Appendices

A Proof of Proposition 2

Proposition 2 (Model Gradient)**.** *Let* ϕ *denote the parameters of a parametric MDP model* $\widehat{T}_\phi$*, and let* V_ϕ^π *denote the value function for policy* π *in* $\widehat{T}_\phi$*. Then:*

$$\nabla_\phi V_\phi^\pi = \mathbb{E}_{s\sim d_\phi^\pi, a\sim\pi, (s',r)\sim\widehat{T}_\phi}\big[(r+\gamma V_\phi^\pi(s'))\cdot\nabla_\phi \log\widehat{T}_\phi(s',r|s,a)\big] \tag{4}$$

Proof: The proof is analogous to the proof of the policy gradient theorem [61, 62]. We start by expressing the value function using Bellman's equation, for some state s:

$$\begin{aligned} V_\phi^\pi(s) &= \sum_a \pi(a|s) Q_\pi(s,a) \\ &= \sum_a \pi(a|s)\sum_{s',r}(r+\gamma V_\phi^\pi(s'))\cdot\widehat{T}_\phi(s',r|s,a). \end{aligned} \tag{10}$$

Applying the product rule:

$$\begin{aligned} \nabla_\phi V_\phi^\pi(s) &= \sum_a \pi(a|s)\sum_{s',r}\Big[(r+\gamma V_\phi^\pi(s'))\cdot\nabla_\phi\widehat{T}_\phi(s',r|s,a) + \widehat{T}_\phi(s',r|s,a)\cdot\nabla_\phi(r+\gamma V_\phi^\pi(s'))\Big] \\ &= \sum_a \pi(a|s)\sum_{s',r}(r+\gamma V_\phi^\pi(s'))\cdot\nabla_\phi\widehat{T}_\phi(s',r|s,a) + \gamma\sum_a\pi(a|s)\sum_{s'}\widehat{T}_\phi(s'|s,a)\cdot\nabla_\phi V_\phi^\pi(s') \end{aligned} \tag{11}$$

Define $\psi(s) := \sum_a \pi(a|s)\sum_{s',r}(r+\gamma V_\phi^\pi(s'))\cdot\nabla_\phi\widehat{T}_\phi(s',r|s,a)$, and additionally define $\rho_\phi^\pi(s\to x,n)$ as the probability of transitioning from state s to x by executing π for n steps in the MDP model defined by $\widehat{T}_\phi$. Using these definitions, we can rewrite the above equation as:

$$\begin{aligned} \nabla_\phi V_\phi^\pi(s) &= \psi(s) + \gamma\sum_{s'}\rho_\phi^\pi(s\to s',1)\cdot\nabla_\phi V_\phi^\pi(s') \\ &= \psi(s) + \gamma\sum_{s'}\rho_\phi^\pi(s\to s',1)\big[\psi(s') + \gamma\sum_{s''}\rho_\phi^\pi(s'\to s'',1)\cdot\nabla_\phi V_\phi^\pi(s'')\big] \\ &= \psi(s) + \gamma\sum_{s'}\rho_\phi^\pi(s\to s',1)\psi(s') + \gamma^2\sum_{s''}\rho_\phi^\pi(s\to s'',2)\cdot\nabla_\phi V_\phi^\pi(s'') \\ &= \sum_{s'}\sum_{t=0}^{\infty}\gamma^t\rho_\phi^\pi(s\to s',t)\psi(s') \qquad \text{(continuing to unroll)} \end{aligned} \tag{12}$$

We have that $V_\phi^\pi = \sum_s \mu_0(s)\cdot V_\phi^\pi(s)$ by definition. Therefore,

$$\begin{aligned} \nabla_\phi V_\phi^\pi &= \nabla_\phi\big(\sum_s \mu_0(s)\cdot V_\phi^\pi(s)\big) \\ &= \sum_s \mu_0(s)\cdot\nabla_\phi V_\phi^\pi(s) \\ &= \sum_s \mu_0(s)\sum_{s'}\sum_{t=0}^{\infty}\gamma^t\rho_\phi^\pi(s\to s',t)\psi(s') \\ &= \sum_s d_\phi^\pi(s)\psi(s) \end{aligned} \tag{13}$$

where $d^\pi_\phi(s)$ is the (improper) discounted state visitation distribution of the policy π in the MDP model $\widehat{T}_\phi$, under initial state distribution μ_0. Finally, we apply the log-derivative trick to arrive at the final result:

$$\begin{aligned}
\nabla_\phi V^\pi_\phi &= \sum_s d^\pi_\phi(s) \sum_a \pi(a|s) \sum_{s',r} (r + \gamma V^\pi_\phi(s')) \cdot \nabla_\phi \widehat{T}_\phi(s', r|s, a) \\
&= \sum_s d^\pi_\phi(s) \sum_a \pi(a|s) \sum_{s',r} \widehat{T}_\phi(s', r|s, a)(r + \gamma V^\pi_\phi(s')) \frac{\nabla_\phi \widehat{T}_\phi(s', r|s, a)}{\widehat{T}_\phi(s', r|s, a)} \\
&= \mathbb{E}_{s \sim d^\pi_\phi, a \sim \pi, (s',r) \sim \widehat{T}_\phi} \big[(r + \gamma V^\pi_\phi(s')) \cdot \nabla_\phi \log \widehat{T}_\phi(s', r|s, a)\big] \quad \square
\end{aligned} \tag{14}$$

A.1 Model Gradient with State-Action Value Baseline

We can subtract $Q^\pi_\phi(s, a)$ as a baseline without biasing the gradient estimate:

$$\nabla_\phi V^\pi_\phi = \mathbb{E}_{s \sim d^\pi_\phi, a \sim \pi, (s',r) \sim \widehat{T}_\phi} \big[(r + \gamma V^\pi_\phi(s') - Q^\pi_\phi(s, a)) \cdot \nabla_\phi \log \widehat{T}_\phi(s', r|s, a)\big] \tag{15}$$

Proof: We can subtract any quantity which does not depend on s' or r without changing the value of the expression because the subtracted quantity is zero. Specifically, for the baseline of $Q^\pi_\phi(s, a)$ we have that:

$$\begin{aligned}
&\mathbb{E}_{s \sim d^\pi_\phi, a \sim \pi, (s',r) \sim \widehat{T}_\phi} \big[Q^\pi_\phi(s, a) \cdot \nabla_\phi \log \widehat{T}_\phi(s', r|s, a)\big] \\
&\qquad = \sum_s d^\pi_\phi(s) \sum_a \pi(a|s) \sum_{s',r} \widehat{T}_\phi(s', r|s, a) \cdot Q(s, a) \cdot \nabla_\phi \log \widehat{T}_\phi(s', r|s, a) \\
&\qquad = \sum_s d^\pi_\phi(s) \sum_a \pi(a|s) \cdot Q(s, a) \cdot \sum_{s',r} \nabla_\phi \widehat{T}_\phi(s', r|s, a) \\
&\qquad = \sum_s d^\pi_\phi(s) \sum_a \pi(a|s) \cdot Q(s, a) \cdot \nabla_\phi \sum_{s',r} \widehat{T}_\phi(s', r|s, a) \\
&\qquad = \sum_s d^\pi_\phi(s) \sum_a \pi(a|s) \cdot Q(s, a) \cdot \nabla_\phi(1) \\
&\qquad = 0
\end{aligned} \tag{16}$$

B Implementation Details

The code for RAMBO is available at github.com/marc-rigter/rambo. Our implementation builds upon the official MOPO codebase.

B.1 Model Training

We represent the model as an ensemble of neural networks that output a Gaussian distribution over the next state and reward given the current state and action:

$$\widehat{T}_\phi(s', r|s, a) = \mathcal{N}(\mu_\phi(s, a), \Sigma_\phi(s, a)).$$

Following previous works [75, 76], during the initial maximum likelihood model training (Line 1 of Algorithm 1) we train an ensemble of 7 such dynamics models and pick the best 5 models based on the validation error on a held-out test set of 1000 transitions from the dataset $\mathcal{D}$. Each model in the ensemble is a 4-layer feedforward neural network with 200 hidden units per layer.

After the maximum likelihood training, RAMBO updates all of these 5 models adversarially according to Algorithm 1. For each model rollout, we randomly pick one of the 5 dynamics models to simulate

Table 4: Base hyperparameter configuration which is shared across all runs of RAMBO.

	Hyperparameter	Value
SAC	critic learning rate	3e-4
	actor learning rate	1e-4
	discount factor (γ)	0.99
	soft update parameter (τ)	5e-3
	target entropy	-dim(A)
	batch size	256
MBPO	no. of model networks	7
	no. of elites	5
	model learning rate	3e-4
	ratio of real data (f)	0.5
	model training batch size	256
	number of iterations (n_{iter})	2000

the rollout. The learning rate for the model training is 3e-4 for both the MLE pretraining, and the adversarial training. The model is trained using the Adam optimiser [26].

For the MLE pretraining, the batch size is 256. For each adversarial update in Equation 9 we sample a batch of 256 transitions from $\mathcal{D}$ to compute the maximum likelihood term, and 256 synthetic transitions for the model gradient term. The hyperparameters used for model training are summarised in Table 4.

B.2 Policy Training

We sample a batch of 256 transitions to train the policy and value function using soft actor-critic (SAC) [19]. We set the ratio of real data to $f = 0.5$ for all datasets, meaning that we sample 50% of the batch transitions from $\mathcal{D}$ and the remaining 50% from $\mathcal{D}_{\widehat{T}_\phi}$. We chose this value because it was used for most datasets by COMBO [75] and we found that it worked well. We represent the Q-networks and the policy as three layer neural networks with 256 hidden units per layer.

For SAC [19] we use automatic entropy tuning, where the entropy target is set to the standard heuristic of $-\text{dim}(A)$. The only hyperparameter that we modify from the standard implementation of SAC is the learning rate for the actor, which we set to 1e-4 as this was reported to work better in the CQL paper [31] which also utilised SAC. For reference, the hyperparameters used for SAC are included in Table 4.

B.3 RAMBO Implementation Details

For each iteration of RAMBO we perform 1000 gradient updates to the actor-critic algorithm, and 1000 adversarial gradient updates to the model. For each run, we perform 2000 iterations. The base hyperparameter configuration for RAMBO that is shared across all runs is shown in Table 4.

The only parameters that we vary between datasets are the length of the synthetic rollouts, k, the weighting of the adversarial term in the model update, λ, and the choice of rollout policy. Details are included in the Hyperparameter Tuning section below.

B.4 Hyperparameter Tuning

We found that the hyperparameters that have the largest influence on the performance of RAMBO are the length of the synthetic rollouts (k), and the weighting of the adversarial term in the model update (λ). For the rollout length, we found that a value of either 2 or 5 worked well across most datasets. This is a slight modification to the values of $k \in \{1, 5\}$ that are typically used in previous model-based offline RL algorithms [8, 75, 76].

To arrive at a suitable value for the adversarial weighting, we used the intuition that we must have $\lambda \ll 1$. This is necessary so that the MLE term dominates in the loss function in Equation 9 for in-distribution samples. This ensures that the model still fits the dataset accurately despite the adversarial

Table 5: Hyperparameters used by RAMBO and RAMBO$^{\text{OFF}}$ for each dataset. k is the rollout length, and λ is the adversary loss weighting. The hyperparameters for RAMBO are those with the best performance from the three configurations tested (see Table 6). The hyperparameters for RAMBO$^{\text{OFF}}$ are those which obtained the lowest value of the offline heuristic in Table 6. As indicated, a random rollout policy was used for some of the AntMaze domains as we found that this performed better.

		RAMBO		RAMBO$^{\text{OFF}}$		
		k	λ	k	λ	Rollout Policy
Random	HalfCheetah	5	0	2	3e-4	Current Policy
	Hopper	2	3e-4	5	0	
	Walker2D	5	0	5	3e-4	
Medium	HalfCheetah	5	3e-4	2	3e-4	
	Hopper	5	3e-4	5	3e-4	
	Walker2D	5	0	5	0	
Medium Replay	HalfCheetah	5	3e-4	5	0	
	Hopper	2	3e-4	2	3e-4	
	Walker2D	5	0	5	0	
Medium Expert	HalfCheetah	5	3e-4	2	3e-4	
	Hopper	5	3e-4	5	3e-4	
	Walker2D	2	3e-4	2	3e-4	
AntMaze	Umaze	5	3e-4	5	3e-4	Current Policy
	Medium-Play	5	0	5	3e-4	Uniform Random
	Large-Play	5	3e-4	5	3e-4	
	Umaze-Diverse	5	0	5	0	
	Medium-Diverse	5	0	2	3e-4	
	Large-Diverse	5	0	5	0	

training. We observed that if λ is set too high, the training can be unstable and the Q-function can become extremely negative. If λ is too small, the adversarial training has no effect as the model is updated too slowly. We found that a value of 3e-4 generally obtained good performance.

For all MuJoCo datasets we performed model rollouts using the current policy, but for some AntMaze datasets we used a random rollout policy as we found this performed better. The rollout policies used are listed in Table 5. For the MuJoCo datasets, we initialised the policy using behaviour cloning (BC) which is common practice in offline RL [25, 74]. Ablation results without the BC initialisation are in Appendix C.4. We did not use the BC initialisation for the AntMaze problems.

For the rollout length and the adversarial weighting, we tested the following three configurations on each dataset: $(k, \lambda) \in \{$(2, 3e-4), (5, 3e-4), (5, 0)$\}$. We included $(k, \lambda) = (5,\ 0)$ as we found that an adversarial weighting of 0 worked well for some datasets. As described in Section 6, to evaluate RAMBO we ran each of the three hyperparameter configurations for five seeds each, and reported the best performance across the three configurations. The results for the best configuration are in Table 1, and the corresponding final hyperparameters chosen can be found in Table 5. The results for each of the three configurations can be found in Table 6.

Offline selection of hyperparameters is an important topic in offline RL [47, 77]. This is because in some applications the selection of hyperparameters based on online performance may be infeasible. Therefore, we present additional results in Table 1 where we select between the three choices of hyperparameters offline using a simple heuristic based on the magnitude and stability of the Q-values during training. After selecting the hyperparameters using the heuristic, we ran a further five seeds using these parameters to produce the final result. We refer to this approach as RAMBO$^{\text{OFF}}$. The hyperparameters used by RAMBO$^{\text{OFF}}$ are also included in Table 5. Details of the heuristic used to select the hyperparameter configuration for RAMBO$^{\text{OFF}}$ can be found in the following subsection.

B.5 Offline Hyperparameter Selection Method

Deep RL algorithms are highly sensitive to the choice of hyperparameters [3]. There is no consensus on how parameters should be selected for offline RL [47]. Some possibilities include a) selection based on online evaluation [8, 9, 25, 39, 72, 76], b) selection based on an offline heuristic [47, 75], and c) methods based on off-policy evaluation [47, 77].

To address applications in which online evaluation is possible, we evaluated RAMBO using approach a) which is the most common practice among model-based offline RL algorithms [8, 25, 37, 39, 76]. To consider situations where online tuning is not possible, we also evaluate using approach b) to select the hyperparameters for each dataset using a simple heuristic in a similar vein to [75]. We refer to this second approach as RAMBO$^{\text{OFF}}$.

To arrive at a heuristic that can be evaluated offline, we use the intuition that the value function should be regularised (i.e. low) as well as stable during training. Therefore, our heuristic makes the final hyperparameter selection from the set of candidate parameters according to: $\min\left(Q_{\text{avg}} + Q_{\text{var}}\right)$, where Q_{avg} is the average Q-value at the end of training, and Q_{var} is the variance of the average Q-value over the final 100 iterations of training.

The values of this heuristic for each of the three configurations that we test are the values in the brackets provided in Table 6. The bolded values in the table indicate the lowest value for the heuristic, which is the parameter configuration chosen for RAMBO$^{\text{OFF}}$. After choosing the hyperparameters in this manner, we *rerun* the algorithm for a further five seeds per dataset. The performance for these additional runs are the results reported in Table 1 for RAMBO$^{\text{OFF}}$.

The results in Table 1 indicate that RAMBO$^{\text{OFF}}$ obtains strong performance compared to existing baselines on the MuJoCo domains. This indicates that it is possible to obtain solid performance with RAMBO by choosing the final hyperparameters according to the magnitude and stability of the Q-values. Unsurprisingly however, RAMBO performs slightly better than RAMBO$^{\text{OFF}}$. This shows that better performance is obtained using online hyperparameter tuning.

B.6 Evaluation Procedure

We report the average undiscounted normalized return averaged over the last 10 iterations of training, with 10 evaluation episodes per iteration. We evaluated the SAC policy by deterministically taking the mean action. The normalization procedure is that proposed by [14], where 100 represents an expert policy and 0 represents a random policy.

B.7 Baselines and Domains

We compare RAMBO against state of the art model-based (COMBO [75], MOReL [25], RepB-SDE [34] and MOPO [76]) and model-free (CQL [31], IQL [28], and TD3+BC [15]) offline RL algorithms on the D4RL benchmarks [14]. The D4RL benchmarks are released with the Apache License 2.0. To facilitate a fair comparison, we provide results for all algorithms for the MuJoCo-v2 D4RL datasets which contain more performant data than v0 [37].

Because the original COMBO, MOPO, and CQL papers use the MuJoCo-v0 datasets we report the results for these algorithms on the MuJoCo-v2 datasets from [67]. The MOReL, IQL, RepB-SDE, and TD3+BC papers include evaluations on the MuJoCo-v2 datasets, so for these algorithms we include the results from the original papers.

For AntMaze-v0, we report the results from the original papers for CQL, IQL, and TD3+BC. For COMBO, MOPO, RepB-SDE, and MOReL, the results are taken from [67]. Following the IQL paper [28], we subtract 1 from the rewards of the AntMaze datasets for our experiments.

B.8 Computational Resources

During our experiments, each run of RAMBO has access to 2 CPUs of an Intel Xeon Platinum 8259CL processor at 3.1GHz, and half of an Nvidia T4 GPU. With this hardware, each run of RAMBO takes 24-30 hours.

For our main evaluation of RAMBO, we ran five seeds for each of three parameter configurations for 18 tasks resulting in a total of 270 runs. This required approximately 150 GPU-days of compute.

C Additional Results

C.1 Single Transition Example

Description In this domain, the state and action spaces are one-dimensional. The agent executes a single action, $a \in [-1, 1]$, from initial state s_0. After executing the action, the agent transitions to s' and receives a reward equal to the successor state, i.e. $r(s') = s'$, and the episode terminates.

The actions in the dataset are sampled uniformly from $a \in [-0.75, 0.7] \cup [-0.15, -0.1] \cup [0.1, 0.15] \cup [0.7, 0.75]$. In the MDP for this domain, the transition distribution for successor states is as follows:

- $s' \sim \mathcal{N}(\mu = 1,\ \sigma = 0.2)$, for $a \in [-0.8,\ -0.65]$.
- $s' \sim \mathcal{N}(\mu = 0.5,\ \sigma = 0.2)$, for $a \in [-0.2,\ -0.05]$.
- $s' \sim \mathcal{N}(\mu = 1.25,\ \sigma = 0.2)$, for $a \in [0.05,\ 0.2]$.
- $s' \sim \mathcal{N}(\mu = 1.5,\ \sigma = 0.2)$, for $a \in [0.65,\ 0.8]$.
- $s' = 0.5$, for all other actions.

The transitions to successor states from the actions in the dataset are illustrated in Figure 1.

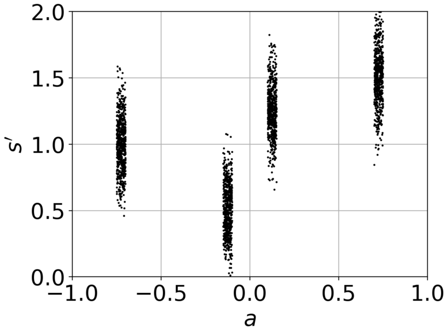


Figure 1: Transition data for the Single Transition Example after executing action a from s_0.

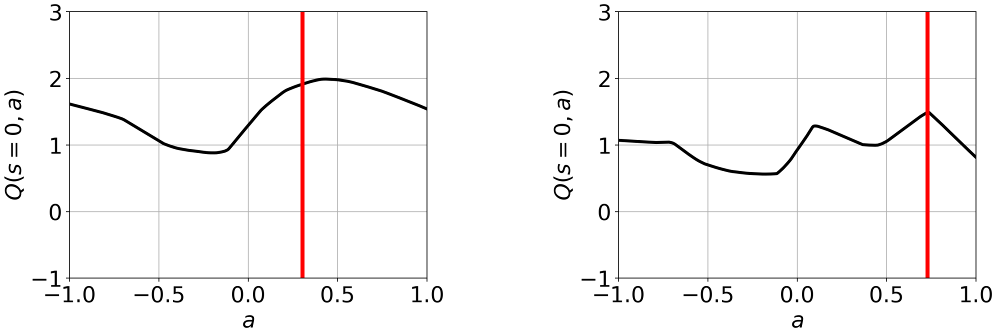
Comparison between RAMBO and COMBO To generate the comparison, we run both algorithms for 50 iterations. To choose the regularisation parameter for COMBO, we sweep over $\beta \in \{0.1, 0.2, 0.5, 1, 5\}$ and choose the parameter with the best performance. Note that 0.5, 1, and 5 are the values used in the original COMBO paper [75], so we consider lower values of regularisation than in the original paper. For RAMBO we use λ =3e-2. We used the implementation of COMBO from github.com/takuseno/d3rlpy.

Figure 2 shows the Q-values produced by RAMBO throughout training. We can see that after 5 iterations, the Q-values for actions which are outside of the dataset are initially *optimistic*, and over-estimate the true values. However, after 50 iterations the Q-values for out of distribution actions have been regularised by RAMBO. The action selected by the policy after 50 iterations is the optimal action in the dataset, $a \in [0.7, 0.75]$. Thus, we see that RAMBO introduces pessimism *gradually* as the adversary is trained to modify the transitions to successor states.

For COMBO, the best performance averaged over 20 seeds was obtained for $\beta = 0.2$, and therefore we report results for this value of the regularisation parameter. Figure 3 illustrates the Q-values produced by COMBO throughout training (with $\beta = 0.2$). We see that at both 5 iterations and 50 iterations, the value estimates for actions outside of the dataset are highly pessimistic. In the run illustrated for COMBO, the action selected by the policy after 50 iterations is $a \in [0.1, 0.15]$, which is not the optimal action in the dataset. Figure 3 shows that the failure to find an optimal action is due to the gradient-based policy optimisation becoming stuck in a local maxima of the Q-function.

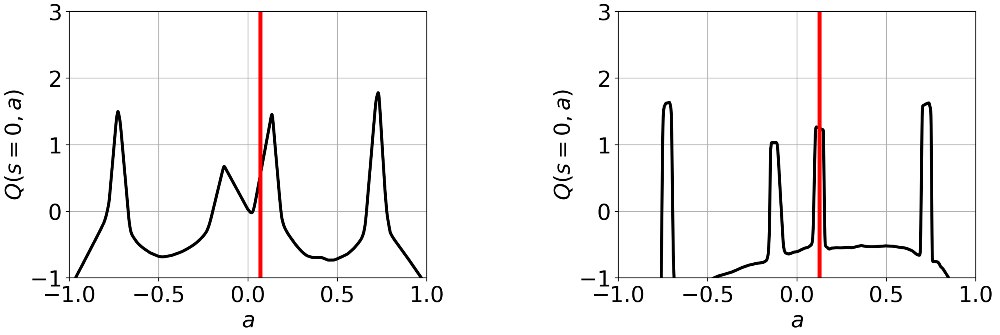
These results highlight a difference in the behaviour of RAMBO compared to COMBO. For RAMBO, pessimism is introduced gradually as the adversary is trained to modify the transitions to successor states. For COMBO, pessimism is introduced at the outset as it is part of the value function update throughout the entirety of training. Additionally, the regularisation of the Q-values for out of distribution actions appears to be less aggressive for RAMBO than for COMBO.

The results averaged over 20 seeds in Table 3 show that RAMBO consistently performs better than COMBO for this problem. This suggests that the gradual introduction of pessimism produced by RAMBO means that the policy optimisation procedure is less likely to get stuck in poor local maxima for this example. The downside of this behaviour is that it may take more iterations for RAMBO to find a performant policy. Modifying other algorithms to gradually introduce pessimism may be an interesting direction for future research.



(a) 5 iterations (b) 50 iterations

Figure 2: Q-values at the initial state during the training of RAMBO on the Single Transition Example. The red line indicates the mean action of the policy.



(a) 5 iterations (b) 50 iterations

Figure 3: Q-values at the initial state during the training of COMBO on the Single Transition Example ($\beta = 0.2$). The red line indicates the mean action of the policy.

C.2 Results for each Hyperparameter Configuration

In Table 6, we report the results for each of the three parameter configurations that we test. The highlighted values indicate the best performance for each dataset, and are the results reported for RAMBO in Table 1. In Table 6 we also report the values of the offline hyperparameter selection heuristic in brackets. The bold values indicate the lowest value of the heuristic, which is the hyperparameter configuration selected for RAMBO$^{\text{OFF}}$.

Table 6: Performance of RAMBO across each of the three configurations of the rollout length, k, and the adversarial weighting, λ, that we tested. Values indicate the average normalized return at the end of training. Each configuration was run for 5 seeds. ± captures the standard deviation over seeds. Highlighted values indicate the best performance for each dataset. The values in brackets indicate the value of the offline tuning heuristic described in Appendix B.5. "–" indicates that the value function diverged.

		$k = 2, \lambda = 3e-4$	$k = 5, \lambda = 3e-4$	$k = 5, \lambda = 0$
Random	HalfCheetah	34.5 ± 3.7 (**348.1**)	38.2 ± 3.9 (408.6)	40.0 ± 2.3 (377.6)
	Hopper	21.6 ± 8.0 (1179.7)	21.3 ± 9.2 (1293.2)	15.1 ± 9.4 (**188.9**)
	Walker2D	–	0.2 ± 0.5 (**1.2e8**)	11.5 ± 10.5 ($2.5e9$)
Medium	HalfCheetah	72.5 ± 4.4 (**671.1**)	77.6 ± 1.5 (717.4)	75.5 ± 6.0 (720.6)
	Hopper	–	92.8 ± 6.0 (**300.4**)	47.5 ± 36.0 (314.1)
	Walker2D	82.2 ± 11.4 (523.6)	76.8 ± 4.8 (2124.2)	86.9 ± 2.7 (**339.4**)
Medium Replay	HalfCheetah	65.4 ± 3.7 (623.5)	68.9 ± 2.3 (647.2)	64.7 ± 3.2 (**608.4**)
	Hopper	96.6 ± 7.0 (**262.4**)	93.5 ± 10.3 (309.1)	36.7 ± 9.5 (263.4)
	Walker2D	6.7 ± 2.5 (2.3e7)	62.1 ± 33.5 (4.3e6)	85.0 ± 15.0 (**259.0**)
Medium Expert	HalfCheetah	78.1 ± 6.6 (**987.1**)	93.7 ± 10.5 (1008.9)	92.9 ± 11.9 (1013.2)
	Hopper	36.4 ± 16.7 (344.1)	83.3 ± 9.1 (**342.8**)	77.6 ± 14.3 (344.4)
	Walker2D	68.3 ± 20.6 (**371.9**)	31.7 ± 49.8 ($2.0e7$)	50.8 ± 57.8 ($6.9e9$)
AntMaze	Umaze	8.8 ± 8.2 (-48.3)	25.0 ± 12.0 (**-54.2**)	3.8 ± 5.8 (-51.5)
	Medium-Play	5.8 ± 6.6 (1421)	8.4 ± 13.5 (**-70.77**)	16.4 ± 17.9 (-68.10)
	Large-Play	0.0 ± 0.0 (-77.9)	0.0 ± 0.0 (**-78.1**)	0.0 ± 0.0 (-77.9)
	Umaze-Diverse	0.0 ± 0.0 (790.8)	0.0 ± 0.0 (68.2)	0.0 ± 0.0 (**-65.4**)
	Medium-Diverse	3.8 ± 1.7 (**-72.4**)	1.6 ± 1.8 (-72.2)	23.2 ± 14.2 (-71.2)
	Large-Diverse	–	0.0 ± 0.0 (6.0e8)	2.4 ± 3.3 (**3.5**e**6**)

C.3 Visualisation of Adversarially Trained Dynamics Model

To visualise the influence of training the transition dynamics model in an adversarial manner as proposed by RAMBO, we consider the following simple example MDP. In the example, the state space (S) and action space (A) are 1-dimensional with, $A = [-1, 1]$. For this example, we assume that the reward function is known and is equal to the current state, i.e. $R(s, a) = s$, meaning that greater values of s have greater expected value. The true transition dynamics to the next state s' depend on the action but are the same regardless of the initial state, s.

In Figure 4 we plot the data present in the offline dataset for this MDP. Note that the actions in the dataset are sampled from a subset of the action space: $a \in [-0.3, 0.3]$. Because greater values of s' correspond to greater expected value, the transition data indicates that the optimal action covered by the dataset is $a = 0.3$.

In Figure 5 we plot the output of running naïve model-based policy optimisation (MBPO) on this illustrative offline RL example. Figure 5a illustrates the MLE transition function used by MBPO. The transition function fits the dataset for $a \in [-0.3, 0.3]$. Outside of the dataset, it predicts that an action of $a \approx 1$ transitions to the best successor state. Figure 5b shows that applying a reinforcement learning algorithm (SAC) to this model results in the value function being over-estimated, and $a \approx 1$ being predicted as the best action. This illustrates that the policy learns to exploit the inaccuracy in the model and choose an out-of-distribution action.

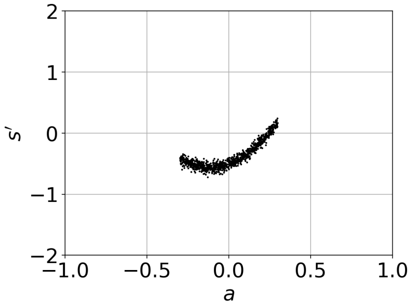
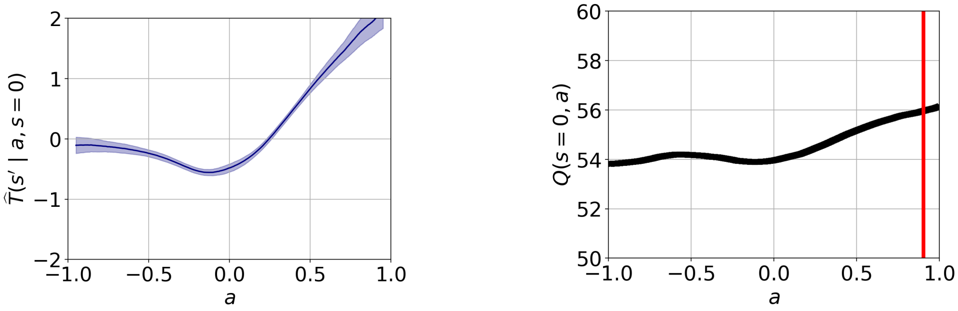


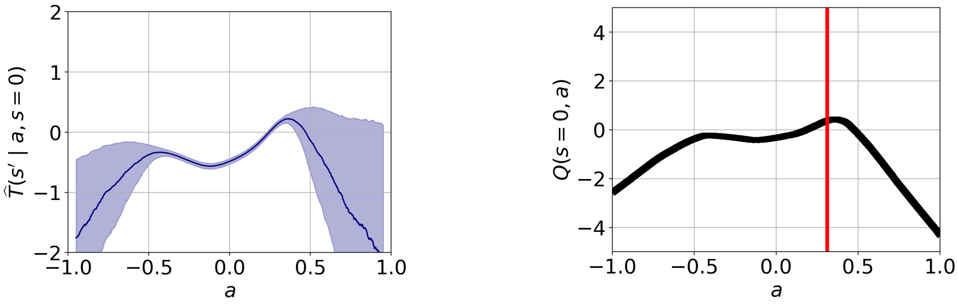
Figure 4: Transition data for illustrative example. The transition function is the same regardless of the initial state. The actions in the dataset are sampled from $a \in [-0.3, 0.3]$.



(a) Maximum likelihood estimate of the transition function, which is used by MBPO. Shaded area indicates ±1 SD of samples generated by the ensemble.

(b) Q-values after running SAC for 15 iterations. The values are significantly over-estimated. The red line indicates the mean action taken by the SAC policy.

Figure 5: Plots generated by running naïve MBPO on the illustrative MDP example.



(a) Model which is adversarially trained using RAMBO. Shaded area indicates ±1 SD of samples generated by the ensemble. The transition function accurately predicts the dataset for in-distribution actions, but predicts transitions to low value states for actions which are out of distribution.

(b) Q-values from SAC critic. The red line indicates the mean action taken by the policy trained using SAC, which is $a \approx 0.3$ (the best in-distribution action). Training on pessimistic synthetic transitions regularises the Q-values for out-of-distribution actions.

Figure 6: Output generated by running RAMBO for 15 iterations on the illustrative MDP example, using an adversary weighting of λ =3e-2.

In Figure 6 we show plots generated after running RAMBO on this example for 15 iterations, using an adversary weighting of $\lambda = 3\text{e-}2$. Figure 6a shows that the transition function still accurately predicts the transitions within the dataset. This is because choosing $\lambda \ll 1$ means that the MLE loss dominates for state-action pairs within the dataset. Due to the adversarial training, for actions $a \notin [-0.3, 0.3]$ which are out of the dataset distribution, the transition function generates pessimistic transitions to low values of s, which have low expected value.

As a result, low values are predicted for out-of-distribution actions, and the SAC agent illustrated in 6b learns to take action $a \approx 0.3$, which is the best action which is within the distribution of the dataset. This demonstrates that by generating pessimistic synthetic transitions for out-of-distribution actions, RAMBO enforces conservatism and prevents the learnt policy from taking state-action pairs which have not been observed in the dataset.

C.4 Ablation of Behaviour Cloning Initialisation

For the MuJoCo locomotion experiments we initialised the policy using behaviour cloning, as noted in Section 6 and Appendix B.4. This is a common initialisation step used in previous works [25, 74]. In Table 7 we present ablation results where the behaviour cloning initialisation is removed, and the policy is initialised randomly. Table 7 indicates that the behaviour cloning initialisation results in a small improvement in the performance of RAMBO.

Table 7: Ablation of RAMBO with no behaviour cloning initialisation on the D4RL benchmark [14]. We report the normalised performance during the last 10 iterations of training averaged over 5 seeds.

		RAMBO	**RAMBO, No BC**
Random	HalfCheetah	40.0 ± 2.3	39.5 ± 3.5
	Hopper	21.6 ± 8.0	25.4 ± 7.5
	Walker2D	11.5 ± 10.5	0.0 ± 0.3
Medium	HalfCheetah	77.6 ± 1.5	77.9 ± 4.0
	Hopper	92.8 ± 6.0	87.0 ± 15.4
	Walker2D	86.9 ± 2.7	84.9 ± 2.6
Medium Replay	HalfCheetah	68.9 ± 2.3	68.7 ± 5.3
	Hopper	96.6 ± 7.0	99.5 ± 4.8
	Walker2D	85.0 ± 15.0	89.2 ± 6.7
Medium Expert	HalfCheetah	93.7 ± 10.5	95.4 ± 5.4
	Hopper	83.3 ± 9.1	88.2 ± 20.5
	Walker2D	68.3 ± 20.6	56.7 ± 39.0
MuJoCo-v2 Total:		826.2 ± 33.8	812.4 ± 47.8