A Environments

In our experiments, we use offline datasets from D4RL [1] for environments from OpenAI gym's [61] MuJoCo continuous control tasks [62]. We look at three locomotion agents shown in Figure 2: HalfCheetah, Hopper, and Walker2d, which are all tasked with moving forward as fast as possible. For each agent, we look at three types of datasets:

1. **Medium**: Approximately 1 million transitions collected from a partially trained SAC agent
2. **Mixed**: Approximately 100000 transitions collected from the entire replay buffer of a SAC agent throughout training
3. **Medium-expert**: Approximately 2 million transitions consisting of half medium samples (collected from a partially trained SAC agent) and half expert samples, which are collected from a fully trained SAC agent.

We don't evaluate on random datasets, which are collected with a random policy for two reasons. First, the actions in these datasets are completely random and behavioral priors are not expected to be helpful since the behaviors are random. Instead we are more interested in evaluating performance on offline datasets with some, even if minimal, structure. Second, we argue that completely random data is a somewhat contrived benchmark. Datasets used to solve real-world problems in robotics, such as autonomous vehicle navigation, locomotion, and manipulation are likely to have some sort of structure.

B Baselines

We compare against several leading model-based and model-free offline RL baselines on the D4RL dataset.

1. **MOPO**: MOPO [21] is an uncertainty-based offline MBRL algorithm. MOPO uses MBPO [31], an off-policy Dyna-style RL algorithm where a replay buffer is populated with synthetic samples from a learned dynamics model and used to train an Soft Actor Critic (SAC) [39] agent. MOPO build on MBPO by penalizing the reward experienced by an agent with a penalty proportional to the prediction uncertainty of the dynamics model. MABE is also built on top of MBPO and thus MOPO is the most directly competing baseline.
2. **MOReL**: MOReL [20] is also an uncertainty-based offline MBRL algorithm. The primary difference between MOReL and MOPO is that MOReL uses an on-policy algorithm, TRPO [63], as its backbone. Otherwise, MOPO and MOReL are similar - both penalize the reward with a term proportional to the forward model uncertainty. The performance differences between MOPO and MOReL on D4RL are mainly due to the performance of the backbone algorithm, SAC and TRPO respectively. SAC outperforms TRPO on the mujoco Cheetah environment while TRPO outperforms TRPO in the Hopper environment, and these differences are also evident in the offline RL results for MOPO and MOReL.
3. **CQL**: Conservative Q-Learning (CQL) [17] is a leading offline model-free baselines. CQL learns Q-functions so that the expected value of a policy under the learned Q-function is a lower-bound of the true policy value. CQL modifies the standard Bellman error with a term that minimizes the Q-function under the policy distribution while maximizing it under the offline data distribution. CQL does not leverage behavioral priors.
4. **BRAC-v**: BRAC-v is another leading model-free RL algorithm that utilizes behavioral priors to learn a conservative policy. BRAC-v is the model-free algorithm most similar to MABE. Like MABE, BRAC-v learns a behavioral prior by fitting a Gaussian distribution to the offline data and regularizing a Gaussian evaluation policy with respect to the behavioral data. Unlike MABE, BRAC-v does not weigh the behavioral prior with the advantage and instead treats all data points equally regardless of the reward achieved.

Additionally, we include comparisons to naive behavior cloning and offline SAC.

Dataset Type	Environment	Target Divergence δ
medium	halfcheetah	100
medium	hopper	0.75
medium	walker2d	1
mixed	halfcheetah	40
mixed	hopper	5
mixed	walker2d	20
medium-replay	halfcheetah	0.1
medium-replay	hopper	0.1
medium-replay	walker2d	0.1

Table 3: Table of target divergences used in MABE per environment

C Experiment Details

C.1 Advantage-Weighted Behavioral Prior

First, to learn the advantages for each datapoint in dataset, we fit a Q-function to the offline dataset. We train until the loss no longer increases any further, then use this Q-function to assign Q-values to each datapoint. We normalize these Q-values by dividing each value by the maximum Q-value assigned to any datapoint.

We train our behavioral prior using a negative log likelihood loss. We weight the loss from each datapoint by the exponentiated normalized Q-values obtained from our learned Q-function. During training, we do a 90-10 train-validation split and stop training when the validation loss stops decreasing.

One note is that for halfcheetah medium-expert, we found that a more simple weighing scheme led to better results. Rather than fitting a Q-function, we weighed datapoints by the final total reward of their trajectory instead. For all other environments, we found that weighing by the Q-function worked better or approximately the same.

C.2 Hyperparameters

Because we built off of MOPO [21], we use the same MOPO-specific hyperparameters for the MOPO hyperparameters of the rollout length h and penalty coefficient λ. We refer you to the MOPO Appendix for these values. We additionally use the MOPO architecture and training method for our dynamics model ensemble. For the dynamics model, we train an ensemble of 7 dynamics models and choose the 5 best models based on their prediction error to use while training our offline SAC agent.

For our policy network, we learn a Gaussian two-head network with 2 hidden layers with 256 hidden units, and two separate linear output layers outputting the mean and log standard deviation of the next action. For our Q networks, we use an architecture of 3 feed-forward layers of 256 hidden units each. Our behavioral prior has the same architecture as our policy network.

Our main hyperparameter for our method is the target KL divergence δ. For our hyperparameter search, we defaulted on a low target divergence for the medium-expert datasets ($\delta = 0.1$), and we performed a grid search for the medium and medium-replay environments, because we found that the different agents required different target divergences based on their dataset composition. The full list of target divergences used can be found in Table C.2

D Compute Resources and Assets Used

Compute Resources Experiments for our main suite of results were run on GPUs using a machine with eight Quadro RTX 6000. However, only one GPU is required for four concurrent experiments, so our main experiments used approximately 1080 GPU hours (including all seeds).

Assets Used In this work we used the D4RL Offline RL Benchmark for evaluation [1] which has an Apache License 2.0. We build our code off of logic from MOPO [21], which is distributed under a MIT License. We built our final codebase off of a PyTorch replication codebase of MBPO [64]. From this codebase, we ported over MOPO logic. Additionally, we train our dynamics models in the

MOPO official codebase for fair comparison against MOPO. For our baselines, we copy MOPO, BC, SAC, and BRAC-v baselines from [21], MOReL baselines from [20], and CQL baselines from [17].