APPENDIX
Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences

Daniel S. Brown [1] **Russell Coleman** [1 2] **Ravi Srinivasan** [2] **Scott Niekum** [1]

A. Source Code and Videos

See the project webpage https://sites.google.com/view/bayesianrex/.

B. MCMC Details

We represent R_θ as a linear combination of pre-trained features:

$$R_\theta(\tau) = \sum_{s\in\tau} w^T\phi(s) = w^T \sum_{s\in\tau}\phi(s) = w^T\Phi_\tau. \quad (1)$$

We pre-compute and cache $\Phi_{\tau_i} = \sum_{s\in\tau_i}\phi(s)$ for $i = 1,\ldots,m$ and the likelihood becomes

$$P(\mathcal{P}, D \mid R_\theta) = \prod_{(i,j)\in\mathcal{P}} \frac{e^{\beta w^T\Phi_{\tau_j}}}{e^{\beta w^T\Phi_{\tau_j}} + e^{\beta w^T\Phi_{\tau_i}}}. \quad (2)$$

We use $\beta = 1$ and enforce constraints on the weight vectors by normalizing the output of the weight vector proposal such that $\|w\|_2 = 1$ and use a Gaussian proposal function centered on w with standard deviation σ. Thus, given the current sample w_t, the proposal is defined as $w_{t+1} = \texttt{normalize}(\mathcal{N}(w_t, \sigma))$, in which `normalize` divides by the L2 norm of the sample to project back to the surface of the L2-unit ball.

For all experiments, except Seaquest, we used a default step size of 0.005. For Seaquest increased the step size to 0.05. We run 200,000 steps of MCMC and use a burn-in of 5000 and skip every 20th sample to reduce auto-correlation. We initialize the MCMC chain with a randomly chosen vector on the L2-unit ball. Because the inverse reinforcement learning is ill-posed there are an infinite number of reward functions that could match any set of demonstrations. Prior work by Finn et al. (2016) demonstrates that strong regularization is needed when learning cost functions via deep neural networks. To ensure that the rewards learned allow good policy optimization when fed into an RL algorithm we used a non-negative return prior on the return of the lowest ranked demonstration. The prior takes the following form:

$$\log P(w) = \begin{cases} 0 & \text{if } e^{\beta w^T\Phi_{\tau_1}} < 0 \\ -\infty & \text{otherwise} \end{cases} \quad (3)$$

This forces MCMC to not only find reward function weights that match the rankings, but to also find weights such that the return of the worse demonstration is non-negative. If the return of the worse demonstration was negative during proposal generation, then we assigned it a prior probability of $-\infty$. Because the ranking likelihood is invariant to affine transformations of the rewards, this prior simply shifts the range of learned returns and does not affect the log likelihood ratios.

C. Bayesian IRL vs. Bayesian REX

Bayesian IRL does not scale to high-dimensional tasks due to the requirement of repeatedly solving for an MDP in the inner loop. In this section we focus on low-dimensional problems where it is tractable to repeatedly solve an MDP. We compare the performance of Bayesian IRL with Bayesian REX when performing reward inference. Because both algorithms make very different assumptions, we compare their performance across three different tasks. The first task attempts to give both algorithms the demonstrations they were designed for. The second evaluation focuses on the case where all demonstrations are optimal and is designed to put Bayesian IRL at a disadvantage. The third evaluation focuses on the case where all demonstrations are optimal and is designed to put Bayesian REX at a disadvantage. Note that we focus on sample efficiency rather than computational efficiency as Bayesian IRL is significantly slower than Bayesian REX as it requires repeatedly solving an MDP, whereas Bayesian REX requires no access to an MDP during reward inference.

All experiments were performed using 6x6 gridworlds with 4 binary features placed randomly in the environment. The ground-truth reward functions are sampled uniformly from the L1-ball (Brown & Niekum, 2018). The agent can move in the four cardinal directions and stays in place if it attempts

[1]Computer Science Department, The University of Texas at Austin. [2]Applied Research Laboratories, The University of Texas at Austin. Correspondence to: Daniel Brown <dsbrown@cs.utexas.edu>.

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

Algorithm 1 Bayesian REX: Bayesian Reward Extrapolation

Input: demonstrations D, pairwise preferences $\mathcal{P}$, MCMC proposal width σ, number of proposals to generate N, deep network architecture R_θ, and prior $P(w)$.
pre-train R_θ using auxiliary tasks (see Section 5.2).
Freeze all but last layer, w, of R_θ.
$\phi(s) :=$ activations of the penultimate layer of R_θ.
Precompute and cache $\Phi_\tau = \sum_{s \in \tau} \phi(s)$ for all $\tau \in D$.
Initialize w randomly.
Chain[0] $\leftarrow w$
Compute $P(\mathcal{P}, D|w)P(w)$ using Equation (2)
for $i \leftarrow 1$ to N **do**
  $\tilde{w} \leftarrow \texttt{normalize}(\mathcal{N}(w_t, \sigma))$
  Compute $P(\mathcal{P}, D|\tilde{w})P(\tilde{w})$ using Equation (2)
  $u \leftarrow \texttt{Uniform}(0, 1)$
  if $u < \dfrac{P(\mathcal{P}, D|\tilde{w})P(\tilde{w})}{P(\mathcal{P}, D|w)P(w)}$ **then**
    Chain[i] $\leftarrow \tilde{w}$
    $w \leftarrow \tilde{w}$
  else
    Chain[i] $\leftarrow w$
  end if
end for
return Chain

to move off the grid. Transitions are deterministic, $\gamma = 0.9$, and there are no terminal states. We perform evaluations over 100 random gridworlds for varying numbers of demonstrations. Each demonstration is truncated to a horizon of 20. We use $\beta = 50$ for both Bayesian IRL and Bayesian REX and we remove duplicates from demonstrations. After performing MCMC we used a 10% burn-in period for both algorithms and only used every 5th sample after the burn-in when computing the mean reward under the posterior. We then optimized a policy under the mean reward from the Bayesian IRL posterior and under the mean reward from the Bayesian REX posterior. We then compare the average policy loss for each algorithm when compared with optimal performance under the ground-truth reward function.

C.1. Ranked Suboptimal vs. Optimal Demonstrations

We first compare Bayesian IRL when it is given optimal demonstrations with Bayesian REX when it receives suboptimal demonstrations. We give each algorithm the demonstrations best suited for its assumptions while keeping the number of demonstrations equal and using the same starting states for each algorithm. To generate suboptimal demonstrations we simply use random rollouts and then rank them according to the ground-truth reward function.

Table 1 shows that, given a sufficient number of suboptimal ranked demonstrations (> 5), Bayesian REX performs on par or slightly better than Bayesian IRL when given the same number of optimal demonstrations starting from the same states as the suboptimal demonstrations. This result shows that not only is Bayesian REX much more computationally efficient, but it also has sample efficiency comparable to Bayesian IRL as long as there are a sufficient number of ranked demonstrations. Note that 2 ranked demonstrations induces only a single constraint on the reward function so it is not surprising that it performs much worse than running full Bayesian IRL with all the counterfactuals afforded by running an MDP solver in the inner-loop.

Table 1. Ranked Suboptimal vs. Optimal Demos: Average policy loss over 100 random 6x6 grid worlds with 4 binary features.

	2	5	10	20	30
B-IRL	**0.044**	**0.033**	0.020	0.009	**0.006**
B-REX	1.779	0.421	**0.019**	**0.006**	**0.006**

Table 2. Ranked Suboptimal Demos: Average policy loss for Bayesian IRL versus Bayesian REX over 100 random 6x6 grid worlds with 4 binary features.

	2	5	10	20	30
B-IRL	3.512	3.319	2.791	3.078	3.158
B-REX	**1.796**	**0.393**	**0.026**	**0.006**	**0.006**

C.2. Only Ranked Suboptimal Demonstrations

For the next experiment we consider what happens when Bayesian IRL recieves suboptimal ranked demonstrations. Table 2 shows that B-REX always significantly outperforms Bayesian IRL when both algorithms receive suboptimal ranked demonstrations. To achieve a fairer comparison, we also compared Bayesian REX with a Bayesian IRL algorithm designed to learn from both good and bad demonstrations (Cui & Niekum, 2018). We labeled the top $x\%$ ranked demonstrations as good and bottom $x\%$ ranked as bad. Table 3 shows that leveraging the ranking significantly improves the performance of Bayesian IRL, but Bayesian REX still performed significantly better across all x.

C.3. Only Optimal Demonstrations

Finally, we compared Bayesian REX with Bayesian IRL when both algorithms are given optimal demonstrations. As an attempt to use Bayesian REX with only optimal demonstrations, we followed prior work (Brown et al., 2019a) and auto-generated pairwise preferences using uniform random rollouts that are labeled as less preferred than the demonstrations. Table 4 shows that Bayesian IRL outperforms Bayesian REX. This demonstrates the value of giving a variety of ranked trajectories to Bayesian REX. For small

Table 3. Ranked Suboptimal Demos: Average policy loss for Bayesian REX and Bayesian IRL using the method proposed by (Cui & Niekum, 2018)* which makes use of good and bad demonstrations. We used the top $x\%$ of the ranked demos as good and bottom $x\%$ as bad. Results are averaged over 100 random 6x6 grid worlds with 4 binary features.

	Top/bottom percent of 20 ranked demos			
	x=5%	x=10%	x=25%	x=50%
B-IRL(x)*	1.283	0.956	1.065	2.096
B-REX			**0.006**	

Table 4. Ranked Suboptimal Demos: Average policy loss for Bayesian IRL versus Bayesian REX over 100 random 6x6 grid worlds with 4 binary features.

	2	5	10	20	30
B-IRL	**0.045**	**0.034**	**0.018**	**0.009**	**0.006**
B-REX	0.051	0.045	0.040	0.034	0.034

numbers of optimal demonstrations (≤ 5) we found that Bayesian REX leveraged the self-supervised rankings to only perform slightly worse than full Bayesian IRL. This result is encouraging since it is possible that a more sophisticated method for auto-generating suboptimal demonstrations and rankings could be used to further improve the performance of Bayesian REX even when demonstrations are not ranked (Brown et al., 2019a).

C.4. Summary

The results above demonstrate that if a very small number of unlabeled near-optimal demonstrations are available, then classical Bayesian IRL is the natural choice for performing reward inference. However, if any of these assumptions are not true, then Bayesian REX is a competitive and often superior alternative for performing Bayesian reward inference. Also implicit in the above results is the assumption that a highly tractable MDP solver is available for performing Bayesian IRL. If this is not the case, then Bayesian IRL is infeasible and Bayesian REX is the natural choice for Bayesian reward inference.

D. Pre-training Latent Reward Features

We experimented with several pretraining methods. One method is to train R_θ using the pairwise ranking loss

$$P(D, \mathcal{P} \mid R_\theta) = \prod_{(i,j)\in\mathcal{P}} \frac{e^{\beta R_\theta(\tau_j)}}{e^{\beta R_\theta(\tau_i)} + e^{\beta R_\theta(\tau_j)}}, \quad (4)$$

and then freeze all but the last layer of weights; however, the learned embedding may overfit to the limited number of preferences over demonstrations and fail to capture features relevant to the ground-truth reward function. Thus, we supplement the pairwise ranking objective with auxiliary objectives that can be optimized in a self-supervised fashion using data from the demonstrations.

Table 5. Self-supervised learning objectives used to pre-train $\phi(s)$.

Inverse Dynamics	$f_{\text{ID}}(\phi(s_t), \phi(s_{t+1})) \rightarrow a_t$
Forward Dynamics	$f_{\text{FD}}(\phi(s_t), a_t) \rightarrow s_{t+1}$
Temporal Distance	$f_{\text{TD}}(\phi(s_t), \phi(s_{t+x}) \rightarrow x$
Variational Autoencoder	$f_A(\phi(s_t)) \rightarrow s_t$

We use the following self-supervised tasks to pre-train R_θ: (1) Learn an inverse dynamics model that uses embeddings $\phi(s_t)$ and $\phi(s_{t+1})$ to predict the corresponding action a_t (Torabi et al., 2018; Hanna & Stone, 2017), (2) Learn a forward dynamics model that predicts s_{t+1} from $\phi(s_t)$ and a_t (Oh et al., 2015; Thananjeyan et al., 2019), (3) Learn an embedding $\phi(s)$ that predicts the temporal distance between two randomly chosen states from the same demonstration (Aytar et al., 2018), and (4) Train a variational pixel-to-pixel autoencoder in which $\phi(s)$ is the learned latent encoding (Makhzani & Frey, 2017; Doersch, 2016). Table 5 summarizes the auxiliary tasks used to train $\phi(s)$.

There are many possibilities for pre-training $\phi(s)$; however, we found that each objective described above encourages the embedding to encode different features. For example, an accurate inverse dynamics model can be learned by only attending to the movement of the agent. Learning forward dynamics supplements this by requiring $\phi(s)$ to encode information about short-term changes to the environment. Learning to predict the temporal distance between states in a trajectory forces $\phi(s)$ to encode long-term progress. Finally, the autoencoder loss acts as a regularizer to the other losses as it seeks to embed all aspects of the state.

In the Atari domain, input to the network is given visually as grayscale frames resized to 84×84. To provide temporal information, four sequential frames are stacked one on top of another to create a *framestack* which provides a brief snapshot of activity. The network architecture takes a framestack, applies four convolutional layers following a similar architecture to Christiano et al. (2017) and Brown et al. (2019b), with leaky ReLU units as non-linearities following each convolution layer. The convolutions follow the following structure:

#	Filter size	Image size	Stride
Input	-	$84 \times 84 \times 4$	-
1	$7x7$	$26 \times 26 \times 16$	3
2	$5x5$	$11 \times 11 \times 32$	2
3	$5x5$	$9 \times 9 \times 32$	1
4	$3x3$	$7 \times 7 \times 16$	1

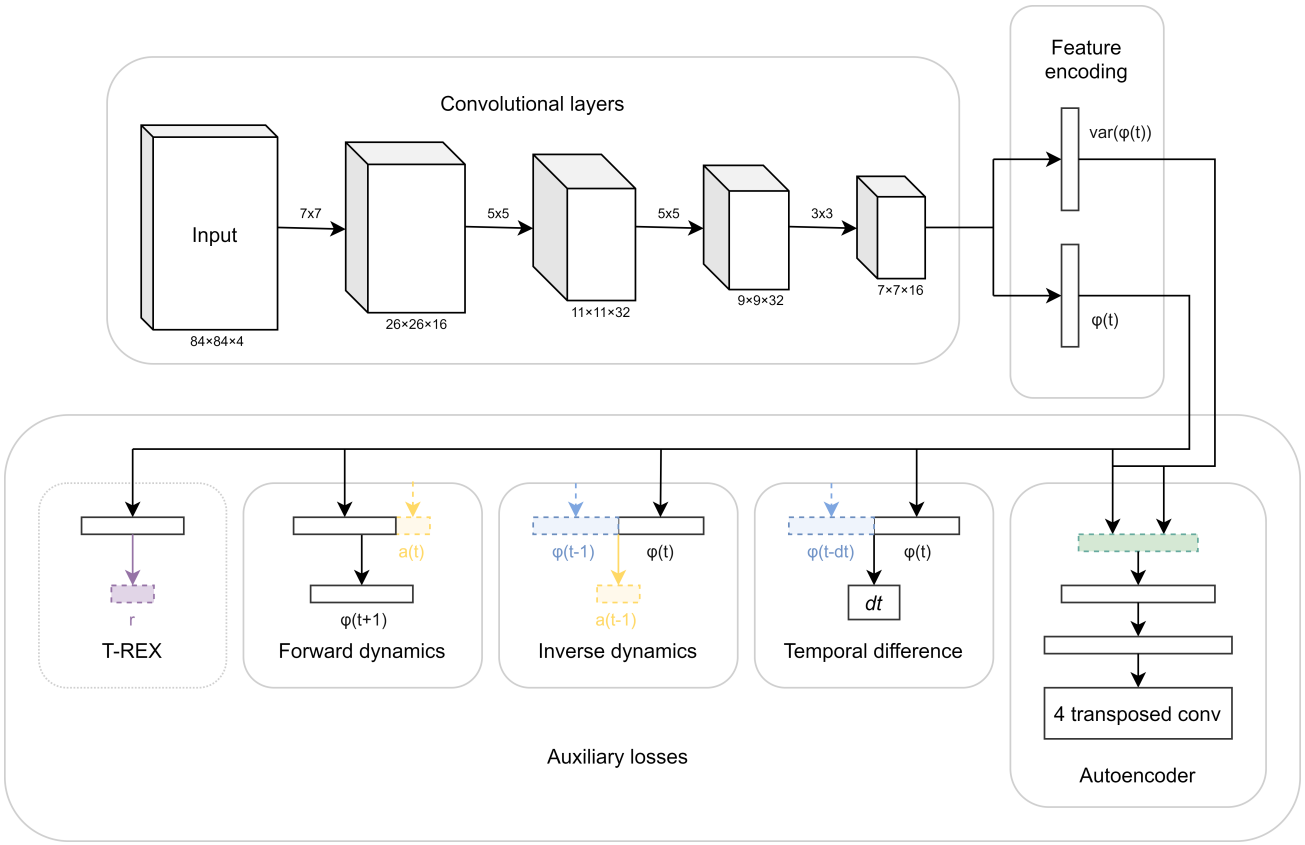

Figure 1. Diagram of the network architecture used when training feature encoding $\phi(s)$ with self-supervised and T-REX losses. Yellow denotes actions, blue denotes feature encodings sampled from elsewhere in a demonstration trajectory, and green denotes random samples for the variational autoencoder.

The convolved image is then flattened. Two sequential fully connected layers, with leaky ReLU applied to the first layer, transform the flattened image into the encoding, $\phi(s)$ where s is the initial framestack. The width of these layers depends on the size of the feature encoding chosen. In our experiments with a latent dimension of 64, the first layer transforms from size 784 to 128 and the second from 128 to 64.

See Figure 1 for a complete diagram of this process.

Architectural information for each auxiliary task is given below.

1. The variational autoencoder (VAE) tries to reconstruct the original framestack from the feature encoding using transposed convolutions. Mirroring the structure of the initial convolutions, two fully connected layers precede four transposed convolution layers. These first two layers transform the 64-dimensional feature encoding from 64 to 128, and from 128 to 1568. The following four layers' structures are summarized below:

#	Filter size	Image size	Stride
Input	-	$28 \times 28 \times 2$	-
1	$3x3$	$30 \times 30 \times 4$	1
2	$6x6$	$35 \times 35 \times 16$	1
3	$7x7$	$75 \times 75 \times 16$	2
4	$10x10$	$84 \times 84 \times 4$	1

   A cross-entropy loss is applied between the reconstructed image and the original, as well as a term added to penalize the KL divergence of the distribution from the unit normal.

2. A temporal difference estimator, which takes two random feature encodings from the same demonstration and predicts the number of timesteps in between. It is a single fully-connected layer, transforming the concatenated feature encodings into a scalar time difference. A mean-squared error loss is applied between the real difference and predicted.

3. An inverse dynamics model, which takes two sequential feature encodings and predicts the action taken in between. It is again a single fully-connected layer,

trained as a classification problem with a binary cross-entropy loss over the discrete action set.

4. A forward dynamics model, which takes a concatenated feature encoding and action and predicts the next feature encoding with a single fully-connected layer. This is repeated 5 times, which increases the difference between the initial and final encoding. It is trained using a mean-squared error between the predicted and real feature encoding.

5. A T-REX loss, which samples feature encodings from two different demonstrations and tries to predict which one of them has preference over the other. This is done with a single fully-connected layer that transforms an encoding into scalar reward, and is then trained as a classification problem with a binary cross-entropy loss. A 1 is assigned to the demonstration sample with higher preference and a 0 to the demonstration sample with lower preference.

In order to encourage a feature encoding that has information easily interpretable via linear combinations, the temporal difference, T-REX, inverse dynamics, and forward dynamics tasks consist of only a single layer atop the feature encoding space rather than multiple layers.

To compute the final loss on which to do the backwards pass, all of the losses described above are summed with weights determined empirically to balance out their values.

D.1. Training specifics

We used an NVIDIA TITAN V GPU for training the embedding. We used the same 12 demonstrations used for MCMC to train the self-supervised and ranking losses described above. We sample 60,000 trajectory snippets pairs from the demonstration pool, where each snippet is between 50 and 100 timesteps long. We use a learning rate of 0.001 and a weight decay of 0.001. We make a single pass through all of the training data using batch size of 1 resulting in 60,000 updates using the Adam (Kingma & Ba, 2014) optimizer. For Enduro prior work (Brown et al., 2019b) showed that full trajectories resulted in better performance than subsampling trajectories. Thus, for Enduro we subsample 10,000 pairs of entire trajectories by randomly selecting a starting time between 0 and 5 steps after the initial state and then skipping every t frames where t is chosen uniformly from the range $[3, 7)$ and train with two passes through the training data. When performing subsampling for either snippets or full trajectories, we subsample pairs of trajectories such that one is from a worse ranked demonstration and one is from a better ranked demonstration following the procedure outlined in (Brown et al., 2019b).

E. Visualizations of Learned Features

Viewable here[1] is a video containing an Enduro demonstration trajectory, its decoding with respect to the pre-trained autoencoder, and a plot of the dimensions in the latent encoding over time. Observe how changes in the demonstration, such as turning right or left or a shift, correspond to changes in the plots of the feature embedding. We noticed that certain features increase when the agent passes other cars while other features decrease when the agent gets passed by other cars. This is evidence that the pretraining has learned features that are relevant to the ground truth reward which gives +1 every time the agent passes a car and -1 every time the agent gets passed.

Viewable here[2] is a similar visualization of the latent space for Space Invaders. Notice how it tends to focus on the movement of enemy ships, useful for game progress in things such as the temporal difference loss, but seems to ignore the player's ship despite its utility in inverse dynamics loss. Likely the information exists in the encoding but is not included in the output of the autoencoder.

Viewable here[3] is visualization of the latent space for Breakout. Observe that breaking a brick often results in a small spike in the latent encoding. Many dimensions, like the dark green curve which begins at the lowest value, seem to invert as game progress continues on, thus acting as a measure of how much time has passed.

F. Imitation Learning Ablations for Pre-training $\phi(s)$

Table 6 shows the results of pre-training reward features only using different losses. We experimented with using only the T-REX Ranking loss (Brown et al., 2019b), only the self-supervised losses shown in Table 1 of the main paper, and using both the T-REX ranking loss plus the self-supervised loss function. We found that performance varried over the different pre-training schemes, but that using Ranking + Self-Supervised achieved high performance across all games, clearly outperforming only using self-supervised losses and achieving superior performance to only using the ranking loss on 3 out of 5 games.

G. Suboptimal Demonstration Details

We used the same suboptimal demonstrations used by Brown et al. (2019b) for comparison. These demonstrations

[1] https://www.youtube.com/watch?v=DMf8kNH9nVg

[2] https://www.youtube.com/watch?v=2uN5uD17H6M

[3] https://www.youtube.com/watch?v=8zgbD1fZOH8

Table 6. Comparison of different reward feature pre-training schemes. Ground-truth average returns for several Atari games when optimizing the mean and MAP rewards found using Bayesian REX. Each algorithm is given the same 12 demonstrations with ground-truth pairwise preferences. The average performance for each IRL algorithm is the average over 30 rollouts.

	Ranking Loss		Self-Supervised		Ranking + Self-Supervised	
Game	Mean	MAP	Mean	MAP	Mean	MAP
Beam Rider	3816.7	4275.7	180.4	143.7	**5870.3**	5504.7
Breakout	389.9	**409.5**	360.1	367.4	393.1	390.7
Enduro	472.7	479.3	0.0	0.0	135.0	**487.7**
Seaquest	675.3	670.7	674.0	683.3	606.0	**734.7**
Space Invaders	**1482.0**	1395.5	391.2	396.2	961.3	1118.8

were obtained by running PPO on the ground truth reward and checkpointing every 50 updates using OpenAI Baselines (Dhariwal et al., 2017). Brown et al. (2019b) make the checkpoint files available, so to generate the demonstration data we used their saved checkpoints and followed the instructions in their released code to generate the data for our algorithm[4]. We gave Bayesian REX these demonstrations as well as ground-truth rankings using the game score; however, other than the rankings, Bayesian REX has no access to the true reward samples. Following the recommendations of Brown et al. (2019b), we mask the game score and other parts of the game that are directly indicative of the game score such as the number of enemy ships left, the number of lives left, the level number, etc. See (Brown et al., 2019b) for full details.

H. Reinforcement Learning Details

We used the OpenAI Baselines implementation of Proximal Policy Optimization (PPO) (Schulman et al., 2017; Dhariwal et al., 2017). We used the default hyperparameters for all games and all experiments. We run RL for 50 million frames and then take the final checkpoint to perform evaluations. We adapted the OpenAI Baselines code so even though the RL agent receives a standard preprocessed observation, it only receives samples of the reward learned via Bayesian REX, rather than the ground-truth reward. T-REX (Brown et al., 2019b) uses a sigmoid to normalize rewards before passing them to the RL algorithm; however, we obtained better performance for Bayesian REX by feeding the unnormalized predicted reward $R_\theta(s)$ into PPO for policy optimization. We follow the OpenAI baselines default preprocessing for the framestacks that are fed into the RL algorithm as observations. We also apply the default OpenAI baselines wrappers the environments. We run PPO with 9 workers on an NVIDIA TITAN V GPU.

[4]Code from (Brown et al., 2019b) is available here `https://github.com/hiwonjoon/ICML2019-TREX`

I. High-Confidence Policy Performance Bounds

In this section we describe the details of the policy performance bounds.

I.1. Policy Evaluation Details

We estimated $\Phi_{\pi_{\text{eval}}}$ using C Monte Carlo rollouts for each evaluation policy. Thus, after generating C rollouts, $\tau_1, \ldots, \tau_C$ from π_{eval} the feature expectations are computed as

$$\Phi_{\pi_{\text{eval}}} = \frac{1}{C}\left[\sum_{i=1}^{C}\sum_{s\in\tau_i}\phi(s)\right]. \tag{5}$$

We used $C = 100$ for all experiments.

I.2. Evaluation Policies

We evaluated several different evaluation policies. To see if the learned reward function posterior can interpolate and extrapolate we created four different evaluation policies: A, B, C, and D. These policies were created by running RL via PPO on the ground truth reward for the different Atari games. We then checkpointed the policy and selected checkpoints that would result in different levels of performance. For all games except for Enduro these checkpoints correspond to 25, 325, 800, and 1450 update steps using OpenAI baselines. For Enduro, PPO performance was stuck at 0 return until much later in learning. To ensure diversity in the evaluation policies, we chose to use evaluation policies corresponding to 3125, 3425, 3900, and 4875 steps. We also evaluated each game with a No-Op policy. These policies are often adversarial for some games, such as Seaquest, Breakout, and Beam Rider, since they allow the agent to live for a very long time without actually playing the game—a potential way to hack the learned reward since most learned rewards for Atari will incentivize longer gameplay.

The results for Beam Rider and Breakout are shown in the main paper. For completeness, we have included the high-confidence policy evaluation results for the other games here

Table 7. Policy evaluation statistics for Enduro over the return distribution from the learned posterior $P(R|D, \mathcal{P})$ compared with the ground truth returns using game scores. Policies A-D correspond to checkpoints of an RL policy partially trained on the ground-truth reward function and correspond to 25, 325, 800, and 1450 training updates to PPO. No-Op that always plays the no-op action, resulting in high mean predicted performance but low 95%-confidence return (0.05-VaR).

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
A	324.7	48.2	7.3	3322.4
B	328.9	52.0	26.0	3322.4
C	424.5	135.8	145.0	3389.0
D	526.2	192.9	199.8	3888.2
Mean	1206.9	547.5	496.7	7249.4
MAP	395.2	113.3	133.6	3355.7
No-Op	245.9	-31.7	0.0	3322.0

in the Appendix. Table 7 shows the high-confidence policy evaluation results for Enduro. Both the average returns over the posterior as well as the the high-confidence performance bounds ($\delta = 0.05$) demonstrate accurate predictions relative to the ground-truth performance. The No-Op policy results in the racecar slowly moving along the track and losing the race. This policy is accurately predicted as being much worse than the other evaluation policies. We also evaluated the Mean and MAP policies found by optimizing the Mean reward and MAP reward from the posterior obtained using Bayesian REX. We found that the learned posterior is able to capture that the MAP policy is more than twice as good as the evaluation policy D and that the Mean policy has performance somewhere between the performance of policies B and C. These results show that Bayesian REX has the potential to predict better-than-demonstrator performance (Brown et al., 2019a).

Table 8 shows the results for high-confidence policy evaluation for Seaquest. The results show that high-confidence performance bounds are able to accurately predict that evaluation policies A and B are worse than C and D. The ground truth performance of policies C and D are too close and the mean performance over the posterior and 0.05-VaR bound on the posterior are not able to find any statistical difference between them. Interestingly the no-op policy has very high mean and 95%-confidence lower bound, despite not scoring any points. However, as shown in the bottom half of Table 8, adding one more ranked demonstration from a 3000 length segment of a no-op policy solves this problem. These results motivate a natural human-in-the-loop approach for safe imitation learning.

Finally, Table 9 shows the results for high-confidence policy evaluation for Space Invaders. The results show that using both the mean performance and 95%-confidence lower bound are good indicators of ground truth performance for the evaluation polices. The No-Op policy for Space Invaders results in the agent getting hit by alien lasers early in the game. The learned reward function posterior correctly assigns low average performance and indicates high risk with a low 95%-confidence lower bound on the expected return of the evaluation policy.

Table 8. Policy evaluation statistics for Seaquest over the return distribution from the learned posterior $P(R|D, \mathcal{P})$ compared with the ground truth returns using game scores. Policies A-D correspond to checkpoints of an RL policy partially trained on the ground-truth reward function and correspond to 25, 325, 800, and 1450 training updates to PPO. No-Op always plays the no-op action, resulting in high mean predicted performance but low 0.05-quantile return (0.05-VaR). Results predict that No-Op is much better than it really is. However, simply adding a single ranked rollout from the No-Op policy and rerunning MCMC results in correct relative rankings with respect to the No-Op policy

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
A	24.3	10.8	338.6	1077.8
B	53.6	24.1	827.2	2214.1
C	56.0	25.4	872.2	2248.5
D	55.8	25.3	887.6	2264.5
No-Op	2471.6	842.5	0.0	99994.0
Results after adding one ranked demo from No-Op				
A	0.5	-0.5	338.6	1077.8
B	3.7	2.0	827.2	2214.1
C	3.8	2.1	872.2	2248.5
D	3.2	1.5	887.6	2264.5
No-Op	-321.7	-578.2	0.0	99994.0

J. Different Evaluation Policies

To test Bayesian REX on different learned policies we took a policy trained with RL on the ground truth reward function for Beam Rider, the MAP policy learned via Bayesian REX for Beam Rider, and a policy trained with an earlier version of Bayesian REX (trained without all of the auxiliary losses) that learned a novel reward hack where the policy repeatedly presses left and then right, enabling the agent's ship to stay in between two of the firing lanes of the enemies. The imitation learning reward hack allows the agent to live for a very long time. We took a 2000 step prefix of each policy and evaluated the expected and 5th perentile worst-case predicted returns for each policy. We found that Bayesian REX is able to accurately predict that the reward hacking policy is worse than both the RL policy and the policy optimizing the Bayesian REX reward. However, we found

Table 9. Policy evaluation statistics for Space Invaders over the return distribution from the learned posterior $P(R|D, \mathcal{P})$ compared with the ground truth returns using game scores. Policies A-D correspond to checkpoints of an RL policy partially trained on the ground-truth reward function and correspond to 25, 325, 800, and 1450 training updates to PPO. The mean and MAP policies are the results of PPO using the mean and MAP rewards, respectively. No-Op that always plays the no-op action, resulting in high mean predicted performance but low 0.05-quantile return (0.05-VaR).

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
A	45.1	20.6	195.3	550.1
B	108.9	48.7	436.0	725.7
C	148.7	63.6	575.2	870.6
D	150.5	63.8	598.2	848.2
Mean	417.4	171.7	1143.7	1885.7
MAP	360.2	145.0	928.0	1629.5
NoOp	18.8	3.8	0.0	504.0

Table 10. Beamrider policy evaluation for an RL policy trained on ground truth reward, an imitation learning policy, and a reward hacking policy that exploits a game hack to live for a long time by moving quickly back and forth.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
RL	36.7	19.5	2135.2	2000
B-REX	68.1	38.1	649.4	2000
Hacking	28.8	10.2	2.2	2000

that the Bayesian REX policy, while not performing as well as the RL policy, was given higher expected return and a higher lower bound on performance than the RL policy. Results are shown in Table 10.

K. Human Demonstrations

To investigate whether Bayesian REX is able to correctly rank human demonstrations, one of the authors provided demonstrations of a variety of different behaviors and then we took the latent embeddings of the demonstrations and used the posterior distribution to find high-confidence performance bounds for these different rollouts.

K.1. Beamrider

We generated four human demonstrations: (1) *good*, a good demonstration that plays the game well, (2) *bad*, a bad demonstration that seeks to play the game but does a poor job, (3) *pessimal*, a demonstration that does not shoot enemies and seeks enemy bullets, and (4) *adversarial* a demonstration that pretends to play the game by moving and shooting as much as possibly but tries to avoid actually shooting enemies. The results of high-confidence policy evaluation are shown in Table 11. The high-confidence bounds and average performance over the posterior correctly rank the behaviors. This provides evidence that the learned linear reward correctly rewards actually destroying aliens and avoiding getting shot, rather than just flying around and shooting.

Table 11. Beam Rider evaluation of a variety of human demonstrations.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	12.4	5.8	1092	1000.0
bad	10.7	4.5	396	1000.0
pessimal	6.6	0.8	0	1000.0
adversarial	8.4	2.4	176	1000.0

Table 12. Space Invaders evaluation of a variety of human demonstrations.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	198.3	89.2	515	1225.0
every other	56.2	25.9	315	728.0
hold 'n fire	44.3	18.6	210	638.0
shoot shelters	47.0	20.6	80	712.0
flee	45.1	19.8	0	722.0
hide	83.0	39.0	0	938.0
miss	66.0	29.9	0	867.0
pessimal	0.5	-13.2	0	266.0

K.2. Space Invaders

For Space Invaders we demonstrated an even wider variety of behaviors to see how Bayesian REX would rank their relative performance. We evaluated the following policies: (1) *good*, a demonstration that attempts to play the game as well as possible, (2) *every other*, a demonstration that only shoots aliens in the 2nd and 4th columns, (3) *flee*, a demonstration that did not shoot aliens, but tried to always be moving while avoiding enemy lasers, (4) *hide*, a demonstration that does not shoot and hides behind on of the barriers to avoid enemy bullets, (5) *pessimal*, a policy that seeks enemy bullets while not shooting, (6) *shoot shelters*, a demonstration that tries to destroy its own shelters by shooting at them, (7) *hold 'n fire*, a demonstration where the player rapidly fires but does not move to avoid enemy lasers, and (8) *miss*, a demonstration where the demonstrator tries to fire but not hit any aliens while avoiding enemy lasers.

Table 12 shows the results of evaluating the different demonstrations. The good demonstration is clearly the best per-

Table 13. Space Invaders evaluation of a variety of human demonstrations when considering only the first 6000 steps.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	47.8	22.8	515	600.0
every other	34.6	15.0	315	600.0
hold 'n fire	40.9	17.1	210	600.0
shoot shelters	33.0	13.3	80	600.0
flee	31.3	11.9	0	600.0
hide	32.4	13.8	0	600.0
miss	30.0	11.3	0	600.0

forming demonstration in terms of mean performance and 95%-confidence lower bound on performance and the pessimal policy is correctly given the lowest performance lower bound. However, we found that the length of the demonstration appears to have a strong effect on the predicted performance for Space Invaders. Demonstrations such as hide and miss are able to live for a longer time than policies that actually hit aliens. This results in them having higher 0.05-quantile worst-case predicted performance and higher mean performance.

To study this further we looked at only the first 600 timesteps of each policy, to remove any confounding by the length of the trajectory. The results are shown in Table 13. With a fixed length demonstration, Bayesian REX is able to correctly rank *good*, *every other*, and *hold 'n fire* as the best demonstrations, despite evaluation policies that are deceptive.

K.3. Enduro

For Enduro we tested four different human demonstrations: (1) *good* a demonstration that seeks to play the game well, (2) *periodic* a demonstration that alternates between speeding up and passing cars and then slowing down and being passed, (3) *neutral* a demonstration that stays right next to the last car in the race and doesn't try to pass or get passed, and (4) *ram* a demonstration that tries to ram into as many cars while going fast. Table 14 shows that Bayesian REX is able to accurately predict the performance and risk of each of these demonstrations and gives the highest (lowest 0.05-VaR) risk to the *ram* demonstration and the least risk to the *good* demonstration.

L. Comparison with Other Methods for Uncertainty Quantification

Bayesian REX is only one possible method for measure uncertainty. Other popular methods for measuring epistemic uncertainty include using bootstrapping to create an ensemble of neural networks (Lakshminarayanan et al., 2017) and using dropout as an approximation of MCMC sampling (Gal & Ghahramani, 2016). In this section we compare our fully Bayesian approach with these two approximations.

Table 14. Enduro evaluation of a variety of human demonstrations.

	Predicted		Ground Truth	
Policy	Mean	0.05-VaR	Avg.	Length
good	246.7	-113.2	177	3325.0
periodic	230.0	-130.4	44	3325.0
neutral	190.8	-160.6	0	3325.0
ram	148.4	-214.3	0	3325.0

L.1. T-REX Ensemble

We used the same implementation used by Brown et al. (2019b)[5], but trained an ensemble of five T-REX networks using the same training demonstrations but with randomized seeds so each network is intitialized differently and has a different training set of subsampled snippets from the full length ranked trajectories. To estimate the uncertainty over the return of a trajectory or policy we run the trajectory through each network to get a return estimate or run multiple rollouts of the policy through each member of the ensemble to get a distribution over returns. We used 100 rollouts for the evaluation policies.

L.2. MC Dropout

For the MC Dropout baseline we used the same base architecture as T-REX and Bayesian REX, except that we did not add additional auxiliary losses, but simply trained the base network to predict returns using dropout during training. For each training pair of trajectories we randomly sample a dropout mask on the last layer of weights. Because MC dropout is supposed to approximate a large ensemble, we kept the dropout mask consistent across each sampled preference pair such that the same portions of the network are dropped out for each frame of each trajectory and for both the more preferred and less preferred trajectories. Thus, for each training sample, consisting of a more and less preferred trajectory, we sample a random dropout mask and then apply this same mask across all states in both trajectories. To keep things as similar to Bayesian REX as possible, we used full trajectories with the same pairwise preferences used by Bayesian REX.

To estimate the posterior distribution over returns for a trajectory or policy we simply sampled 50 random masks on the last layer of weights. Thus, this method corresponds to the MC dropout equivalent of Bayesian REX where the latent

[5]https://github.com/hiwonjoon/icml2019-trex

state encoding is trained end-to-end via dropout rather than pre-trained and where the posterior distribution is estimated via randomly dropping out weights on the corresponding linear reward function. We applied these 50 random dropouts to each of 100 rollouts for each evaluation policy. We used a dropout probability of 0.5.

Table 15 shows the results for running RL on the learned reward functions. The results show that Bayesian REX is superior or competitive with T-REX Ensemble and MC Dropout across all games except Beam Rider, where MC Dropout performs much better.

L.3. T-REX Ensemble High-Confidence Bounds

Tables 16–20 show the results for evaluating different evaluation policies via high-confidence performance bounds. Table 16 shows that the ensemble has accurate expected returns, but that the 95% confidence lower bounds are not informative and do not represent risk as accurately as Bayesian REX since policy D is viewed as much worse than policy A. Note that we normalized the predicted scores by calculating the average predicted return of each ensemble member for rollouts from policy A and then using this as a baseline for all other predictions of each ensemble member by subtracting off the average predicted return for policy A from return predictions of other policies. Tables 17 and 19 show that the T-REX Ensemble can sometimes fail to produce meaningful predictions for the expectation or the 95% worst-case bounds. Table 18 and 20 show good predictions.

L.4. MC Dropout Results

Tables 21–25 show the results for high-confidence bounds for MC Dropout. Tables 21 and 25 show that MC Dropout is able to accurately predict high risk for the Beam Rider and Space Invaders No-Op policies. However, table 22 23, and 24 show that MC Dropout often fails to predict that the No-Op policy has high risk. Recent work has shown that MC Dropout is not a principled Bayesian approximation since the distribution obtained from MC Dropout does not concentrate in the limit as the number of data samples goes to infinity and thus does not necessarily measure the kind of epistemic risk we are interested in (Osband, 2016). Thus, while MC Dropout does not perform full Bayesian inference like Bayesian REX, it appears to work sometimes in practice. Future work should examine more sophisticated applications of dropout to uncertainty estimation that seek to solve the theoretical and practical problems with vanilla MC Dropout (Hron et al., 2018).

References

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*, 2018.

Brown, D. S. and Niekum, S. Efficient Probabilistic Performance Bounds for Inverse Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2018.

Brown, D. S., Goo, W., and Niekum, S. Better-than-demonstrator imitation learning via automaticaly-ranked demonstrations. In *Conference on Robot Learning (CoRL)*, 2019a.

Brown, D. S., Goo, W., Prabhat, N., and Niekum, S. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, 2019b.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pp. 4299–4307, 2017.

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

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. `https://github.com/openai/baselines`, 2017.

Doersch, C. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, 2016.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Hanna, J. P. and Stone, P. Grounded action transformation for robot learning in simulation. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Hron, J., Matthews, D. G., Ghahramani, Z., et al. Variational bayesian dropout: Pitfalls and fixes. In *35th International Conference on Machine Learning, ICML 2018*, volume 5, pp. 3199–3219, 2018.

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

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pp. 6402–6413, 2017.

Table 15. Comparison of policy performance when using a reward function learned by Bayesian REX, a T-REX ensemble, and a dropout version of Bayesian REX. The results show averages (standard deviations over 30 rollouts. Bayesian REX results in comparable or better performance across all games except Beam Rider.

	Bayesian REX			
Game	Mean	MAP	T-REX Ensemble	MC Dropout
Beam Rider	5870.3 (1905.1)	5504.7 (2121.2)	5925.0 (2097.9)	**7243.1** (2543.6)
Breakout	**393.1** (63.7)	390.7 (48.8)	253.7 (136.2)	52.6 (10.1)
Enduro	135.0 (24.8)	**487.7** (89.4)	281.5 (95.2)	111.8 (17.9)
Seaquest	606.0 (37.6)	**734.7** (41.9)	0.0 (0.0)	0.0 (0.0)
Space Invaders	961.3 (392.3)	1118.8 (483.1)	**1164.8** (546.3)	387.5 (166.3)

Table 16. Beam Rider T-REX ensemble.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	0.0	-119.5	454.4	1372.6
B	76.1	-63.7	774.8	1412.8
C	201.3	-173.8	1791.8	2389.9
D	282.0	-304.0	2664.5	2965.0
Mean	956.9	-2294.8	5736.8	9495.6
MAP	1095.7	-2743.4	5283.0	11033.4
No-Op	-1000.0	-5643.7	0.0	99,994.0

Table 17. Breakout T-REX ensemble.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	0.0	-506.3	1.8	202.7
B	-305.8	-1509.1	16.1	608.4
C	-241.1	-1780.7	24.8	849.3
D	-57.9	-2140.0	42.7	1020.8
Mean	-5389.5	-867.4	388.9	13762.1
MAP	-3168.5	-1066.5	401.0	8780.0
No-Op	-39338.4	-95987.2	0.0	99,994.0

Table 18. Enduro T-REX ensemble.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	-0.0	-137.7	9.4	3322.4
B	23.6	-157.4	23.2	3322.4
C	485.4	232.2	145.6	2289.0
D	1081.1	270.3	214.2	3888.2
Mean	3408.9	908.0	496.7	7249.4
MAP	23.2	-1854.1	133.6	3355.7
No-Op	-1618.1	-3875.9	0.0	3322.0

Table 19. Seaquest T-REX ensemble.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	0.0	-320.9	321.0	1077.8
B	-425.2	-889.3	826.6	2214.1
C	-336.7	-784.3	863.4	2248.5
D	-386.3	-837.3	884.4	2264.5
Mean	-1013.1	-2621.8	721.8	2221.7
MAP	-636.8	-1820.1	607.4	2247.2
No-Op	-19817.9	-28209.7	0.0	99,994.0

Makhzani, A. and Frey, B. J. Pixelgan autoencoders. In *Advances in Neural Information Processing Systems*, pp. 1975–1985, 2017.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pp. 2863–2871, 2015.

Osband, I. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

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

Thananjeyan, B., Balakrishna, A., Rosolia, U., Li, F., McAllister, R., Gonzalez, J. E., Levine, S., Borrelli, F., and Goldberg, K. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. *arXiv preprint arXiv:1905.13402*, 2019.

Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.

Table 20. Space Invaders T-REX ensemble.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	0.0	-136.8	159.4	550.1
B	257.3	-47.9	425.0	725.7
C	446.5	-6.0	553.1	870.6
D	443.3	9.0	591.5	848.2
Mean	1105.6	-392.4	1143.7	1885.7
MAP	989.0	-387.2	928.0	1629.5
No-Op	-211.9	-311.9	0.0	504.0

Table 21. Beam Rider MC Dropout.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	20.9	-1.5	454.4	1372.6
B	27.9	2.3	774.8	1412.8
C	48.7	8.3	1791.8	2389.9
D	63.5	11.0	2664.5	2965.0
Mean	218.2	-89.2	5736.8	1380
MAP	211.2	-148.7	5283.0	708
No-Op	171.2	-3385.7	0.0	99,994.0

Table 22. Breakout MC Dropout.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	10.8	5.2	1.8	202.7
B	33.1	17.7	16.1	608.4
C	43.5	24.1	24.8	849.3
D	56.0	28.5	42.7	1020.8
Mean	822.9	77.3	388.9	13762.1
MAP	519.7	73.8	401.0	8780.0
No-Op	6050.7	3912.4	0.0	99,994.0

Table 23. Enduro MC Dropout.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	541.7	398.0	7.3	3322.4
B	543.6	401.0	26.4	3322.4
C	556.7	409.3	142.5	3389.0
D	663.3	422.3	200.3	3888.2
Mean	2473.0	1701.7	496.7	7249.4
MAP	1097.3	799.5	133.6	3355.7
No-Op	1084.1	849.8	0.0	3322.0

Table 24. Seaquest MC Dropout.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	98.9	49.5	321.0	1077.8
B	258.8	194.8	826.6	2214.1
C	277.7	213.2	863.4	2248.5
D	279.6	214.2	884.4	2264.5
Mean	375.6	272.8	721.8	2221.7
MAP	426.3	319.8	607.4	2247.2
No-Op	16211.1	10478.5	0.0	99,994.0

Table 25. Space Invaders MC Dropout.

	Predicted		Ground Truth Avg.	
Policy	Mean	0.05-VaR	Score	Length
A	10.6	0.8	195.3	550.1
B	22.3	8.8	434.9	725.7
C	26.7	9.8	535.3	870.6
D	28.9	15.6	620.9	848.2
Mean	125.9	54.4	1143.7	848.2
MAP	110.6	52.5	928.0	1885.7
No-Op	8.4	-8.6	0.0	504.0