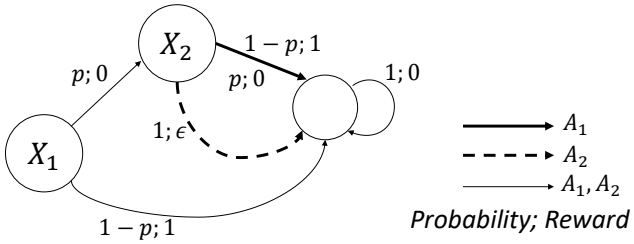
For details on reproducing all our experiment results, including hardware and software requirements, please consult the README.txt file in the root folder of the included code.

## A MDP with stationary and Markov CVaR-optimal policy

We compare the actual performance of our proposed distributional Bellman operator $\widetilde{\mathcal{T}}_\psi$ against that of the Markov action-selection strategy $\widetilde{\mathcal{T}}_\alpha^D$ on the MDP in Figure 1(c), repeated here for reference:




As shown in the proof of Proposition 1, $\widetilde{\mathcal{T}}_\alpha^D$ converges to a suboptimal $\alpha$-CVaR policy if we choose $p, \epsilon, \alpha$ such that $0 < \epsilon < p < 1 - \epsilon$ and $p^2 + \epsilon < \alpha < p$. We now test both $\widetilde{\mathcal{T}}_\alpha^D$ and $\widetilde{\mathcal{T}}_\psi$ on this MDP using quantile-regression distributional RL.

For this simple MDP, we use a linear neural network with one-hot encoding of the state, equivalent to a tabular representation. The hyperparameters such as the learning rate for Adam, schedule for exploration, buffer size for the examples etc can be found in the included source code to reproduce the results presented here. We observe that the results are not overly sensitive to these hyperparameters, although we did not perform an extensive hyperparameter search.

We fix $\epsilon = 0.01$ and choose $\alpha = 0.9p$ for $p$ from 0.1 to 0.8, which satisfies the conditions above. Figure 6 shows the performance of the policies throughout the training period in terms of their $\alpha$-level CVaR for each $p$. Clearly, we can see that the proposed $\widetilde{\mathcal{T}}_\psi$ is able to learn the optimal CVaR policy across all $p$ while $\widetilde{\mathcal{T}}_\alpha^D$ as predicted by Proposition 1 failed to do so.

## B MDP with non-stationary CVaR-optimal policy

Figure 5 shows an example MDP where the optimal CVaR policy is non-stationary and non-Markov.

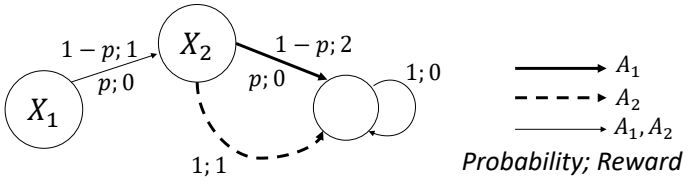



Figure 5: MDP with non-stationary optimal CVaR policy

There are two stationary policies here: always choosing $A_1$ or $A_2$ in $X_2$. There are also two non-stationary policies, corresponding to choosing a different action in $X_2$ based on the outcome at $X_1$, i.e. whether the reward received is 1 or 0. For $0 < p < 1/2$, and let $\alpha = p$, the optimal $\alpha$-CVaR policy is to choose $A_1$ if 0 reward is received at $X_1$ and $A_2$ otherwise. The reader can easily verify this by examining the return distribution for each of the four policies and computing their respective $\alpha$-CVaR.

As in the previous section, we compare the performance of $\widetilde{\mathcal{T}}_\alpha^D$ against that of $\widetilde{\mathcal{T}}_\psi$. We use the same representation and the same set of hyperparameters for training. Again, the complete code to reproduce the results can be found in the supplementary material.

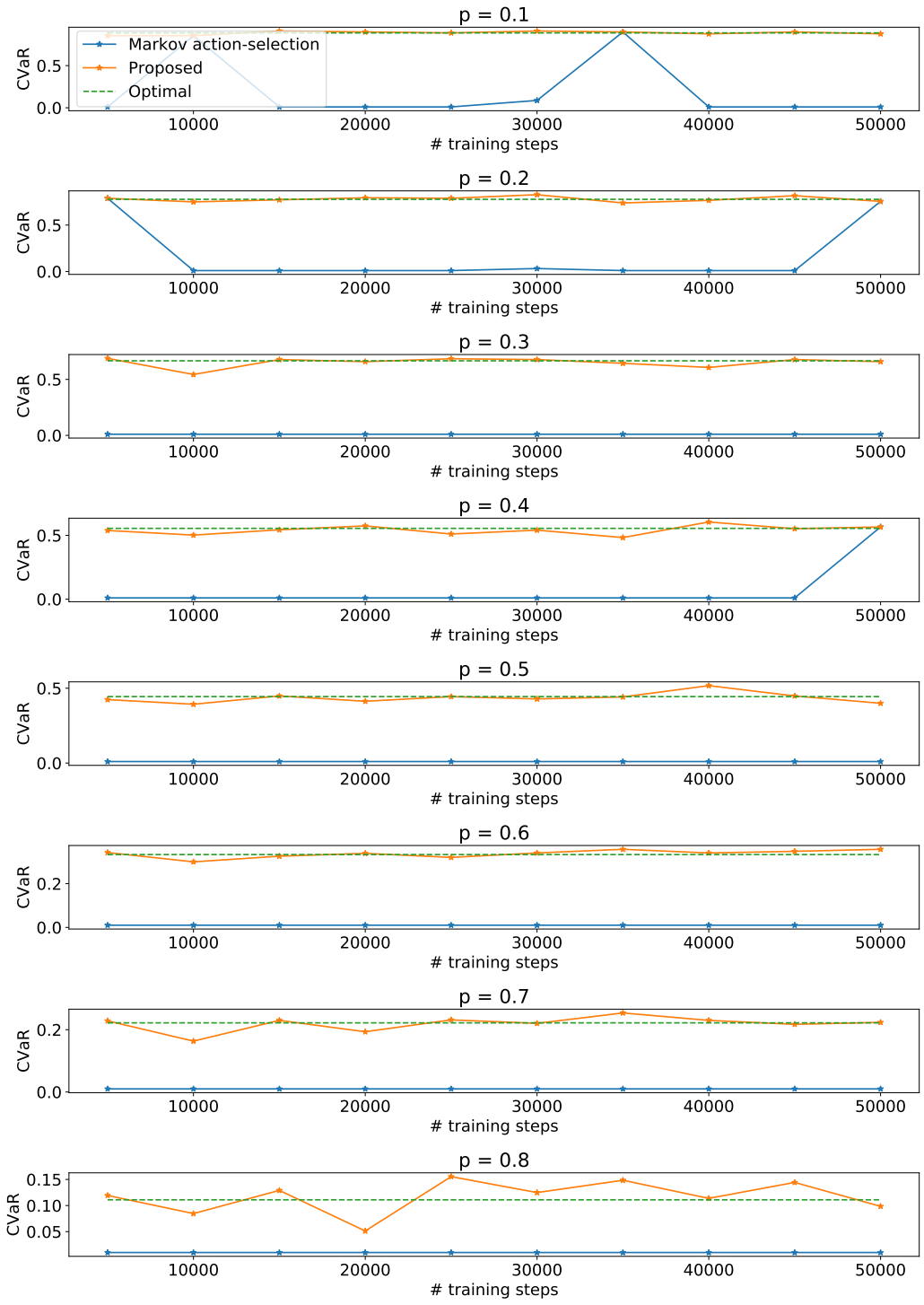



Figure 6: Evaluation results over the training period for the MDP in Figure 1(c). Each point represents the estimated $\alpha$-level CVaR using 1000 independent evaluation episodes. The optimal CVaR is shown in green, dashed lines.

Figure 7 shows the results for $p$ from 0.1 to 0.4. While $\widetilde{\mathcal{T}}_\alpha^D$ converges to a stationary and therefore suboptimal policy, we observe that $\widetilde{\mathcal{T}}_\psi$ converges to the optimal policy. This shows that our proposed Algorithm 1 and 2 can learn, extract and execute non-stationary policies that are CVaR-optimal.

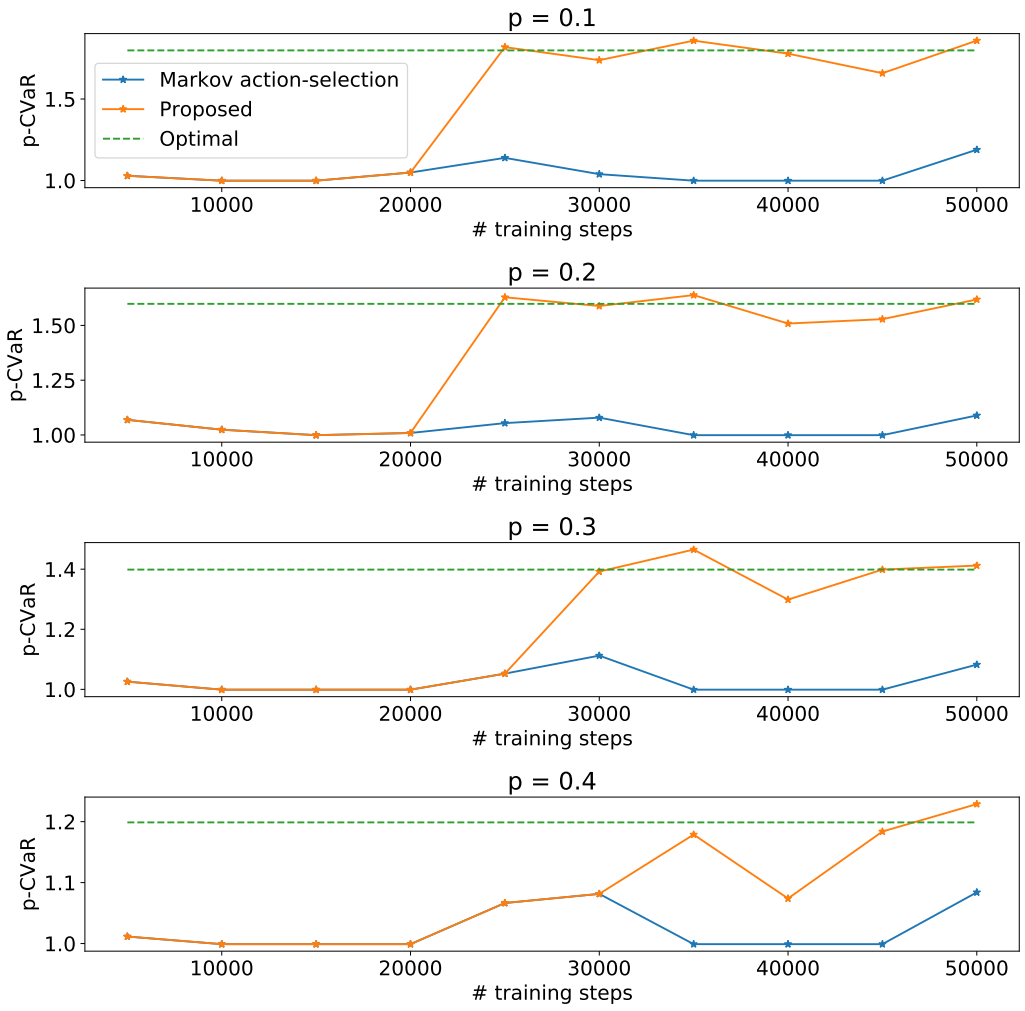



Figure 7: Evaluation results over the training period for the MDP in Figure 5. Each point represents the estimated $p$-level CVaR using 1000 independent evaluation episodes. The optimal CVaR is shown in green, dashed lines.

Figures 8 and 9 show how the distributions for each state-action pair change during the training process for $\widetilde{\mathcal{T}}_\alpha^D$ and $\widetilde{\mathcal{T}}_\psi$ respectively, for the case $p = 0.2$. Similar patterns can be observed for other $p$.

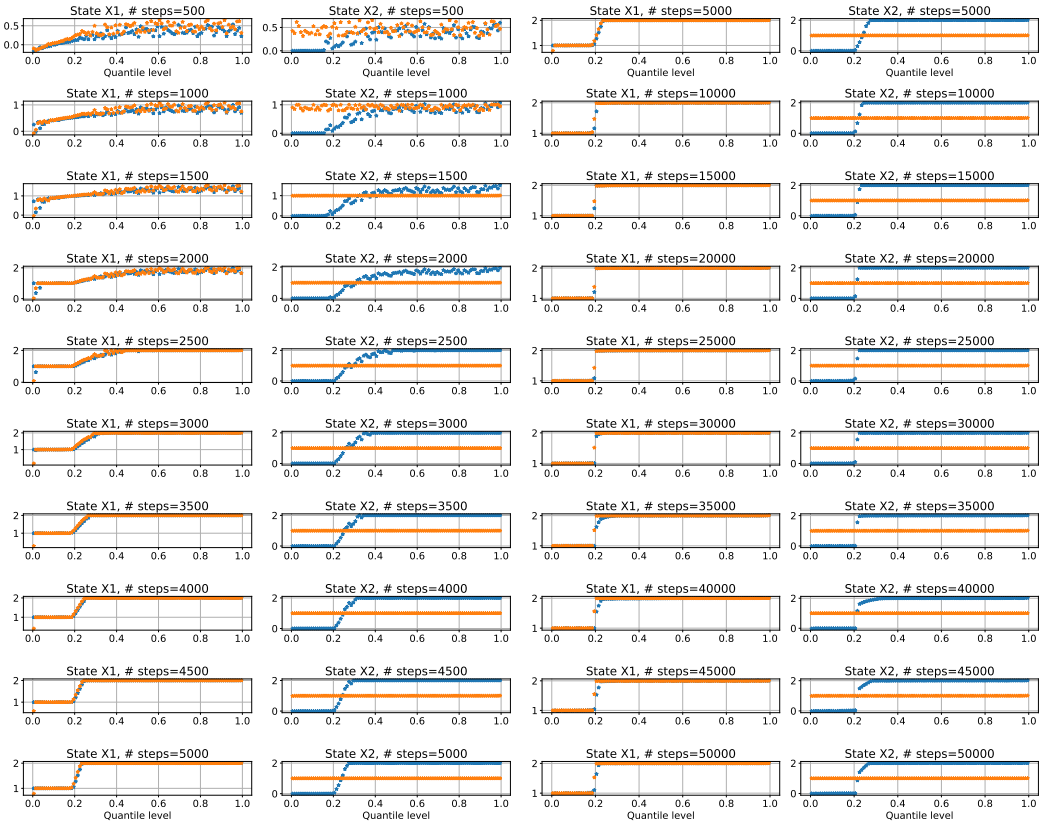



Figure 8: Learning progress for $\widetilde{\mathcal{T}}_{\alpha}^{D}$, showing $\theta_i(x, a)$ for each state-action pair. First 2 columns: the initial 5000 steps, at 500-step intervals. Last 2 columns: 50000 steps, at 5000-step intervals. Action $A_1$ in blue while Action $A_2$ in orange.

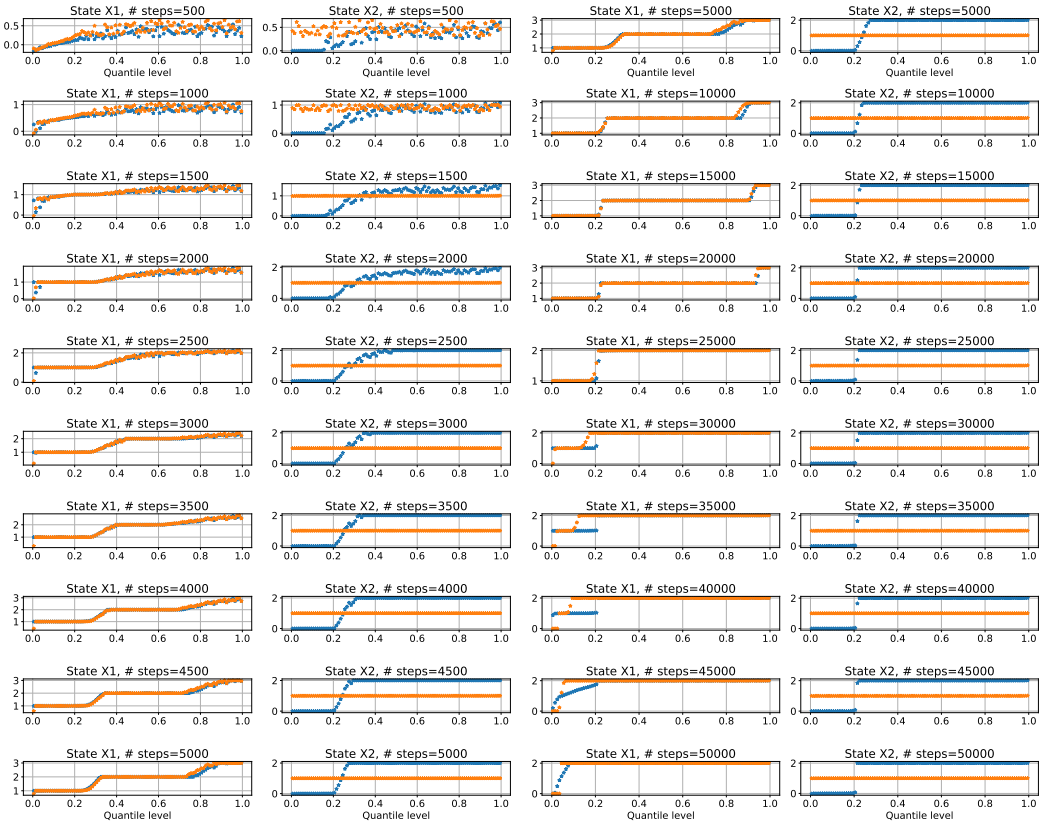



Figure 9: Learning progress for $\widetilde{\mathcal{T}}_\psi$, showing $\theta_i(x, a)$ for each state-action pair. First 2 columns: the initial 5000 steps, at 500-step intervals. Last 2 columns: 50000 steps, at 5000-step intervals. Action $A_1$ in blue while Action $A_2$ in orange.

## C Modified Puddle World

The original Puddle World (Sutton and Barto, 2018) is a continuous-state 2D navigation task within a unit square filled with "puddles". The 4 actions correspond to moving in the 4 directions, with a Gaussian noise added to each step. While each step has a cost of 1 (i.e. reward -1), stepping into a puddle will result in an additional penalty with magnitude that scales with how close the player is to the center of the puddle. We modify the task such that the in-puddle penalty only happens with probability $0.1$. Stepping into the puddles is therefore an uncertain and risky move. The objective is to reach a goal region (upper right corner) from the initial state (upper left corner) with as little cost as possible.

We compare the Markov action-selection strategy $\widetilde{\mathcal{T}}_\alpha^D$ (labeled "Dynamic") with our proposed $\widetilde{\mathcal{T}}_\psi$ (labeled "Static") using quantile-regression distributional RL. Figure 10 shows the results for CVaR at $\alpha = 0.2$ after 1 million training steps, on 5 random seeds. The best checkpoint is selected for each seed and the policy is evaluated for 1000 episodes, each episode ends after the player reaches the goal state or after 100 steps, whichever happens earlier. Each plot shows the histogram for the returns over the 1000 episodes. The first column (labeled "Expectation") shows performance of the best policy that optimizes for the expected return. The second and the third column show results for $\widetilde{\mathcal{T}}_\alpha^D$ and $\widetilde{\mathcal{T}}_\psi$ respectively.

While the "Expectation" policies achieve that best expected returns, we see that the proposed approach learns better CVaR-optimized policies. Figures 11, 12 and 13 show sample trajectories for the learned policies from the first 3 random seeds, for each of the strategies. The gray-color regions are the puddles, the green cross marks the start state while the goal region is in yellow.

The complete code to reproduce all our results here can be found in the supplementary material.

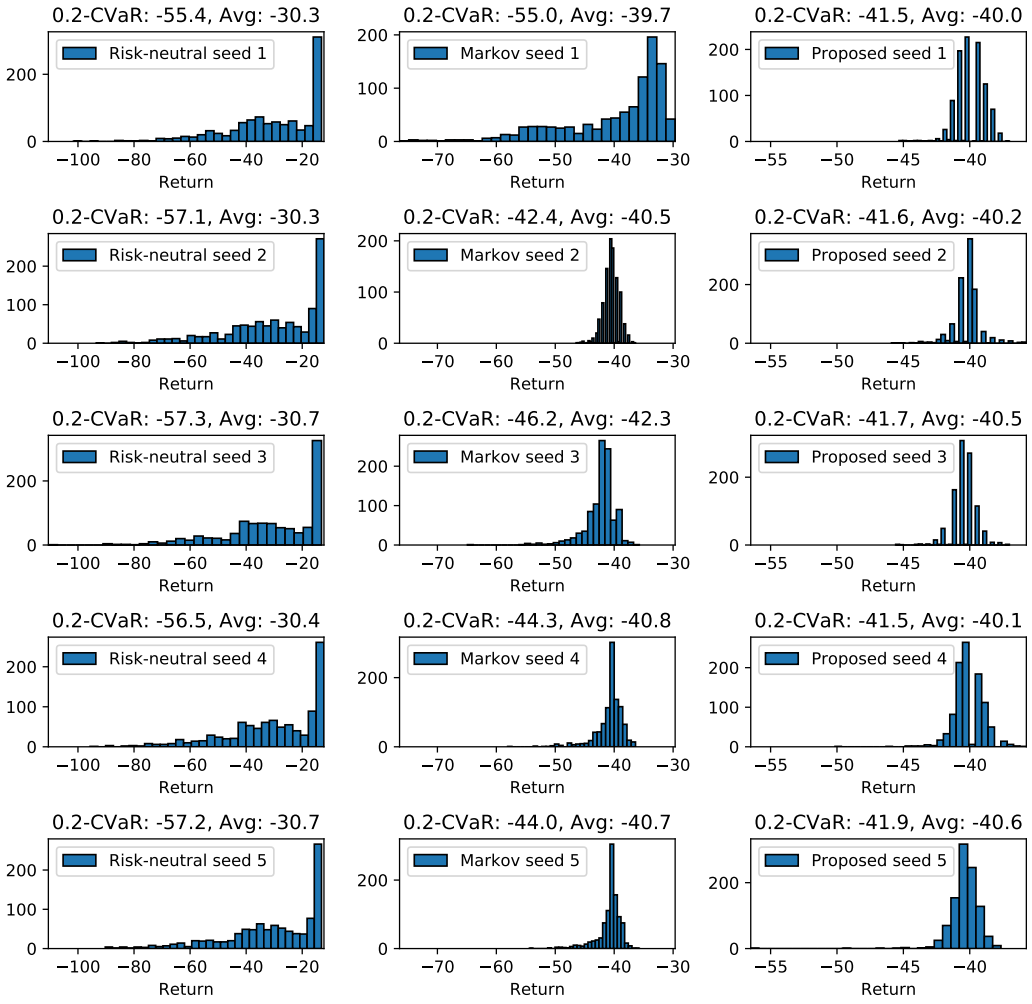



Figure 10: Histograms for returns over 1000 episodes for the best policy of each random seed. First column: policies optimized for the expected return. Middle column: Markov action-selection. Right column: the proposed algorithm.

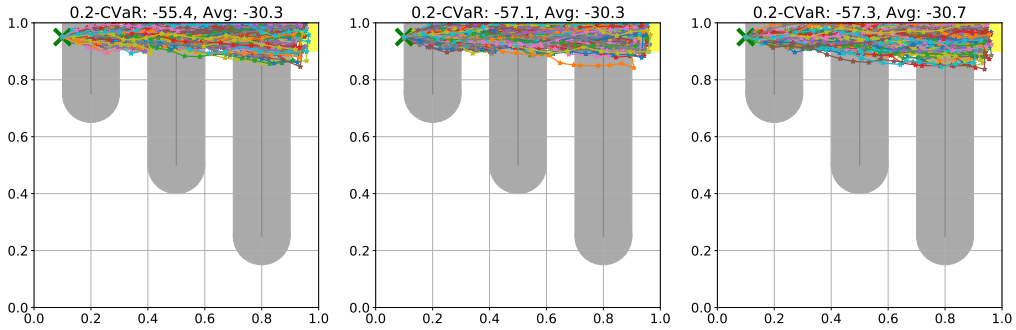



Figure 11: 100 random trajectories from policies optimizing the expected return, for 3 random seeds.

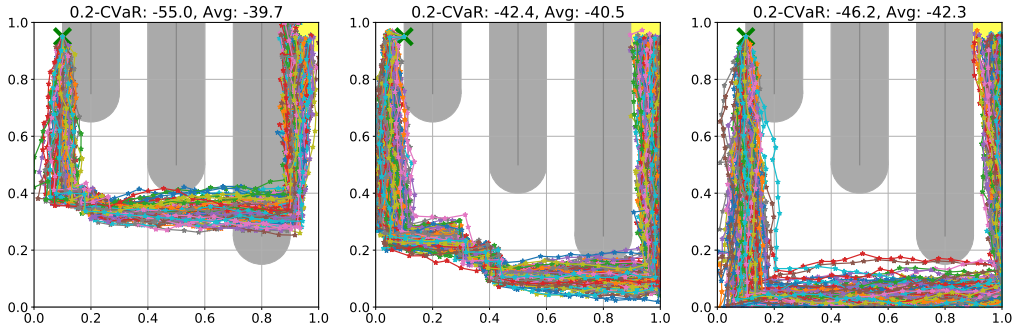



Figure 12: 100 random trajectories from policies optimizing the CVaR using $\widetilde{\mathcal{T}}_{\alpha}^{D}$, for 3 random seeds.

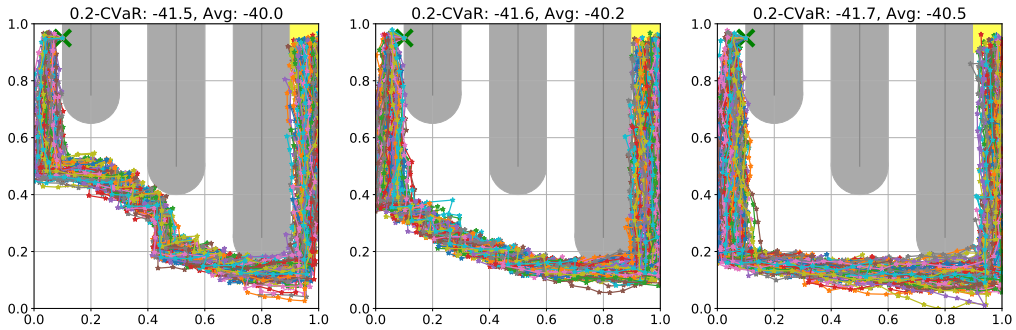



Figure 13: 100 random trajectories from policies optimizing the CVaR using $\widetilde{\mathcal{T}}_{\psi}$, for 3 random seeds.

## D Lunar Lander with Noisy Observation

Lunar Lander is a standard OpenAI gym environment with 8-dimensional continuous state space and 4 discrete actions. To add uncertainty to the environment, we add Gaussian noise to the observations of the lander location. Figure 14 shows the results after running each algorithm for 1 million steps, with $\gamma = 0.99$. We observe that the performance of the proposed approach is more consistent across the 3 random seeds and achieve better 0.25-CVaR overall.

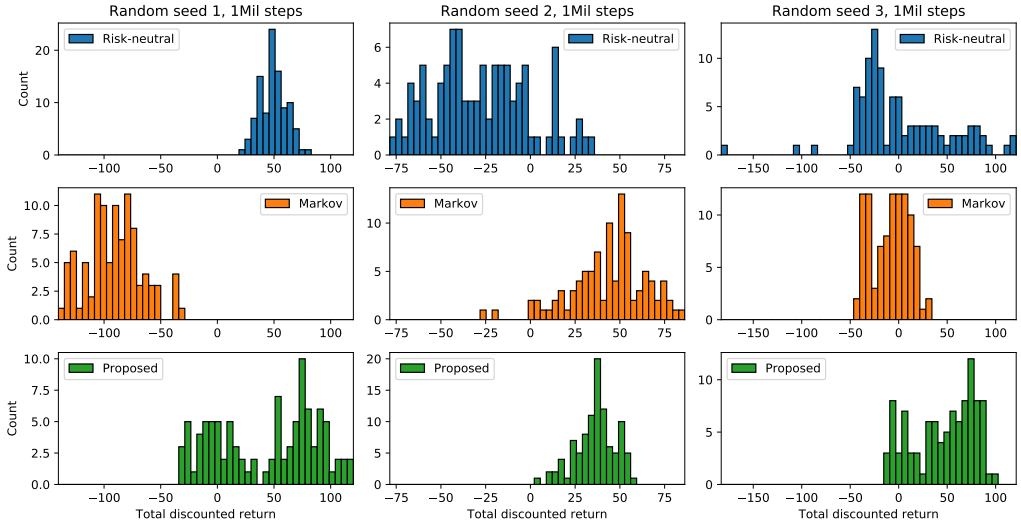



Figure 14: Results on noisy Lunar Lander after training for 1 million steps. Each plot is a histogram of the discounted returns over 100 evaluation episodes.

## E Additional Results on Atari Games

Figure 15 shows results for the game Qbert, using the exact same settings for the game Asterix. Again, we run each algorithm for 30 million steps and evaluate the final policies for 100 episodes each.

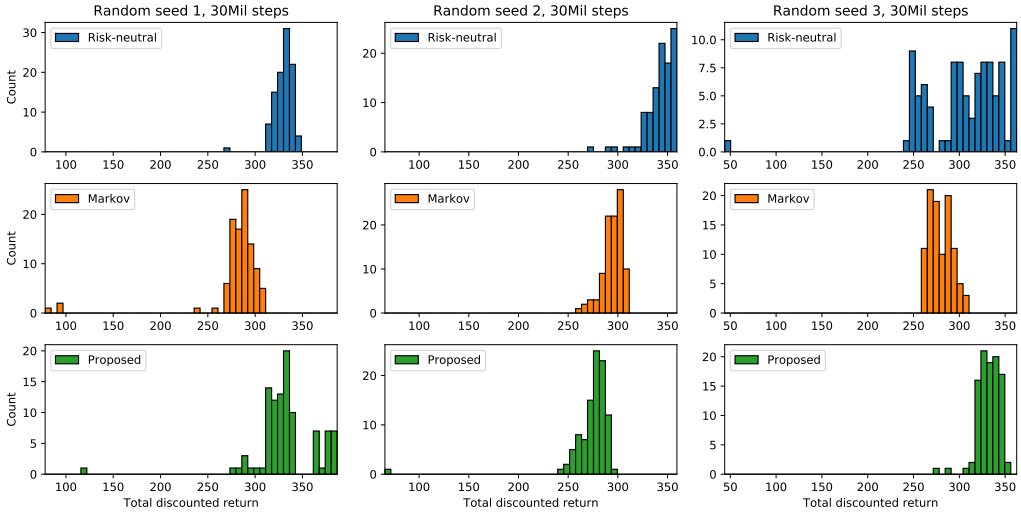



Figure 15: Results on Atari game Qbert after training for 30 million steps. Each plot is a histogram of the discounted returns over 100 evaluation episodes.