Learning Human Objectives by Evaluating Hypothetical Behavior

Siddharth Reddy [1] **Anca D. Dragan** [1] **Sergey Levine** [1] **Shane Legg** [2] **Jan Leike** [2]

Abstract

We seek to align agent behavior with a user's objectives in a reinforcement learning setting with unknown dynamics, an unknown reward function, and unknown unsafe states. The user knows the rewards and unsafe states, but querying the user is expensive. We propose an algorithm that safely and efficiently learns a model of the user's reward function by posing 'what if?' questions about hypothetical agent behavior. We start with a generative model of initial states and a forward dynamics model trained on off-policy data. Our method uses these models to synthesize hypothetical behaviors, asks the user to label the behaviors with rewards, and trains a neural network to predict the rewards. The key idea is to actively synthesize the hypothetical behaviors from scratch by maximizing tractable proxies for the value of information, without interacting with the environment. We call this method *reward query synthesis via trajectory optimization* (ReQueST). We evaluate ReQueST with simulated users on a state-based 2D navigation task and the image-based Car Racing video game. The results show that ReQueST significantly outperforms prior methods in learning reward models that transfer to new environments with different initial state distributions. Moreover, ReQueST safely trains the reward model to detect unsafe states, and corrects reward hacking before deploying the agent.

1. Introduction

Users typically specify objectives for reinforcement learning (RL) agents through scalar-valued reward functions (Sutton & Barto, 2018). While users can easily define reward functions for tasks like playing games of Go or StarCraft, users may struggle to describe practical tasks like driving cars or controlling robotic arms in terms of rewards (Hadfield-Menell et al., 2017). Understanding user objectives in these settings can be challenging – not only for machines, but also for humans modeling each other and introspecting on themselves (Premack & Woodruff, 1978).

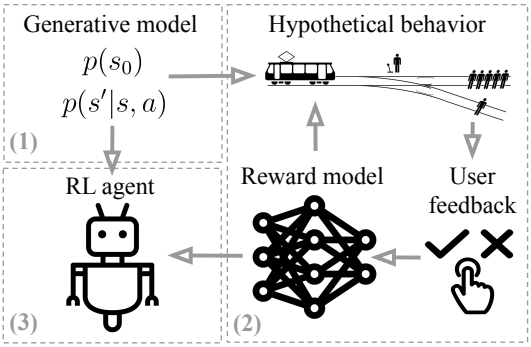

Figure 1. Our method learns a reward model from user feedback on hypothetical behaviors, then deploys a model-based RL agent.

For example, consider the *trolley problem* (Foot, 1967): if you were the train conductor in Figure 1, presented with the choice of either allowing multiple people to come to harm by letting the train continue on its current track, or harming one person by diverting the train, what would you do? The answer depends on whether your value system leans toward consequentialism or deontological ethics – a distinction that may not be captured by a reward function designed to evaluate common situations, in which ethical dilemmas like the trolley problem rarely occur. In complex domains, the user may not be able to anticipate all possible agent behaviors and specify a reward function that accurately describes user preferences over those behaviors.

We address this problem by actively synthesizing hypothetical behaviors from scratch, and asking the user to label them with rewards. Figure 1 describes our algorithm: using a generative model of initial states and a forward dynamics model trained on off-policy data, we synthesize hypothetical behaviors, ask the user to label the behaviors with rewards, and train a neural network to predict the rewards. We repeat this process until the reward model converges, then deploy a model-based RL agent that optimizes the learned rewards.

The key idea in this paper is synthesizing informative hypo-

[1]University of California, Berkeley [2]DeepMind. Correspondence to: Siddharth Reddy <sgr@berkeley.edu>, Jan Leike <leike@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

theticals. Ideally, we would generate these hypotheticals by optimizing the value of information (VOI; Savage, 1954), but the VOI is intractable for real-world domains with high-dimensional, continuous states. Instead, we use trajectory optimization to produce four types of hypotheticals that improve the reward model in different ways: behaviors that (1) maximize reward model uncertainty, to elicit labels that are likely to change the updated reward model's outputs; (2) maximize predicted rewards, to detect and correct reward hacking; (3) minimize predicted rewards, to safely explore unsafe states; or (4) maximize novelty of trajectories regardless of predicted rewards, to improve the diversity of the training data. To ensure that the hypothetical trajectories remain comprehensible to the user and resemble realistic behaviors, we use a generative model of initial states and a forward dynamics model for regularization. We call this method *reward query synthesis via trajectory optimization* (ReQueST).

Our primary contribution is an algorithm that synthesizes hypothetical behaviors in order to safely and efficiently train neural network reward models in environments with high-dimensional, continuous states. We evaluate ReQueST with simulated users in three domains: MNIST classification, a state-based 2D navigation task, and the image-based Car Racing video game in the OpenAI Gym. Our experiments show that ReQueST learns robust reward models that transfer to new environments with different initial state distributions, achieving at least 2x better final performance than baselines adapted from prior work (e.g., see Figure 4). In the navigation task, ReQueST safely learns to classify 100% of unsafe states as unsafe and deploys an agent that never visits unsafe states, while the baselines fail to learn about even one unsafe state and deploy agents with a failure rate of 75%.

2. Related Work

In this work, we align agent behavior with a user's objectives by learning a model of the user's reward function and training the agent via RL (Ng & Russell, 2000; Ziebart et al., 2008; Leike et al., 2018). The idea behind modeling the user's reward function – as opposed to the user's policy (Pomerleau, 1991; Ross et al., 2011; Ho & Ermon, 2016), value function (Dvijotham & Todorov, 2010; Warnell et al., 2018; Reddy et al., 2018), or advantage function (MacGlashan et al., 2017) – is to acquire a compact, transferable representation of the user's objectives; not just in the training environment, but also in new environments with different dynamics or initial states.

The closest prior work is on active learning methods for learning rewards from pairwise comparisons (Sadigh et al., 2017; Bıyık & Sadigh, 2018; Wirth et al., 2017), critiques (Cui & Niekum, 2018), demonstrations (Ibarz et al., 2018; Brown et al., 2018), designs (Mindermann et al., 2018), and numerical feedback (Daniel et al., 2014). ReQueST differs in three key ways: it produces hypothetical trajectories using a generative model, in a way that enables trading off between producing realistic vs. informative queries; it optimizes queries not only to reduce model uncertainty, but also to detect reward hacking and safely explore unsafe states; and it scales to learning neural network reward models that operate on high-dimensional, continuous state spaces.

ReQueST shares ideas with prior work (Saunders et al., 2018; Prakash et al., 2019) on learning to detect unsafe behaviors by initially seeking out catastrophes, selectively querying the user, and using model-based RL. ReQueST differs primarily in that it learns a complete task specification, not just an unsafe state detector. ReQueST is also complementary to prior work on safe exploration, which typically assumes a known reward function and side constraints, and focuses on ensuring that the agent never visits unsafe states during policy optimization (Dalal et al., 2018; Garcıa & Fernández, 2015).

3. Learning Rewards from User Feedback on Hypothetical Behavior

We formulate the reward modeling problem as follows. We assume access to a training environment that follows a Markov decision process (MDP; Sutton & Barto, 2018) with unknown state transition dynamics $\mathcal{T}$, unknown initial state distribution $\mathcal{S}_0^{\text{train}}$, and an unknown reward function R that can be evaluated on specific inputs by querying the user. We learn a model of the reward function $\hat{R}$ by querying the user for reward signals. At test time, we train an RL agent with the learned reward function $\hat{R}$ in a new environment with the same dynamics $\mathcal{T}$, but a potentially different initial state distribution $\mathcal{S}_0^{\text{test}}$. The goal is for the agent to perform well in the test environment with respect to the true reward function R.

Our approach to this problem is outlined in Figure 1, and can be split into three steps. In step (1) we use off-policy data to train a generative model $p_{\boldsymbol{\phi}}(\tau)$ that can be used to evaluate the likelihood of a trajectory $\tau = (s_0, a_0, s_1, a_1, ..., s_T)$, which enables us to synthesize hypothetical trajectories that can be shown to the user. In step (2) we produce synthetic trajectories, which consist of sequences of state transitions (s, a, s'), that seek out different kinds of hypotheticals (detailed in Section 3.3). We ask the user to label each transition with a scalar reward $R(s, a, s')$, and fit a reward model $\hat{R}(s, a, s')$ using standard supervised learning techniques. In step (3) we use standard RL methods to train an agent using the learned rewards.

3.1. Learning a Generative Model of Trajectories

In order to generate hypothetical behaviors with uncertain rewards, we first learn a generative model of initial states and a forward dynamics model. We use these models to synthesize plausible trajectories, without requiring that an agent actually perform those behaviors in the real environment.

In step (1) we collect off-policy data by interacting with the training environment in an unsupervised fashion; i.e., without the user in the loop. To simplify our experiments, we sample trajectories τ by following random policies that explore a wide variety of states. We use the observed trajectories to train a likelihood model,

$$p_{\boldsymbol{\phi}}(\tau) \propto p_{\boldsymbol{\phi}}(s_0) \prod_{t=0}^{T-1} p_{\boldsymbol{\phi}}(s_{t+1}|s_t, a_t), \tag{1}$$

where $p_{\boldsymbol{\phi}}(s_0)$ models the initial state distribution, $p_{\boldsymbol{\phi}}(s_{t+1}|s_t, a_t)$ models the forward dynamics, and $\boldsymbol{\phi}$ are the model parameters (e.g., neural network weights). We train the model $p_{\boldsymbol{\phi}}$ using maximum-likelihood estimation, given the sampled trajectories. As described in the next section, we use the likelihood model to generate realistic synthetic trajectories to show to the user.

In environments with high-dimensional, continuous states, such as images, we also train a state encoder $f_{\boldsymbol{\phi}} : \mathcal{S} \to \mathcal{Z}$ and decoder $f_{\boldsymbol{\phi}}^{-1} : \mathcal{Z} \to \mathcal{S}$, where $\mathcal{S} = \mathbb{R}^n$, $\mathcal{Z} = \mathbb{R}^d$, and $d << n$. As described in Section 3.4, embedding states in a low-dimensional latent space $\mathcal{Z}$ enables us to synthesize realistic hypotheticals. In our experiments, we train $f_{\boldsymbol{\phi}}$ and $f_{\boldsymbol{\phi}}^{-1}$ using the variational auto-encoder method (VAE; Kingma & Welling, 2013).

3.2. Representing the Reward Model as a Classifier

Our goal is to learn a model $\hat{R}$ of the user's reward function. In step (2) we represent $\hat{R}$ by classifying state transitions as good, unsafe, or neutral – similar to Cui & Niekum (2018) – and assigning a known, constant reward to each of these three categories:

$$\hat{R}(s, a, s') = \sum_{c \in \{\text{good,unsafe,neutral}\}} p_{\boldsymbol{\theta}}(c|s, a, s') R_c, \tag{2}$$

where $p_{\boldsymbol{\theta}}(c|s, a, s') = \frac{1}{m} \sum_{i=1}^{m} p_{\boldsymbol{\theta}_i}(c|s, a, s')$ is the mean of an ensemble of m classifiers $\{p_{\boldsymbol{\theta}_i}\}_{i=1}^{m}$, and $\boldsymbol{\theta}_i$ are the weights of the i-th neural network in the ensemble. R_c is the constant reward for any state transition in class c, where $R_{\text{unsafe}} \leq R_{\text{neutral}} \leq R_{\text{good}}$. Modeling the reward function as a classifier simplifies our experiments and makes it easier for the user to provide labels. In principle, our method can also work with other architectures, such as a regression model $\hat{R} = R_{\boldsymbol{\theta}}$.

3.3. Designing Objectives for Informative Queries

Our approach to reward modeling involves asking the user to label trajectories with reward signals. In step (2) we synthesize query trajectories to elicit user labels that are informative for learning the reward model.

To generate a useful query, we synthesize a trajectory τ that maximizes an acquisition function (AF) denoted by $J(\tau)$. The AF evaluates how useful it would be to elicit reward labels for τ, then update the reward model given the newly-labeled data. Since we do not assume knowledge of the initial state distribution of the test environment where the agent is deployed, we cannot optimize the ideal AF: the value of information (VOI; Savage, 1954), defined as the gain in performance of an agent that optimizes the updated reward model in the test environment. Prior work on active learning tackles this problem by optimizing proxies for VOI (Settles, 2009). In this work, we synthesize a separate trajectory for each of four AFs defined in the following paragraphs.

Maximizing uncertainty. The first AF $J_u(\tau)$ implements one of the simplest query selection strategies from the active learning literature: uncertainty sampling (Lewis & Gale, 1994). The idea is to elicit labels for examples that the model is least certain how to label, and thus reduce model uncertainty. To do so, we train an ensemble of neural network reward models, and generate trajectories that maximize the disagreement between ensemble members. Following Lakshminarayanan et al. (2017), we measure ensemble disagreement using the average KL-divergence between the output of a single ensemble member and the ensemble mean (Equation 5 in the appendix).

Maximizing reward. The second AF $J_+(\tau)$ is intended to detect examples of false positives, or 'reward hacking': behaviors for which the reward model incorrectly outputs high reward (Amodei et al., 2016; Christiano et al., 2017). The idea is to show the user what the reward model predicts to be good behavior, with the expectation that some of these behaviors are actually suboptimal, and will be labeled as such by the user. To do so, we simply synthesize trajectories that maximize $J_+(\tau) = \sum_{(s,a,s') \in \tau} \hat{R}(s, a, s')$.

Minimizing reward. The third AF $J_-(\tau)$ is intended to augment the training data with more examples of unsafe states than would normally be encountered, e.g., by a reward-maximizing agent acting in the training environment. The idea is to show the user what the reward model considers unsafe behavior, so the user can confirm whether the model has captured the correct notion of unsafe states. To do so, we produce trajectories that maximize $J_-(\tau) = -J_+(\tau)$.

Maximizing novelty. The fourth AF $J_n(\tau)$ is intended to produce novel trajectories that differ from those already in the training data, regardless of their predicted reward;

akin to prior work on geometric AFs (Sener & Savarese, 2018). This is especially helpful early during training, when uncertainty estimates are not accurate, and the reward model has not yet captured interesting notions of reward-maximizing and reward-minimizing behavior. To do so, we produce trajectories τ that maximize the distance between τ and previously-labeled trajectories $\tau' \in \mathcal{D}$, $J_n(\tau) = \frac{1}{|\mathcal{D}|}\sum_{\tau' \in \mathcal{D}} d(\tau, \tau')$. In this work, we use a distance function d that computes the Euclidean distance between state embeddings (Equation 6 in the appendix).

3.4. Query Synthesis via Trajectory Optimization

The four AFs defined in the previous section specify the types of hypotheticals we would like to show the user. In this section, we discuss how to optimize each AF, then use the resulting hypotheticals to learn a reward model.

We synthesize a query trajectory,

$$\tau_{\text{query}} = \max_{z_0, a_0, z_1, \ldots, z_T} J(\tau) + \lambda \log p_{\boldsymbol{\phi}}(\tau), \qquad (3)$$

where z_t is the embedding of state s_t in the latent space of the encoder f trained in step (1), $\tau = (f^{-1}(z_0), a_0, f^{-1}(z_1), a_1, ..., f^{-1}(z_T))$ is the decoded trajectory, J is the acquisition function (Section 3.3), $\lambda \in \mathbb{R}_{\geq 0}$ is a regularization constant, and $p_{\boldsymbol{\phi}}$ is the generative model of trajectories (Section 3.1). In this work, we assume $p_{\boldsymbol{\phi}}(\tau)$ is differentiable, and optimize τ_{query} using Adam (Kingma & Ba, 2014). Optimizing latent states z instead of high-dimensional states s reduces computational requirements, and regularizes the optimized states to be more realistic.

The regularization constant λ from Equation 3 controls the trade-off between how realistic τ_{query} is and how aggressively it maximizes the AF. Setting $\lambda = 0$ can result in query trajectories that are incomprehensible to the user and unlikely to be seen in the test environment, while setting λ to a high value can constrain the query trajectories from seeking interesting hypotheticals. The experiments in Section A.3 in the appendix analyze this trade-off.

Our reward modeling algorithm is summarized in Algorithm 1. Given a generative model of trajectories $p_{\boldsymbol{\phi}}(\tau)$, it generates one query trajectory τ_{query} for each of the four AFs, asks the user to label the states in the query trajectories, retrains the reward model ensemble $\{\boldsymbol{\theta}_i\}_{i=1}^m$ on the updated training data $\mathcal{D}$ using maximum-likelihood estimation, and repeats this process until the user is satisfied with the outputs of the reward model. The ablation study in Section 4.5 analyzes the effect of using different subsets of the four AFs to generate queries.

3.5. Deploying a Model-Based RL Agent

Given the learned reward model $\hat{R}$, the agent can, in principle, be trained using any RL algorithm in step (3). Since our method learns a forward dynamics model in step (1), model-based RL is a good fit for step (3). In this work, we deploy an agent π_{mpc} that combines planning with model-predictive control (MPC):

$$\pi_{\text{mpc}}(a|s) = \mathbb{1}\Big[a = \arg\max_{a_0} \max_{a_1,\ldots,a_H} \hat{R}(s, a_0) + \hat{R}(\mathbb{E}_{\boldsymbol{\phi}}[s_1|s, a_0], a_1) + ... + \hat{R}(\mathbb{E}_{\boldsymbol{\phi}}[s_H|s, a_0, a_1, ..., a_{H-1}], a_H)\Big], \qquad (4)$$

where the future states $\mathbb{E}_{\boldsymbol{\phi}}[s_t|s, a_0, a_1, ..., a_{t-1}]$ are predicted using the forward dynamics model $p_{\boldsymbol{\phi}}$ trained in step (1), H is the planning horizon, and $\hat{R}$ is the reward model trained in step (2). We solve the optimization problem in the right-hand side using Adam (Kingma & Ba, 2014).

Algorithm 1 Reward Query Synthesis via Trajectory Optimization (ReQueST)

Require $\lambda, p_{\boldsymbol{\phi}}$
Initialize $\mathcal{D} \leftarrow \emptyset$
while $\boldsymbol{\theta}$ not converged **do**
  for $J \in \{J_u, J_+, J_-, J_n\}$ **do**
    $\tau_{\text{query}} \leftarrow \max_\tau J(\tau) + \lambda \log p_{\boldsymbol{\phi}}(\tau)$
    for $(s, a, s') \in \tau_{\text{query}}$ **do**
      $c \leftarrow c \sim p_{\text{user}}(c|s, a, s')$ {Query the user}
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, s', c)\}$
    end for
  end for
  for $i \in \{1, 2, ..., m\}$ **do**
    $\boldsymbol{\theta}_i \leftarrow \arg\max_{\boldsymbol{\theta}_i} \sum_{(s,a,s',c)\in\mathcal{D}} \log p_{\boldsymbol{\theta}_i}(c|s, a, s')$
  end for
end while
Return reward model $\hat{R}$ {Defined via $\boldsymbol{\theta}$ in Equation 2}

4. Experimental Evaluation

We seek to answer the following questions. **Q1**: Does synthesizing hypothetical trajectories elicit more informative labels than rolling out a policy in the training environment? **Q2**: Can our method detect and correct reward hacking? **Q3**: Can our method safely learn about unsafe states? **Q4**: Do the proposed AFs improve upon random sampling from the generative model? **Q5**: How does the regularization constant λ control the trade-off between realistic and informative queries? **Q6**: How much do each of the four AFs contribute to performance? To answer these questions under ideal assumptions, we run experiments in MNIST (LeCun, 1998), state-based 2D navigation (Figure 2), and image-based Car Racing from the OpenAI Gym (Brockman et al., 2016), with simulated users that label trajectories using a ground-truth reward function. Section A.4 in the appendix discusses implementation details.

MNIST classification. This domain enables us to focus on testing the active learning component of our method, since

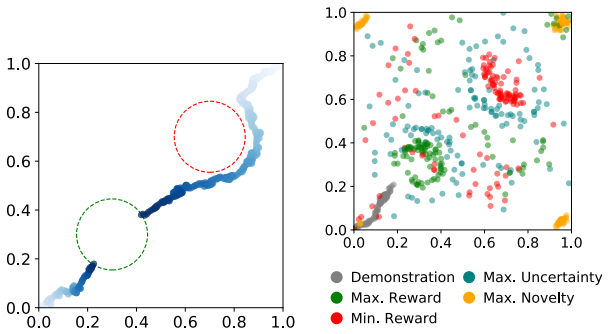

Figure 2. Left: The 2D navigation task, where the agent navigates to the goal region (green) in the lower left while avoiding the trap region (red) in the upper right. The agent starts in the lower left corner in the training environment, and starts in the upper right corner in the test environment. Right: Examples of hypothetical states synthesized throughout learning, illustrating the qualitative differences in the behaviors targeted by each AF.

the standard digit classification task does not involve sequential decision-making. When we generate queries, we synthesize an image $s_0 \in \mathbb{R}^{28\times28}$, and ask the simulated user to label it with an action $a \in \{0, 1, ..., 9\}$. The initial state distribution of the training environment $\mathcal{S}_0^{\text{train}}$ puts a uniform probability on sampling $s_0 \in \{5, 6, 7, 8, 9\}$, and a probability of 0 on sampling $s_0 \in \{0, 1, 2, 3, 4\}$. We intentionally introduce a significant shift in the test environment, by putting a uniform probability on sampling $s_0 \in \{0, 1, 2, 3, 4\}$ and a probability of 0 on sampling $s_0 \in \{5, 6, 7, 8, 9\}$. This mismatch is intended to test how well the learned classifier performs under distribution shift.

State-based 2D navigation. This domain enables us to focus on the challenges of sequential decision-making, without dealing with high-dimensional states. Here, the state $s \in \mathbb{R}^2$ is the agent's position, and the action $a \in \mathbb{R}^2$ is a velocity vector. The task requires navigating to a target region, while avoiding a trap region (Figure 2). The task is harder to complete in the test environment, since the agent starts closer to the trap, and must navigate around the trap to reach the goal.

Image-based Car Racing. This domain enables us to test whether our method scales to learning sequential tasks with high-dimensional states. Here, the state $s \in \mathbb{R}^{64\times64\times3}$ is an RGB image with a top-down view of the car (Figure 3), and the action $a \in \mathbb{R}^3$ controls steering, gas, and brake. Here, we set the same initial state distribution for the training and test environments, since the reward modeling problem is challenging even when the initial state distributions are identical. We train a generative model of images and a forward dynamics model in step (1) using the unsupervised method from Ha & Schmidhuber (2018).

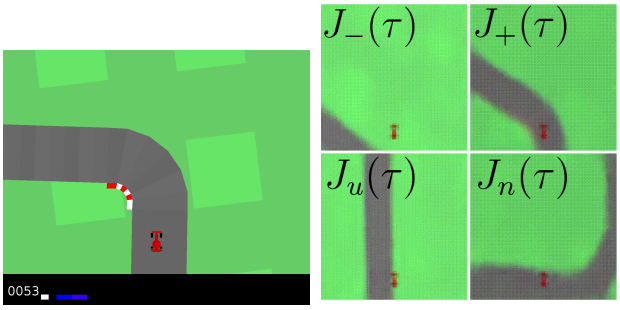

Figure 3. Left: A screenshot of the image-based Car Racing video game in the OpenAI Gym. Right: Examples of synthesized hypothetical behaviors, including going off road (J_-), making progress (J_+), driving on the edge (J_u), and seeking unusual car configurations (J_n). See Figure 13 in the appendix for synthesized videos.

4.1. Robustness Compared to Baselines

Our first experiment tests whether our method can learn a reward model that is robust enough to perform well in the test environment, and tracks how many queries to the user it takes to learn an accurate reward model. To answer **Q1**, we compare our method to a baseline that, instead of generating hypothetical trajectories for the user to label, generates trajectories by rolling out a policy that optimizes the current reward model in the training environment – an approach adapted from prior work (Christiano et al., 2017). The baseline generates τ_{query} in line 5 of Algorithm 1 by rolling out the MPC policy in Equation 4, instead of solving the optimization problem in Equation 3. To test how generating queries using a reward-maximizing policy compares to using a policy that does not depend on the reward model, we also evaluate a simpler baseline that generates query trajectories using a uniform random policy, instead of the MPC policy. We measure performance in MNIST using the agent's classification accuracy in the test environment; in 2D navigation, the agent's success rate at reaching the goal while avoiding the trap in the test environment; and in Car Racing, the agent's true reward, which rewards progress and penalizes going off-road.

The results in Figure 4 show that our method produces reward models that transfer to the test environment better than the baselines. Our method also learns to outperform the suboptimal demonstrations used to initialize the reward model (Figure 10 in the appendix). In MNIST, our method performs substantially better than the baseline, which samples queries s_0 from the initial state distribution of the training environment. The reason is simple: the initial state distribution of the test environment differs significantly from that of the training environment. Since our method is not restricted to sampling from the training environment, it performs better than the baseline.

In 2D navigation, our method substantially outperforms both

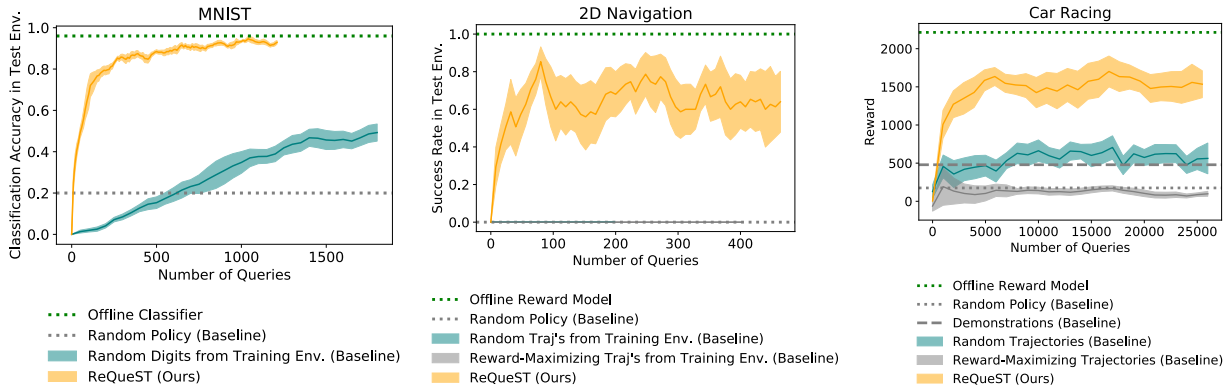

Figure 4. Experiments that address **Q1** – does synthesizing hypothetical trajectories elicit more informative labels than rolling out a policy in the training environment? – by comparing our method, which uses synthetic trajectories, to baselines that only use real trajectories generated in the training environment. The results on MNIST, 2D navigation, and Car Racing show that our method (orange) significantly outperforms the baselines (blue and gray), which never succeed in 2D navigation. The x-axis represents the number of queries to the user, where each query elicits a label for a single state transition (s, a, s'). The shaded areas show standard error over three random seeds.

baselines, which never succeed in the test environment. This is unsurprising, since the training environment is set up in such a way that, because the agent starts out in the lower left corner, they rarely visit the trap region in the upper right by simply taking actions – whether reward-maximizing actions (as in the first baseline), or uniform random actions (as in the second baseline). Hence, when a reward model trained by the baselines is transferred to the test environment, it is not aware of the trap, so the agent tends to get caught in the trap on its way to the goal. Our method, however, is not restricted to feasible trajectories in the training environment, and can potentially query the label for any position in the environment – including the trap (see Figure 2). Hence, our method learns a reward model that is aware of the trap, which enables the agent to navigate around it in the test environment.

In Car Racing, our method outperforms both baselines. The baselines tend to generate queries that are not diverse and rarely visit unsafe states, so the resulting reward models are not able to accurately distinguish between good, unsafe, and neutral states. Our method, on the other hand, explicitly seeks out a wide variety of states by maximizing the four AFs, which leads to more diverse training data, and a more accurate reward model.

4.2. Detecting Reward Hacking

One of the benefits of our method is that it can detect and correct reward hacking before deploying the agent, using reward-maximizing synthetic queries. In the next experiment, we test this claim. We replicate the experimental setup in Section 4.1 for 2D navigation, including the same baselines. We measure performance using the false positive rate of the reward model: the fraction of neutral or unsafe states incorrectly classified as good, evaluated on the offline dataset of trajectories described in Section 4.1. A reward model that outputs false positives is susceptible to reward hacking, since a reward-maximizing agent can game the reward model into emitting high rewards by visiting incorrectly classified states.

The results in Figure 9 in the appendix show that our method drives down the false positive rate in 2D navigation: the learned reward model rarely incorrectly classifies an unsafe or neutral state as a good state. As a result, the deployed agent actually performs the desired task (center plot in Figure 4), instead of seeking false positives. As discussed in Section 4.3 and illustrated in the right-most plot of Figure 5, the baselines learn a reward model that incorrectly extrapolates that continuing up and to the right past the goal region is good behavior. For a concrete example of reward-maximizing synthetic queries that detect reward hacking, consider the reward-maximizing queries in the upper right corner of Figure 2, which are analyzed in Section A.2 in the appendix.

4.3. Safe Exploration

One of the benefits of our method is that it can learn a reward model that accurately detects unsafe states, without having to visit unsafe states during the training process. In the next experiment, we test this claim. We replicate the experimental setup in Section 4.1 for 2D navigation, including the same baselines. We measure performance using the true negative rate of the reward model: the fraction of unsafe states correctly classified as unsafe, evaluated on the offline dataset of trajectories described in Section 4.1. We also use the crash rate of the deployed agent: the rate at which it gets caught in the trap region.

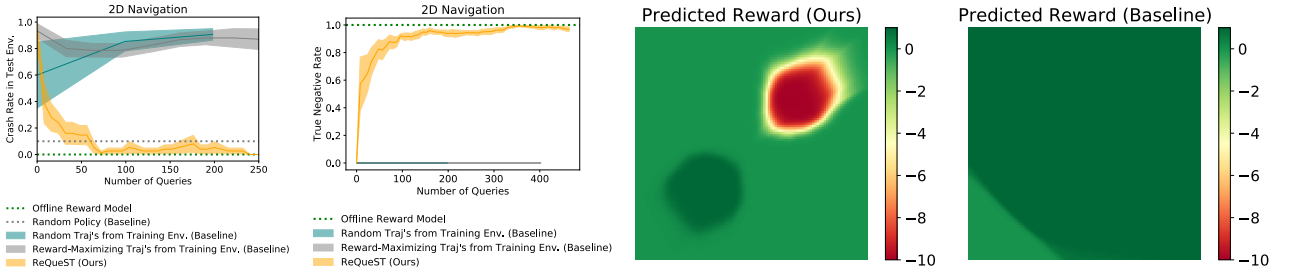

Figure 5. Experiments that address **Q3** – can our method safely learn about unsafe states? – by comparing our method, which uses synthetic trajectories, to baselines that only use real trajectories generated in the training environment. The results on 2D navigation show that our method (orange) significantly outperforms the baselines (blue and gray). The x-axis represents the number of queries to the user, where each query elicits a label for a single state transition (s, a, s'). The shaded areas show standard error over three random seeds. The heat maps represent the reward models learned by our method (left) and by the baselines (right).

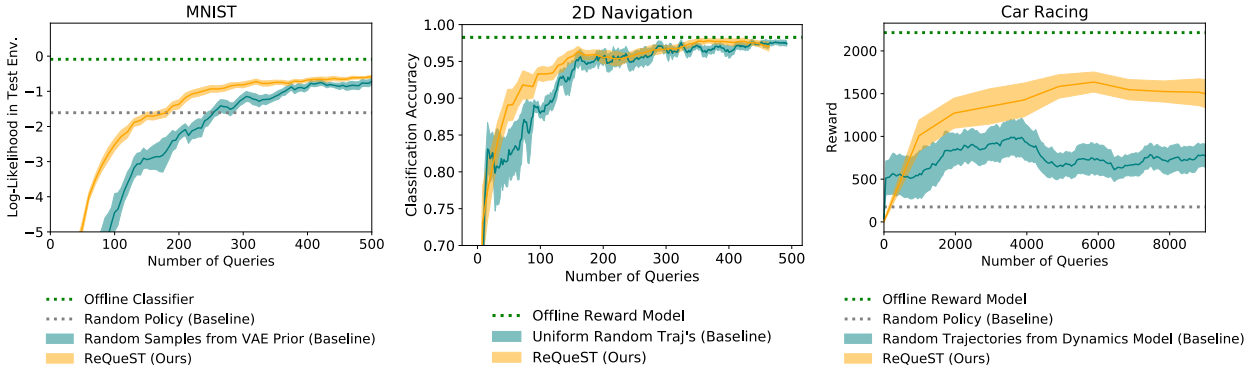

Figure 6. Experiments that address **Q4** – do the proposed AFs improve upon random sampling from the generative model? – by comparing our method, which synthesizes trajectories by optimizing AFs, to a baseline that ignores the AFs and randomly samples from the generative model. The results on MNIST, 2D navigation, and Car Racing show that our method (orange) significantly outperforms the baseline (blue) in Car Racing, and learns faster in MNIST and 2D navigation. The x-axis represents the number of queries to the user, where each query elicits a label for a single state transition (s, a, s'). The shaded areas show standard error over three random seeds.

The results in Figure 5 show that our method learns a reward model that classifies all unsafe states as unsafe, without visiting unsafe states during training (second and third figure from left); in fact, without visiting any states at all, since the queries are synthetic. This enables the agent to avoid crashing during deployment (first figure from left). The baselines differ from our method in that they actually have to visit unsafe states in order to query the user for labels at those states. Since the baselines tend to not visit unsafe states during training, they do not learn about unsafe states (second and fourth figure from left), and the agent frequently crashes during deployment (first figure from left).

4.4. Query Efficiency Compared to Baselines

The previous experiment compared to baselines that are restricted to generating query trajectories by taking actions in the training environment. In this experiment, we lift this restriction on the baselines: instead of taking actions in the training environment, the baselines can make use of the generative model trained in step (1). To answer **Q4**, we compare our method to a baseline that randomly samples trajectories from the generative model p_ϕ – using uniform random actions in Car Racing, samples from the VAE prior in MNIST, and uniform positions across the map in 2D navigation. We measure performance in MNIST using the reward model's predicted log-likelihood of the ground-truth user labels in the test environment; in 2D navigation, the reward model's classification accuracy on an offline dataset containing states sampled uniformly throughout the environment; and in Car Racing, the true reward collected by an MPC agent that optimizes the learned reward, where the true reward gives a bonus for driving onto new patches of road, and penalizes going off-road.

The results in Figure 6 show that our method, which optimizes trajectories using various AFs, requires fewer queries to the user than the baseline, which randomly samples trajectories. This suggests that our four AFs guide query synthesis toward informative trajectories. These results, and the re-

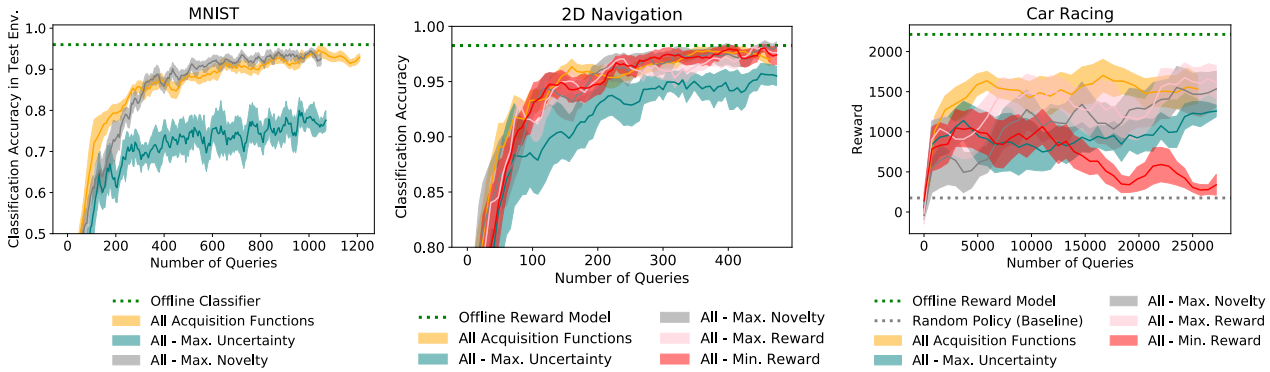

Figure 7. Experiments that address **Q6** – how much do each of the four AFs contribute to performance? – by comparing our method to ablated variants that drop each AF, one at a time, from the set of four AFs in line 4 of Algorithm 1. The results on MNIST, 2D navigation, and Car Racing show that our method (orange) generally outperforms its ablated variants (blue, gray, red, and pink), although the usefulness of each AF depends on the domain and amount of training data.. The x-axis represents the number of queries to the user, where each query elicits a label for a single state transition (s, a, s'). The shaded areas show standard error over three random seeds.

sults from Section 4.1, suggest that our method benefits not only from using a generative model instead of the default training environment, but also from optimizing the AFs instead of randomly sampling from the generative model.

4.5. Acquisition Function Ablation Study

We propose four AFs intended to produce different types of hypotheticals. In this experiment, we investigate the contribution of each type of query to the performance of the overall method. To answer **Q6**, we conduct an ablation study, in which we drop out each the four AFs, one by one, from line 4 in Algorithm 1, and measure the performance of only generating queries using the remaining three AFs.

The results in Figure 7 show that the usefulness of each AF depends on the domain and the amount of training data collected. In MNIST, dropping J_u hurts performance, suggesting that uncertainty-maximizing queries elicit useful labels. Dropping J_n also hurts performance when the number of queries is small, but actually improves performance if enough queries have already been collected. Novelty-maximizing queries tend to be repetitive in practice: although they are distant from the training data in terms of Equation 6, they are visually similar to the existing training data in that they appear to be the same digits. Hence, while they are helpful at first, they hurt query efficiency later in training.

In 2D navigation, dropping J_u hurts performance, while dropping any of the other AFs individually does not hurt performance. These results suggest that uncertainty-maximizing queries can be useful, in domains like MNIST and 2D navigation, where uncertainty can be modeled and estimated accurately. In Car Racing, dropping J_- hurts the most. Reward-minimizing queries elicit labels for unsafe states, which are rare in the training environment unless you explicitly seek them out. Hence, this type of query performs the desired function of augmenting the training data with more examples of unsafe states, thereby making the reward model better at detecting unsafe states.

5. Discussion

The experiments show that ReQueST produces accurate reward models that transfer well to new environments and require fewer queries to the user, compared to baseline methods adapted from prior work. Our method detects reward hacking before the agent is deployed, and safely learns about unsafe states. Through a hyperparameter sweep, we find that our method can trade off between producing realistic vs. informative queries, and that the optimal trade-off varies across domains. Through an ablation study, we find that the usefulness of each of the four acquisition functions we propose for optimizing queries depends on the domain and the amount of training data collected.

One of the benefits of ReQueST is that, since it learns from synthetic trajectories instead of real trajectories, it only has to imagine visiting unsafe states, instead of actually visiting them. Although unsafe states may be visited during unsupervised exploration of the environment for training the generative model in step (1), the same generative model can be reused to learn reward models for any number of future tasks. Hence, the cost of visiting a fixed number of unsafe states in step (1) can be amortized across a large number of tasks in step (2). We could also train the generative model on other off-policy data, including safe expert demonstrations and examples of past failures by humans.

The main practical limitation of ReQueST is that it requires a generative model of initial states and a forward dynamics

model, which can be difficult to learn from purely off-policy data in complex, visual environments. One direction for future work is relaxing this assumption; e.g., by incrementally training a generative model on on-policy data collected from an RL agent in the training environment (Kaiser et al., 2019; Hafner et al., 2019).

Acknowledgments

Thanks to Gabriella Bensinyor, Tim Genewein, Ramana Kumar, Tom McGrath, Victoria Krakovna, Tom Everitt, Zac Kenton, Richard Ngo, Miljan Martic, Adam Gleave, and Eric Langlois for useful suggestions and feedback. Thanks in particular to Gabriella Bensinyor, who proposed the name and acronym of our method: reward query synthesis via trajectory optimization, or ReQueST. This work was supported in part by an NVIDIA Graduate Fellowship.

References

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Betts, J. T. *Practical methods for optimal control and estimation using nonlinear programming*. 2010.

Bıyık, E. and Sadigh, D. Batch active preference-based learning of reward functions. *arXiv preprint arXiv:1810.04303*, 2018.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Brown, D. S., Cui, Y., and Niekum, S. Risk-aware active inverse reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2018.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Neural Information Processing Systems*, 2017.

Cui, Y. and Niekum, S. Active reward learning from critiques. In *International Conference on Robotics and Automation*, 2018.

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Daniel, C., Viering, M., Metz, J., Kroemer, O., and Peters, J. Active reward learning. In *Robotics: Science and Systems*, 2014.

Dvijotham, K. and Todorov, E. Inverse optimal control with linearly-solvable MDPs. In *International Conference on Machine Learning*, 2010.

Foot, P. The problem of abortion and the doctrine of double effect. 1967.

Garcıa, J. and Fernández, F. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 2015.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Neural Information Processing Systems*, 2018.

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. D. Inverse reward design. In *Neural Information Processing Systems*, 2017.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *International Conference on Machine Learning*, 2019.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Neural Information Processing Systems*, 2016.

Huijser, M. and van Gemert, J. C. Active decision boundary annotation with deep generative models. In *IEEE International Conference on Computer Vision*, 2017.

Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., and Amodei, D. Reward learning from human preferences and demonstrations in Atari. In *Neural Information Processing Systems*, 2018.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model-based reinforcement learning for Atari. *arXiv preprint arXiv:1903.00374*, 2019.

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

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems*, 2017.

LeCun, Y. The MNIST database of handwritten digits, 1998. URL http://yann.lecun.com/exdb/mnist/.

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., and Legg, S. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

Lewis, D. D. and Gale, W. A. A sequential algorithm for training text classifiers. In *SIGIR Conference on Research and Development in Information Retrieval*, 1994.

MacGlashan, J., Ho, M. K., Loftin, R., Peng, B., Wang, G., Roberts, D. L., Taylor, M. E., and Littman, M. L. Interactive learning from policy-dependent human feedback. In *International Conference on Machine Learning*, 2017.

Mindermann, S., Shah, R., Gleave, A., and Hadfield-Menell, D. Active inverse reward design. *arXiv preprint arXiv:1809.03060*, 2018.

Ng, A. Y. and Russell, S. J. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

Olah, C., Mordvintsev, A., and Schubert, L. Feature visualization. *Distill*, 2017.

Pomerleau, D. A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 1991.

Prakash, B., Khatwani, M., Waytowich, N., and Mohsenin, T. Improving safety in reinforcement learning using model-based architectures and human intervention. In *Florida Artificial Intelligence Research Society Conference*, 2019.

Premack, D. and Woodruff, G. Does the chimpanzee have a theory of mind? *Behavioral and Brain Sciences*, 1978.

Reddy, S., Dragan, A. D., and Levine, S. Shared autonomy via deep reinforcement learning. In *Robotics: Science and Systems*, 2018.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Sadigh, D., Dragan, A. D., Sastry, S., and Seshia, S. A. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017.

Saunders, W., Sastry, G., Stuhlmueller, A., and Evans, O. Trial without error: Towards safe reinforcement learning via human intervention. In *International Conference on Autonomous Agents and MultiAgent Systems*, 2018.

Savage, L. J. *The foundations of statistics*. John Wiley & Sons, 1954.

Sener, O. and Savarese, S. Active learning for convolutional neural networks: A core-set approach. *International Conference on Learning Representations*, 2018.

Settles, B. Active learning literature survey. Technical report, 2009.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2018.

Warnell, G., Waytowich, N., Lawhern, V., and Stone, P. Deep TAMER: Interactive agent shaping in high-dimensional state spaces. In *AAAI Conference on Artificial Intelligence*, 2018.

Williams, G., Drews, P., Goldfain, B., Rehg, J. M., and Theodorou, E. A. Aggressive driving with model predictive path integral control. In *International Conference on Robotics and Automation*, 2016.

Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J. M., Boots, B., and Theodorou, E. A. Information theoretic MPC for model-based reinforcement learning. In *International Conference on Robotics and Automation*, 2017.

Wirth, C., Akrour, R., Neumann, G., and Fürnkranz, J. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 2017.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008.