A. Appendix

A.1. Designing Objectives for Informative Queries

Maximizing uncertainty. 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,

$$J_u(\tau) = \frac{1}{|\tau|} \sum_{(s,a,s') \in \tau} \frac{1}{m} \sum_{i=1}^{m} D_{\text{KL}}(p_{\boldsymbol{\theta}_i}(c|s,a,s') \parallel p_{\boldsymbol{\theta}}(c|s,a,s')), \quad (5)$$

where $p_{\boldsymbol{\theta}}$ is the reward classifier defined in Section 3.2.

Maximizing novelty. In this work, we use a distance function that computes the Euclidean distance between state embeddings,

$$d(\tau, \tau') = \frac{1}{|\tau||\tau'|} \sum_{s \in \tau, s' \in \tau'} -e^{-\|f_{\boldsymbol{\phi}}(s_t) - f_{\boldsymbol{\phi}}(s'_t)\|_2}, \quad (6)$$

where $f_{\boldsymbol{\phi}}$ is the state encoder trained in step (1).

A.2. Visualizing Synthesized Queries

Figures 2, 12, and 13 illustrate examples of queries generated by each of the four AFs.

In MNIST (Figure 12), the uncertainty-maximizing queries are digits that appear ambiguous but coherent, while the novelty-maximizing queries tend to cluster around a small subset of the digits and appear grainy.

In 2D navigation (Figure 2), the demonstrations contain mostly neutral states en route to the goal, and a few good states at the goal. If we were to train on only the demonstrations, the reward model would be unaware of the trap. Initially, the queries, which we restrict to just one state transition from the initial state s_0 to a synthesized next state s_1, are relatively uniform. The first reward-maximizing queries are in the upper right corner, which makes sense: the demonstrations contain neutral states in the lower left, and good states farther up and to the right inside the goal region, so the reward model extrapolates that continuing up and to the right, past the goal region, is good behavior. The reward model, at this stage, is susceptible to reward hacking – a problem that gets addressed when the user labels the reward-maximizing queries in the upper right corner as neutral.

After a few more queries, the reward-maximizing queries start to cluster inside the goal region, and the reward-minimizing queries cluster inside the trap. This is helpful early during training, for identifying the locations of the goal and trap. The uncertainty-maximizing queries cluster around the boundaries of the goal and the trap, since that is where model uncertainty is highest. This is helpful for refining the reward model's knowledge of the shapes of the goal and trap. The novelty-maximizing queries get pushed to the corners of the environment. This is helpful for determining that the goal and trap are relatively small and circular, and do not bleed into the corners of map.

In Car Racing (Figure 13), the reward-maximizing queries show the car driving down the road and making a turn. The reward-minimizing queries show the car going off-road as quickly as possible. The uncertainty-maximizing queries show the car driving to the edge of the road and slowing down. The novelty-maximizing queries show the car staying still, which makes sense since the training data tends to contain mostly trajectories of the car in motion.

A.3. Effect of Regularization Constant λ

One of the core features of our method is that, in Equation 3, it can trade off between producing realistic queries that maximize the regularization term $\log p_{\boldsymbol{\phi}}(\tau)$, and producing informative queries that maximize the AF $J(\tau)$. In this experiment, we examine how the regularization constant λ controls this trade-off, and how the trade-off affects performance.

To answer **Q5**, we sweep different values of the regularization constant λ. At one extreme, we constrain the query trajectories τ_{query} to be feasible under the generative model, by setting the next states z_{t+1} to be the next states predicted by the dynamics model instead of free variables – we label this setting as $\lambda = \infty$ for convenience (see Section A.4 in the appendix for details). At the other extreme, we set $\lambda = 0$, which allows τ_{query} to be infeasible under the model. Note that, even when $\lambda = 0$, the optimized trajectory τ_{query} is still regularized by the fact that it is optimized in the latent space of the state encoder f, instead of, e.g., raw pixel space.

The results in Figure 8 show that the usefulness of generating unrealistic trajectories depends on the domain. In MNIST, producing unrealistic images by decreasing λ can improve performance, although an intermediate value works best. In 2D navigation, setting λ to a low value is critical for learning the task. Note that we only tested $\lambda = 0$ and $\lambda = \infty$ in this domain, since we intentionally setup the training and test environments as a sanity check, where $\lambda = 0$ should perform best, and $\lambda = \infty$ should not succeed. In Car Racing, constraining the queries to be feasible ($\lambda = \infty$) performs best.

There is a trade-off between being informative (by maximizing the AF) and staying on the distribution of states in the training environment (by maximizing likelihood). In domains like Car Racing – where the training and test environments have similar state distributions, and off-distribution

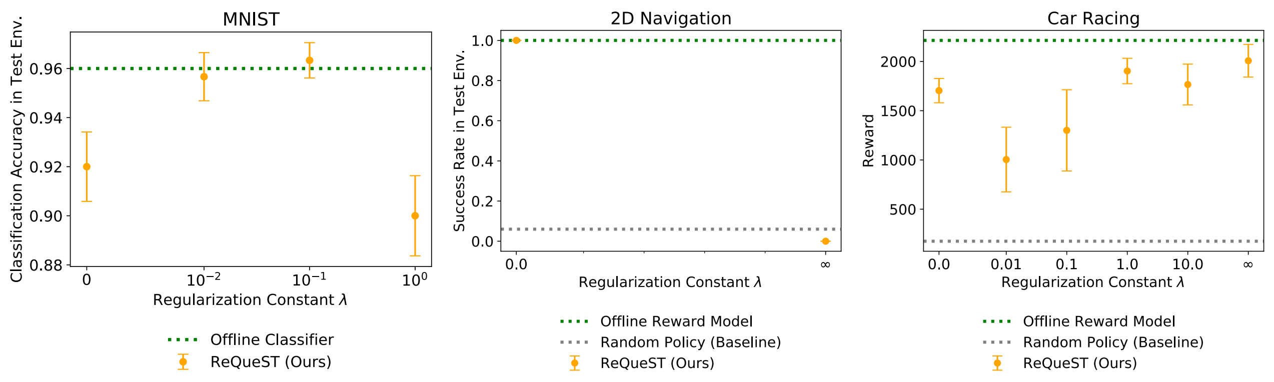

Figure 8. Experiments that address **Q5** – how does the regularization constant λ control the trade-off between realistic and informative queries? – by evaluating our method with different values of λ, which controls the trade-off between producing realistic trajectories (higher λ) and informative trajectories (lower λ). The results on MNIST, 2D navigation, and Car Racing show that, while intermediate and low values of λ work best for MNIST and 2D navigation respectively, a high value of $\lambda = \infty$ works best for Car Racing. The x-axis is log-scaled. The error bars show standard error over three random seeds, which is negligible in the results for 2D navigation.

queries can be difficult for the user to interpret and label – it makes sense to trade off being informative for staying on-distribution. In domains like MNIST and 2D navigation, where we intentionally create a significant shift in the state distribution between the training and test environments, it makes more sense to trade off staying on-distribution for being informative.

Figure 13 shows examples of Car Racing query trajectories τ_{query} optimized with either $\lambda = 0$ or $\lambda = \infty$. Unsurprisingly, the $\lambda = 0$ queries appear less realistic, but clearly maximize the AF better than their $\lambda = \infty$ counterparts.

A.4. Implementation Details

Initializing with demonstrations. In many real-world settings, the user can help initialize the reward model by providing a small number of (suboptimal) demonstrations and labeling them with rewards. Hence, we initialize the training data $\mathcal{D}$ in line 2 of Algorithm 1 with a small set of labeled, suboptimal, user demonstrations collected in the training environment.

Model predictive control. Although more sophisticated MPC algorithms exist (Williams et al., 2016; 2017), we find that the simple gradient descent-based planner described in Section 3.5 works well in the 2D navigation and Car Racing domains.

Query synthesis. Our approach to query synthesis in Section 3.4 draws inspiration from direct collocation methods in the trajectory optimization literature (Betts, 2010), feature visualization methods in the neural network interpretability literature (Olah et al., 2017), and prior work on active learning with deep generative models (Huijser & van Gemert, 2017). It can be extended to settings where $p_{\boldsymbol{\phi}}(\tau)$ is not differentiable, by using a gradient-free optimization method to synthesize τ_{query}. This can be helpful, e.g., when using a non-differentiable simulator to model the environment.

Ensemble of reward models. In line 12 of Algorithm 1, we train each ensemble member on all of the data $\mathcal{D}$, instead of a random subset of the data (i.e., bootstrapping). As in Lakshminarayanan et al. (2017), we find that simply training each reward network $\boldsymbol{\theta}_i$ using a different random seed works well in practice for modeling uncertainty.

Shooting vs. collocation. We use the notation $\lambda = \infty$ to denote solving the optimization problem in Equation 3 with a shooting method instead of a collocation method. The shooting method optimizes $(z_0, a_0, a_1, ..., a_{T-1})$, and represents $z_{t+1} = \mathbb{E}_{\boldsymbol{\phi}}[z_{t+1}|z_0, a_0, a_1, ..., a_t]$ using the forward dynamics model $p_{\boldsymbol{\phi}}$ learned in step (1).

MNIST classification. We train a state encoder $f_{\boldsymbol{\phi}}$ and decoder $f_{\boldsymbol{\phi}}^{-1}$ in step (1) by training a VAE with an 8-dimensional latent space $Z = \mathbb{R}^8$ on all the images in the MNIST training set. We simulate the user in line 7 of Algorithm 1 as an expert, k-nearest neighbors classifier $p_{\text{user}}(a|s)$ trained on all labeled data. We only generate queries using the AFs J_n and J_u in line 4 of Algorithm 1, since J_+ and J_- are not useful for single-step classification. We replace $p_{\boldsymbol{\theta}_i}(c|s, a, s')$ with $p_{\boldsymbol{\theta}_i}(a|s)$ in Equation 5 and line 12 of Algorithm 1. We represent $p_{\boldsymbol{\theta}}(a|s)$ in Equation 2 using a feedforward neural network with two fully-connected hidden layers containing 256 hidden units each, and $m = 4$ separate networks in the ensemble. The MPC agent in Equation 4 reduces to $\pi_{\text{mpc}}(a|s) = p_{\boldsymbol{\theta}}(a|s)$. The Gaussian prior distribution of the VAE yields the likelihood model, $p_{\boldsymbol{\phi}}(s_0) \propto \exp\left(\|f_{\boldsymbol{\phi}}(s_0)\|_2^2\right)$. The state inputs to the reward model are the latent embeddings produced by $f_{\boldsymbol{\phi}}$, instead of the raw pixel inputs. We set $\lambda = 0.1$ when synthesizing queries with the AF J_u, and $\lambda = 0.01$ when synthesizing queries with the AF J_n. We establish a lower bound on performance using a uniform random policy, and an upper bound by deploying an MPC agent equipped with a reward model trained on a large, offline dataset of 100 expert trajectories and 100 random trajectories containing balanced classes of good, unsafe, and neutral state transitions.

Note that our approach of training a VAE generative model on all of the MNIST training data differs from the random sampling method for collecting off-policy data described in Section 3.1. Though the initial state distribution of the training environment is a uniform distribution over $\{5, 6, 7, 8, 9\}$, we train the generative model on all the digits $\{0, 1, 2, ..., 9\}$. This simplifies our experiments, and enables ReQueST to synthesize hypothetical digits from $\{0, 1, 2, 3, 4\}$.

State-based 2D navigation. The simulated user labels a state transition (s, a, s') with category $c \in \{\text{good}, \text{unsafe}, \text{neutral}\}$, by looking at the state s', and identifying whether it is inside the goal region (good), inside the trap region (unsafe), or outside both regions (neutral). To encourage the agent to avoid the trap, the reward constants are asymmetric: $R_{\text{good}} = 1$, $R_{\text{unsafe}} = -10$, and $R_{\text{neutral}} = 0$. Since the states are already low-dimensional, we simply use the identity function for the state encoder and decoder. We represent $p_{\boldsymbol{\theta}}(c|s, a, s')$ in Equation 2 using a feedforward neural network with two fully-connected hidden layers containing 32 hidden units each, and $m = 4$ separate networks in the ensemble. We hard-code a Gaussian forward dynamics model, $p(s_{t+1}|s_t, a_t) = \mathcal{N}(s_{t+1}; s_t + a_t, \sigma^2)$. Each episode lasts at most 1000 steps, and the maximum speed is restricted to $\|a\|_2 \leq 0.01$. In Equation 4, we use a planning horizon of $H = 500$. In Equation 3, we use a query trajectory length of $T = 1$; i.e., the query consists of one state transition from the hard-coded initial state s_0 to a synthesized next state s_1. We set $\lambda = 0$ when synthesizing queries for any of the four AFs.

Car Racing. The simulated user labels a state transition (s, a, s') with category $c \in \{\text{good}, \text{unsafe}, \text{neutral}\}$, by looking at the state s', and identifying whether it shows the car driving onto a new patch of road (good), off-road (unsafe), or in a previously-visited road patch (neutral). To encourage the agent to drive without being overly conservative, the reward constants are asymmetric: $R_{\text{good}} = 10$, $R_{\text{unsafe}} = -1$, and $R_{\text{neutral}} = 0$. We represent $p_{\boldsymbol{\theta}}(c|s, a, s')$ in Equation 2 using a feedforward neural network with two fully-connected hidden layers containing 256 hidden units each, and $m = 4$ separate networks in the ensemble. We train a generative model using the unsupervised approach in Ha & Schmidhuber (2018), which learns a VAE state encoder and decoder with a 32-dimensional latent space, a recurrent dynamics model with a 256-dimensional latent space, and a mixture density network with 5 components that predicts stochastic transitions. Since the environment

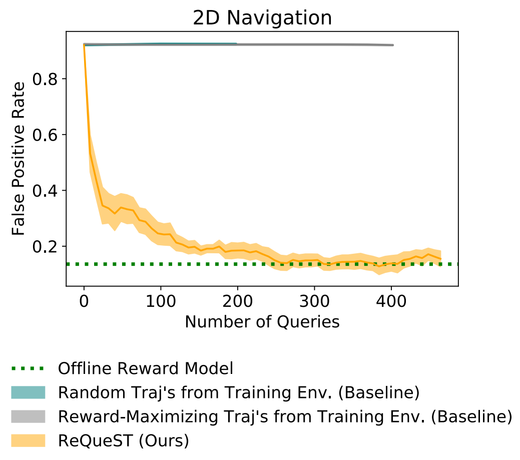

Figure 9. Experiments that address **Q2** – can our method detect and correct reward hacking? – by comparing our method, which uses synthetic trajectories, to baselines that only use real trajectories generated in the training environment. The results on 2D navigation show that our method (orange) significantly 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.

is partially observable, we represent the state input to the reward model by concatenating the VAE latent embedding with the RNN latent embedding. Each episode lasts at most 1000 timesteps. In Equation 4, we use a planning horizon of $H = 50$. In Equation 3, we use a query trajectory length of $T = 50$. We set $\lambda = \infty$ when synthesizing queries for any of the four AFs.

In the high-dimensional Car Racing environment, we find that optimizing Equation 3 leads to incomprehensible query trajectories τ_{query}, even for high values of the regularization constant λ. To address this issue, we modify the method in two ways that provide additional regularization. First, instead of optimizing the initial state s_0 in τ_{query}, we set it to some real state sampled from the training environment during step (1). Second, instead of optimizing $(z_0, a_0, z_1, ..., z_T)$, where $\tau = (f^{-1}(z_0), a_0, f^{-1}(z_1), a_1, ..., f^{-1}(z_T))$, we optimize $(z_0, a_0, m_0, a_1, m_1, ..., a_{T-1}, m_{T-1})$, where $\tau = (f^{-1}(z_0), a_0, f^{-1}(\text{MDN}(z_0, a_0, m_0)), a_1, ..., f^{-1}(\text{MDN}(z_{T-1}, a_{T-1}, m_{T-1})))$. The function $\text{MDN}(z_t, a_t, m_t)$ denotes using the mixture coefficients m_t to compute the expected next state, instead of using the mixture coefficients $\psi(z_t, a_t)$ predicted by the mixture density network. Thus, the likelihood regularization term becomes $\log p_{\boldsymbol{\phi}}(\tau) = \sum_{t=0}^{T-1} H(m_t, \psi(z_t, a_t))$, where H is the cross-entropy. This representation of the trajectory τ is easier to optimize, and results in more comprehensible queries.

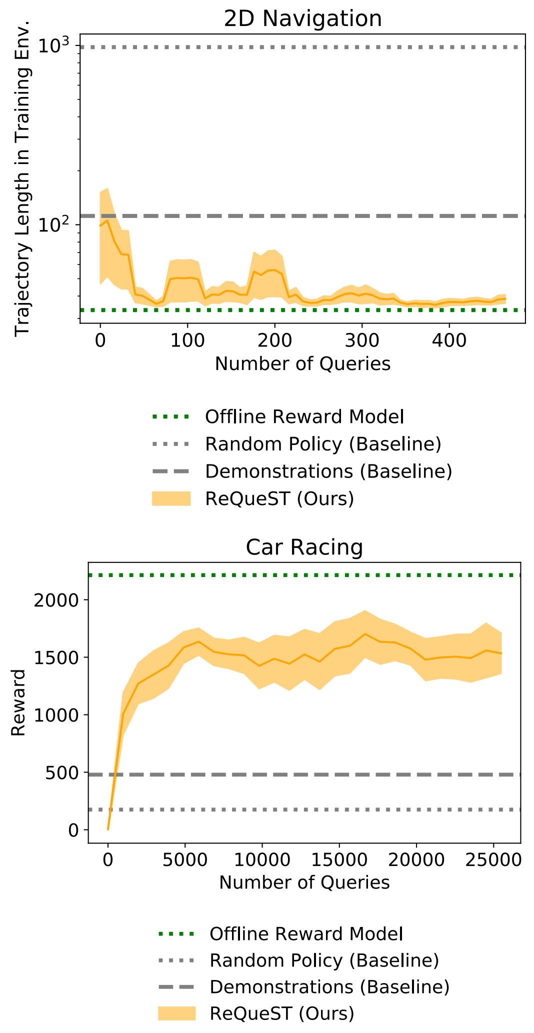

Figure 10. Our method initializes the reward model with suboptimal user demonstrations, in line 2 of Algorithm 1. The experiments in Section 4.1 show that our method learns a reward model that enables the agent to outperform the suboptimal demonstrator. In 2D navigation (top), the agent gets to the goal faster than the demonstrator, even in the training environment – the demonstrator takes a tortuous path to the goal, while the agent goes straight to the goal. In Car Racing (bottom), the agent drives faster and visits more new road patches than the cautious, slow demonstrator. We do not include results for MNIST, since it does not make sense to initialize the classifier with incorrect labels in this domain.

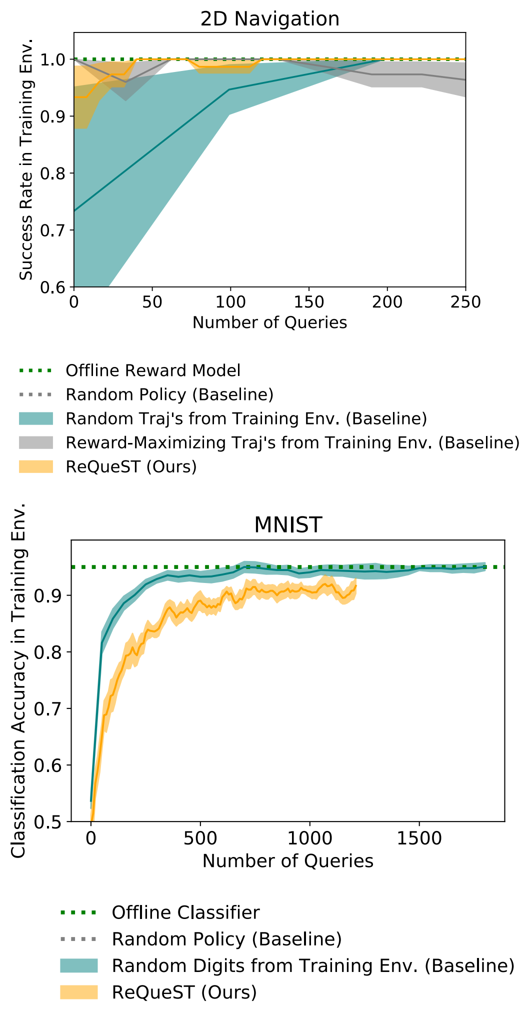

Figure 11. Our method performs worse than or comparably to the baselines in Section 4.1, when the reward model is evaluated in the training environment instead of the test environment. Since there is no state distribution shift in this setting, training on real trajectories from the training environment (baselines) is more effective than training on hypothetical trajectories synthesized using our method (ReQueST). We do not include results for Car Racing, since the test environment is already identical to the training environment in this domain.

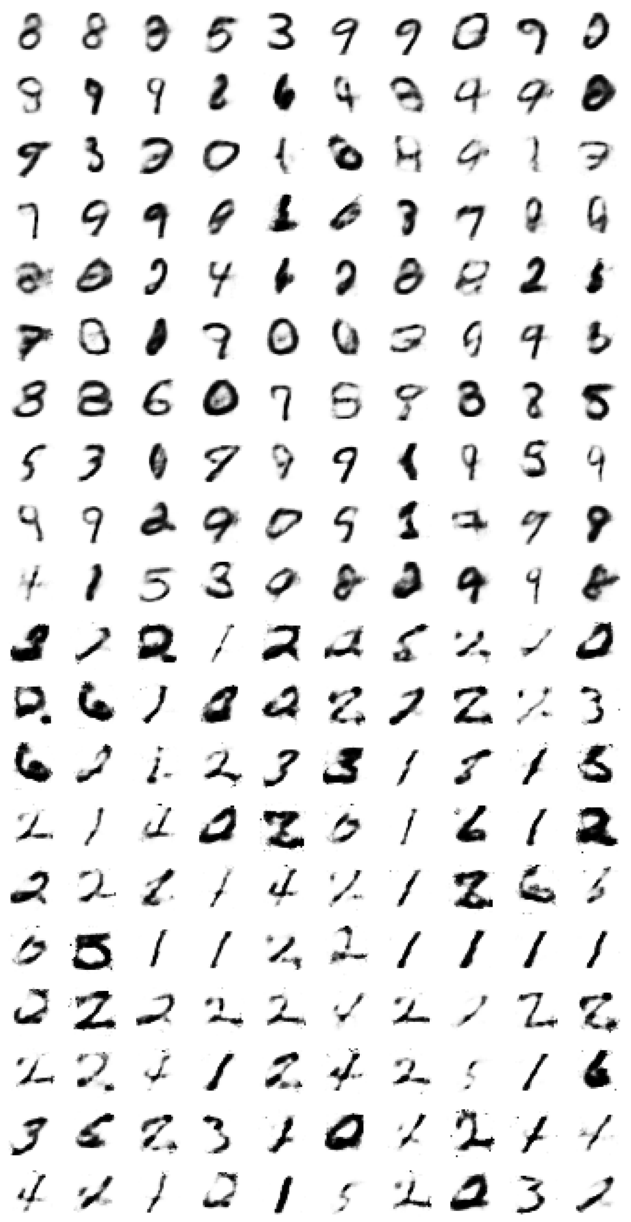

Figure 12. Examples of MNIST queries that optimize different AFs, illustrating the qualitative differences in the hypotheticals targeted by each AF. Top 10 rows: uncertainty-maximizing queries. Bottom 10 rows: novelty-maximizing queries. The uncertainty-maximizing queries are digits that appear ambiguous but coherent, while the novelty-maximizing queries tend to cluster around a small subset of the digits and appear grainy.

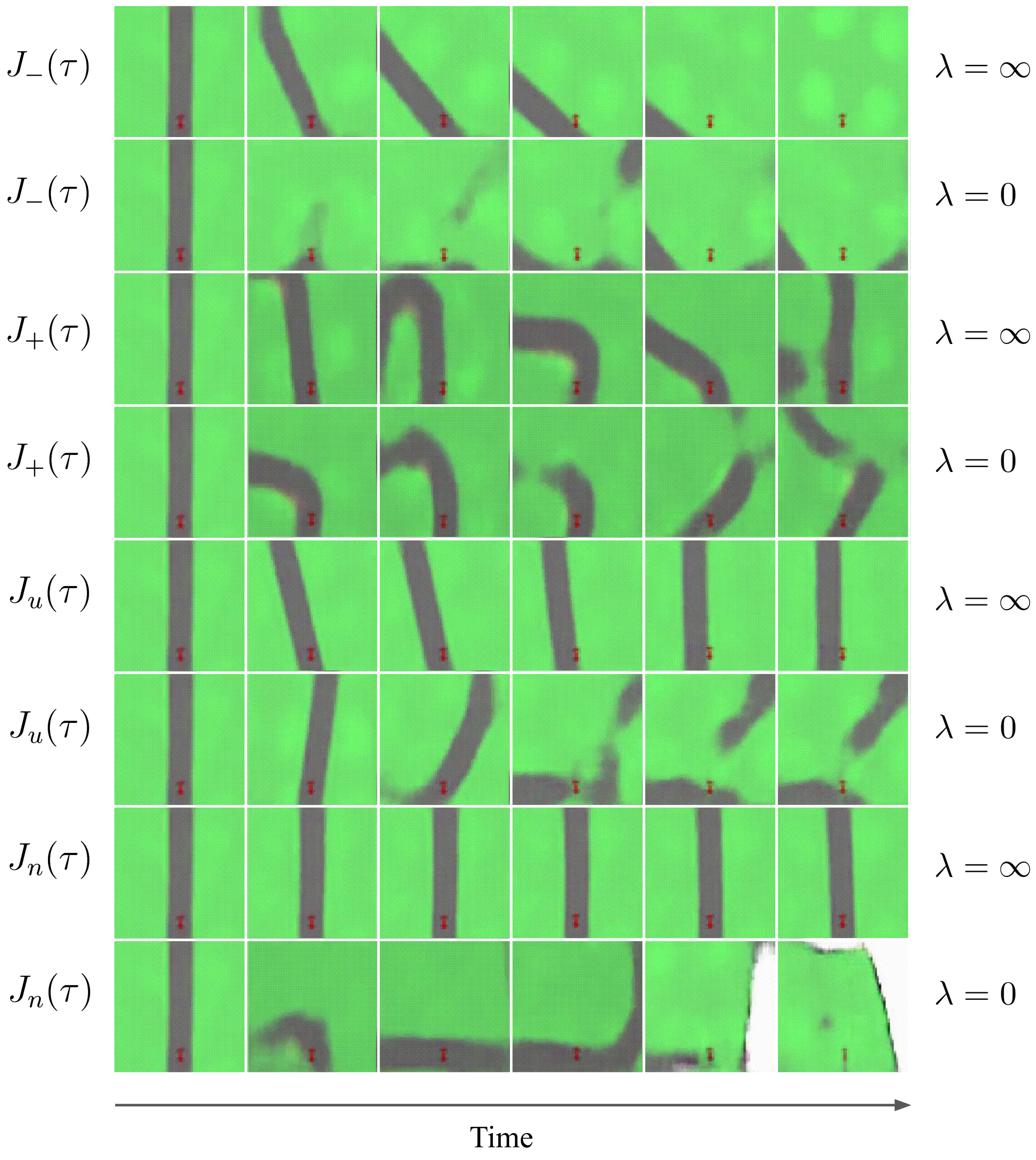

Figure 13. Examples of Car Racing queries that optimize different AFs with different settings of the regularization constant λ, illustrating the qualitative differences in the hypotheticals targeted by each AF, and the trade-off between producing realistic ($\lambda = \infty$) vs. informative ($\lambda = 0$) queries. When $\lambda = \infty$, the reward-maximizing query shows the car driving down the road and making a turn; the reward-minimizing query shows the car going off-road as quickly as possible; the uncertainty-maximizing query shows the car driving to the edge of the road and slowing down; and the novelty-maximizing query shows the car staying still, which makes sense since the training data tends to contain mostly trajectories of the car in motion. When $\lambda = 0$, most of the behaviors are qualitatively similar to their $\lambda = \infty$ counterparts, but less realistic and more aggressively optimizing the AF – only the novelty-maximizing query is qualitatively different, in that it seeks the boundaries of the map (the white void) instead of staying still. Full videos available in the supplementary material.