0$
for $t = 1$ **to** T **do**
  repeat
    Sample proposal instance $X_t \sim P_X$
    Accept X_t with probability $f^\alpha(X_t)$
  until accepting initial condition X_t
  Evaluate the agent on X_t and observe $C_t = c(X_t, Z_t)$
  $S \leftarrow S + C_t / f^\alpha(X_t)$
end for
Compute normalization constant $Z \leftarrow \frac{1}{m} \sum_{i=1}^m f^\alpha(X_i')$ for $X_i' \sim P_X$, $m \gg T$
return $\hat{P} \leftarrow ZS/T$

C Agent Training Details

C.1 Environments

For the *Driving* domain, we use the TORCS 3D car racing game (Wymann et al., 2000) with settings corresponding to the "Fast car, no bots" setting in Mnih et al. (2016). At each step, the agent receives an 15-dimensional observation vector summarizing its position, velocity, and the local geometry of the track. The agent receives a reward proportional to its velocity along the center of the track at its current position, while collisions with a wall terminate the episode and provide a large negative reward.

Each problem instance is a track shape, parameterized by a 12-dimensional vector encoding the locations and curvatures of waypoints specifying the track. Problem instances are sampled by randomly sampling a set of waypoints from a fixed distribution, and rejecting any tracks with self-intersections.

For the *Humanoid* domain, we use the Humanoid Stand task from Tassa et al. (2018). At each step, the agent receives a 67-dimensional observation vector summarizing its joint angles and velocities, and the locations of various joints in Cartesian coordinates. The agent receives a reward proportional to its head height. If the head height is below 0.7m, the episode is terminated and the agent receives a large negative reward.

Each problem instance is defined by an initial standing pose. To define this distribution, we sample $1e5$ random trajectories from a separately trained D4PG agent. We sample a random state from these

trajectories, ignoring the first 10 seconds of each trajectory, as well as any trajectories terminating in failure, to ensure all problem instances are feasible. Together, we obtain $6e7$ problem instances, so it is unlikely that any problem instance at train or test time has been previously seen.

C.2 Agents

We now describe the agents we used for evaluation on each task. The focus of this work is not on training agents, and hence we leave the agents fixed for all experiments. However, we did make some modifications to decrease agent failure probabilities, as we are most interested in whether we can design effective adversaries when failures are rare, and thus there is not much learning signal to guide the adversary.

On Driving, we use an asynchronous batched implementation of Advantage Actor-Critic, using a V-trace correction, following Espeholt et al. (2018). We use Population-Based Training, and evolve the learning rate and entropy cost weights Jaderberg et al. (2017), with a population size of 5. Each learner is trained for $1e9$ actor steps, which takes 4 hours distributed over 100 CPU workers and a single GPU learner. Since episodes are at most 3600 steps, this equates to roughly $270e3$ episodes per learner, and $1.3e6$ episodes for the entire population. At test time, we take the most likely predicted action, rather than sampling, as we found it decreased the failure probability by roughly half. Additionally, we used a hand-crafted form of adversarial training by training on a more difficult distribution of track shapes, with sharper turns than the original distribution, since this decreased failure probability roughly 20-fold.

On Humanoid, we use a D4PG agent, using the same hyperparameters as Barth-Maron et al. (2018). The agent is trained for $4e6$ learner steps, which corresponds to between $250e6$ and $300e6$ actor steps, which takes 8 hours distributed over 32 CPU workers and a single GPU learner. Since episodes are at most 1000 steps, this equates to roughly $275e3$ episodes. We use different exploration rates on actors, as in Horgan et al. (2018), with noise drawn from a normal distribution with standard deviation σ evenly spaced from 0.0 to 0.4. We additionally use demonstrations from the agent described in the previous section, which was used for defining the initial pose distribution, following Vecerík et al. (2017). In particular, we use 1000 demonstration trajectories, and use demonstration data for half of each batch to update both the critic and policy networks. This results in a roughly 4-fold improvement in the failure rate.

D Experimental Details

D.1 AVF Details

When constructing the training datasets, we ignore the beginning of training during which the agent fails very frequently. This amounts to using the last $150,000$ episodes of data on Driving and last $200,000$ on Humanoid. To include information about the training iteration of the agent, we simply include a single real value, the current training iteration divided by the maximum number of training iterations. Similarly, for noise applied to the policy, we include the amount of noise divided by the maximum amount of noise. These are all concatenated before applying the MLP. We train both AVF models to minimize the cross-entropy loss, with the Adam optimizer (Kingma and Ba, 2014), for $20,000$ and $40,000$ iterations on Driving and Humanoid respectively, which requires 4.5 and 50 minutes respectively on a single GPU.

On Driving, the AVF architecture uses a 4-layer MLP with 32 hidden units per layer and a single output, with a sigmoid activation to convert the output to a probability. On Humanoid, since failures of the most robust agents are very rare, we use a simplified Differentiable Neural Dictionary architecture (Pritzel et al., 2017) to more effectively leverage the limited number of positive training examples. In particular, to classify an input x, the model first retrieves the $K = 32$ nearest neighbors and their corresponding labels (x_i, y_i) from the training set. The final prediction is then a weighted average $(b + \sum_{i:y_i=1} w_i)/(2b + \sum_i w_i)$, where b is a learned pseudocount, which allows the model to make uncertain predictions when all weights are close to 0. To compute weights w_i, each point is embedded by a 1-layer MLP f into 16 dimensions, and the weight of each neighbor is computed $w_i = \kappa(f(x), f(x_i))$, where κ is a Gaussian kernel, $\kappa(x, y) = exp(\|x - y\|_2^2/2)$. We tried different hyperparameters for the number of MLP layers on Driving, and the number of neighbors on Humanoid, and selected based on a held-out test set using data collected during training.

Sample size n	$n_0/10$ Cost	$n_0/10$ Speedup	n_0 Cost	$10n_0$ Cost	$10n_0$ Speedup
Driving	$4/23/61$	$0.18/0.22/0.59$	$3/5/11$	$2/3/6$	$1.1/1.7/2.4$
Humanoid	$57/173/220$	$0.16/0.23/0.28$	$19/33/56$	$7/11/25$	$1.8/2.4/3.1$

Table 2: **Adversary hyperparameter sensitivity analysis.** We show the performance of Algorithm 1 for varying values of the sample size parameter n. The middle column shows results for $n = n_0$, the value we used in our experiments, and we compare to a factor of 10 larger and smaller. Costs represent expected number of episodes until a failure, and speedups are relative to when $n = n_0$. As before, each cell shows min, median, and max values across 5 agents. On both domains, applying greater optimization pressure improves results.

In particular, to match real-world situations, we did not select hyperparameters based on either failure search or risk estimation experiments.

D.2 Failure Search

We ran the VMC adversary for $5e6$ experiments for each agent. The expected cost is simply $1/p$, where p is the fraction of experiments resulting in failures. For the AVF adversary, we ran $2e4$ experiments for each agent, since failures are more common, so less experiments are necessary to estimate the failure probabilities precisely. For the PR adversary, the adversary first selected problem instances which caused failures on the actor with the least noise, most recent first, followed by the actor with the second least noise, and so on. We also tried a version which did not account for the amount of noise and simply ran the most recent failures first, which was slightly worse.

For the failure search experiments on the Humanoid domain, we always evaluate the best version of each agent according to the AVF model selection procedure from Section 4.3. We chose this because we are most interested in whether we can design effective adversaries when failures are rare, and simply choosing the final agent is ineffective, as discussed in Appendix D.4.

We now discuss the sample size parameter n in Algorithm 1. As discussed, using larger values of n applies more selection pressure in determining which problem instances to run experiments on, while using smaller values of n draws samples from a larger fraction of $\mathcal{X}$, and thus provides some robustness to inaccuracies in the learned f. Of course, other more sophisticated approaches may be much more effective in producing samples from diverse regions of $\mathcal{X}$, but we only try this very simple approach. For our experiments, we use $n = 1000$ for Driving and $n = 10000$ for Humanoid. We did not perform any hyperparameter selection on n, in order to match the real-world setting where the experimenter has only a fixed budget of experiments, and wishes to find a single failure. However, we include additional experiments to study the robustness of the AVF adversary to the choice of n. These are summarized in Table 2.

We observe that on both domains, applying greater optimization pressure improves results, over the default choices for n we used. However, relative to the improvements over the VMC adversary, these differences are small, and the AVF adversary is dramatically faster than the VMC adversary for all choices of n we tried.

D.3 Risk Estimation Details

In our paper, we showed that the AVF estimator can estimate the failure probability of an agent much faster than the VMC estimator. In particular, we showed in both the Driving and Humanoid domains that our estimator requires an order of magnitude less episodes to achieve a 3-approximation at any given confidence level. In general, we may be interested in a ρ-approximation, that is we might want the estimates to fall in the range $(p/\rho, p\rho)$ where p is the probability of failure that we wish to measure. One might ask whether our results are sensitive to the choice of ρ. In other words, did we select $\rho = 3$ so that our results look good? To address this concern, we include results for lower and higher choices of ρ. We see that the AVF estimator performs significantly better than the VMC estimator across choices of ρ.

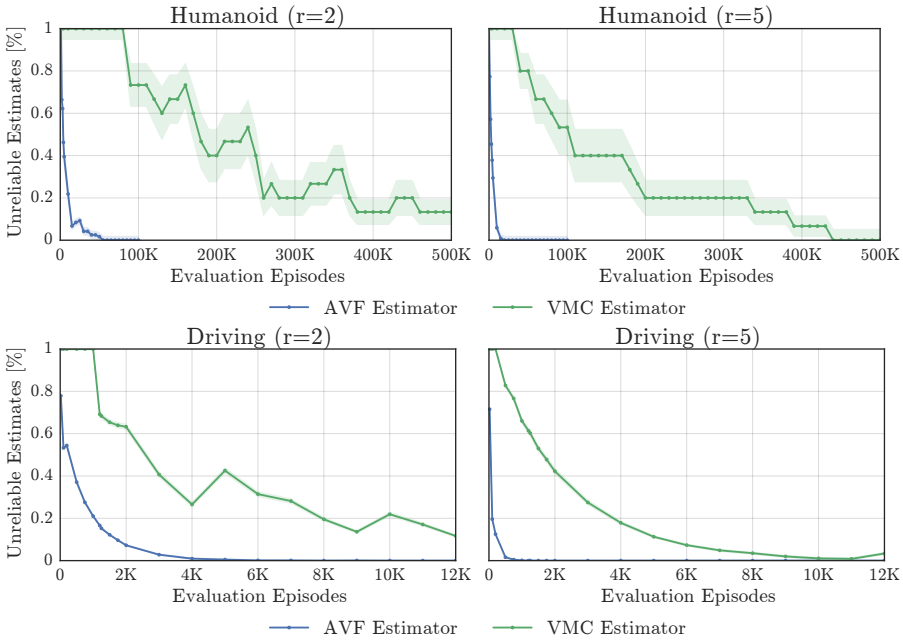

Figure 2: This figure shows the reliability of the AVF and VMC estimators for various choices of $r = \rho$. As before, the x axis shows the number of evaluation episodes. The y-axis shows the probability that $\hat{P}$ does not belong in the interval $(p/r, pr)$. The probability is obtained by running the evaluation multiple times, and we include error bars of 2 standard errors. Note that the curves are not smooth especially for the VMC estimator. This is not because of the uncertainty in the plots but is a property of ρ-estimators when the sample size is small, that is on the order of $1/p$. For such sample sizes, increasing the sample size does not monotonically improve the estimator's accuracy.

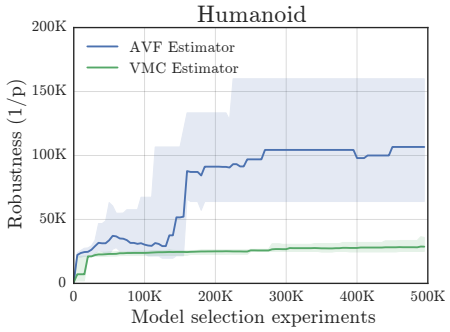

Figure 3: **Model selection**: We plot the expected number of episodes until failure, i.e. $1/p$ where p is the probability of failure, for the best policy selected by the AVF vs. VMC estimators. The error bars show the min/max over 5 random seeds, while the solid lines correspond to the averaged failure probability. The VMC estimator largely maintains a uniform distribution over many of the policies, whereas the AVF estimator quickly eliminates the worst policies.

D.4 Model Selection

We show the results for VMC and AVF based model selection procedures in Figure 3. We ran each procedure 5 times, with different random seeds.

Measuring ground truth failure probabilities. In Figure 4, we show the "ground truth" failure probabilities for each of the 50 model's robustness. Each bar represents a version of the agent at a different point in training, starting from step $5e5$, and taking a new version every $7e4$ learner steps (so that there are 50 agents in total). Failure probabilities are computed by taking the fraction of experiments resulting in failures, using $160,000$ experiments per agent. We show robustness $(1/p)$

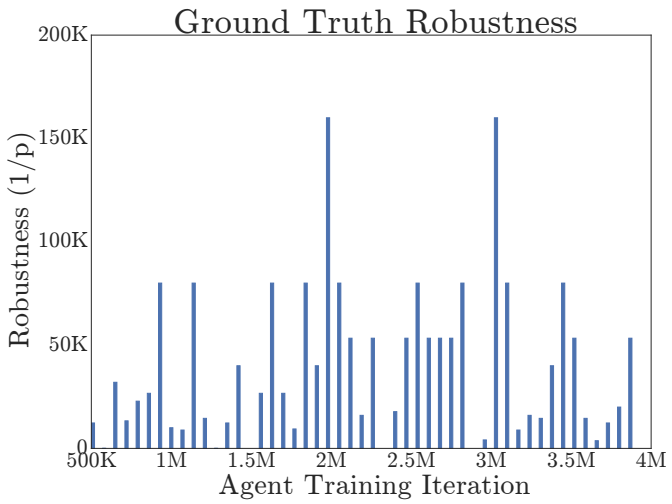

Figure 4: We show the robustness for agents taken from different points in training. The failure probability of each agent is estimated using $160,000$ experiments with VMC. Note that the robustness does not improve monotonically, and that simply choosing the final agent would not provide a robust agent.

rather than failure probabilities, so that it is easier to see differences between the most robust models. Robustness should be interpreted as the expected number of experiments until seeing a failure.

We note that the failure probabilities we report after model selection in Figure 3 are conservative, for two reasons. First, for the two agents which did not fail in our ground truth measurements, we use a failure probability of $1/160,000$ rather than 0. Second, because our ground truth measurements are only noisy measurements of the true failure probability for each agent, it is possible that even an optimal model selection algorithm would not pick the optimal agent according to our ground truth measurements. We could run the ground truth measurements for longer, but already, this required $160,000 * 50 = 8e6$ experiments, or 231 CPU-days. Nonetheless, even with these conservative estimates, the AVF estimator showed significant improvements over using the VMC estimator for model selection.

Non-monotonicity of robustness over training. Finally, we note that the robustness is not monotonically increasing, and thus simply taking the last agent (or randomly choosing from the last few) would be a worse model selection strategy than uniform exploration with the AVF estimator. This differs from the standard setting in deep reinforcement learning, with tightly bounded rewards, where expected return usually (roughly) monotonically increases over the course of training. As discussed in the body of the paper, when the agent is optimized using VMC, failures are observed very rarely, and thus the optimization process does not effectively decrease the failure probability, which is consistent with the non-monotonicity we observe here.