# Appendix for *Implicit Behavioral Cloning*

## Contents

# A Contributions Statement

Due to space constraints we did not include a comprehensive contributions statement in the main manuscript, but include one here for clarity:

1. We present Implicit Behavioral Cloning (Implicit BC), which is a novel, simple method for imitation learning in which behavioral cloning is cast as a conditional energy-based modeling (EBM) problem, and inference is performed via sampling-based or gradient-based optimization.
2. We validate Implicit BC in real-world robot experiments, in which we demonstrate physical robots performing several end-to-end, contact-rich pushing tasks (including precision insertion, and multi-item sorting) driven with only images as input, and only human demonstrations provided as training data. Implicit BC performs significantly better than our explicit BC baseline across all real-world tasks, including an *order-of-magnitude* increase in performance on the precision insertion task. On the sorting task, the models are shown to be capable of solving an up-to-60-second horizon for a contact-rich, combinatorial task with complex multi-object collisions.
3. We present extensive simulation experiments comparing Implicit BC to both comparable explicit models from the same codebase, and also author-reported quantitative results on the human-expert tasks from the standard D4RL benchmark. We find both our explicit BC and implicit BC models provide competitive or state-of-the-art performance on D4RL tasks with human-provided demonstrations, despite using no reward information. Averaged across all tasks, we find implicit BC outperforms our own best explicit BC models.
4. We analyze the nature of implicit models in simple 1D-1D examples, and we highlight aspects of implicit models that we believe are not known to the generative modeling community, including their behavior (i) at discontinuities and (ii) in extrapolation.
5. We provide theoretical insight into implicit models, including proofs of their (i) representational abilities (Thm. 1), and (ii) approximation abilities (Thm. 2), which are shown to be distinct from continuous explicit models in their ability to handle discontinuities and set-valued functions.

# B Energy-Based Model Training and Implicit Inference Details

Our results critically depend on energy-based model (EBM) training, but we do not consider the specific methods we use to be our main contributions (see Sec. A for a list). That said, after considerable experience training conditional EBMs on both simple function-fitting tasks, and on policy learning tasks, we believe it is useful to the research community to describe method specifics in detail. Our goal is to emphasize simplicity when possible, in order to encourage more folks to use implicit energy-based regression rather than explicit regression. We first review our approach using derivative-free optimization, then our autoregressive version, and then our approach using Langevin gradient-based sampling. For each, we discuss (i) how to *train* the models, and (ii) how to perform *inference* with the models. For a more comprehensive overview of training EBMs, see [1]. Note we will release code as well for training and inference.

For all methods, to compute $\mathbf{y}_{\text{min}}$ and $\mathbf{y}_{\text{max}}$ we (1) take the per-dimension min and max over the training data, (2) add a small buffer, typically $0.05(\mathbf{y}_{\text{max}} - \mathbf{y}_{\text{min}})$ on each side, and then (3) clip these min and max values to the environments' allowed min/max values. For agents that do not use the full range of the environments' allowed values for a given dimension, this enables more precision on that action dimension. Also all methods use Adam optimizer with default $\beta_1 = 0.9$, $\beta_2 = 0.999$ values.

## B.1 Method with Derivative-Free Optimization.

For training, this method is very simple. For counter-examples we draw from the uniform random distribution: $\tilde{\mathbf{y}} \sim \mathcal{U}(\mathbf{y}_{\text{min}}, \mathbf{y}_{\text{max}})$, where $\mathbf{y}_{\text{min}}, \mathbf{y}_{\text{max}} \in \mathbb{R}^m$. Training consists of drawing batches of data, sampling counterexamples for each sample in each batch, and applying $\mathcal{L}_{\text{InfoNCE}}$ (Sec. 2). We typically use a batch size of 512, with 256 counter-examples per sample in the batch. All $\{\mathbf{x}\}$ and $\{\mathbf{y}\}$ (i.e. $\mathbf{o}$ and $\mathbf{a}$ for observations and actions), in the training dataset are normalized to per-dimension zero-mean, unit variance. We use typically a $1e-3$ initial learning rate and an exponential decay, 0.99 decay each 100 steps. We find that regularizing the models with Dropout does not help performance, perhaps because the stochastic training process (counter-example sampling in each training step) self-regularizes the models.

Given a trained energy model $E_\theta(\mathbf{x},\mathbf{y})$, we use the following derivative-free optimization algorithm to perform inference:

**Algorithm 1:** Derivative-Free Optimizer

**Result:** $\hat{\mathbf{y}}$

Initialize: $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} \sim \mathcal{U}(\mathbf{y}_{\text{min}},\mathbf{y}_{\text{max}})$, $\sigma = \sigma_{\text{init}}$ ;

**for** *iter in 1, 2, ..., $N_{iters}$* **do**

  $\{E_i\}_{i=1}^{N_{\text{samples}}} = \{E_\theta(\mathbf{x},\tilde{\mathbf{y}}_i)\}_i^{N_{\text{samples}}}$ (compute energies);

  $\{\tilde{p}_i\}_{i=1}^{N_{\text{samples}}} = \{\frac{e^{-E_i}}{\sum_{j=1}^{N_{\text{samples}}} e^{-E_j}}\}$ (softmax);

  **if** *iter* < $N_{iters}$ **then**

    $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} \leftarrow \sim \text{Multinomial}(N_{\text{samples}}, \{\tilde{p}_i\}_{i=1}^{N_{\text{samples}}}, \{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}})$ (resample with replacement);

    $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} \leftarrow \{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} + \sim \mathcal{N}(0,\sigma)$ (add noise);

    $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} = \text{clip}(\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}}, \mathbf{y}_{\text{min}}, \mathbf{y}_{\text{max}})$ (clip to $\mathbf{y}$ bounds) ;

    $\sigma \leftarrow K\sigma$ (shrink sampling scale) ;

$\hat{\mathbf{y}} = \text{argmax}(\{\tilde{p}_i\}, \{\tilde{\mathbf{y}}_i\})$

Where $\text{Multinomial}(N_{\text{samples}}, \{\tilde{p}_i\}_{i=1}^{N_{\text{samples}}}, \{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}})$ refers to sampling $N_{\text{samples}}$ times from the multinomial distribution with probabilities $\{\tilde{p}_i\}_{i=1}^{N_{\text{samples}}}$ returning associated elements $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}}$. For simplicity the noise is written as being drawn from $\sim\mathcal{N}(0,\sigma)$, but this should be an $N_{\text{samples}}$-dimensional vector with an independent Gaussian noise sample for each element. This algorithm is very similar to the Cross Entropy Method [2], but has a few differences: (i) our algorithm does not use a fixed number of elites, (ii) re-sampling with replacement, and (iii) we shrink the sampling variance via a prescribed schedule rather than computing empirical variances. We typically use $\sigma_{\text{init}} = 0.33$, $K = 0.5$, $N_{\text{iters}} = 3$, $N_{\text{samples}} = 16{,}384$, unless otherwise noted.

While the above method works great for up to $\mathbf{y}$ of 5 dimensions or less (Sec. B.4), we look at both autoregressive and gradient-based methods for scaling to higher dimensions.

### B.2 Method with Autoregressive Derivative-Free Optimization.

In the autoregressive version we interleave training and inference with $m$ models, for $\mathbf{y} \in \mathbb{R}^m$, i.e. one model $E_\theta^j(\mathbf{x},\mathbf{y}^{:j})$ for each dimension $j = 1,2,...,m$. Model $E_\theta^j(\mathbf{x},\mathbf{y}^{:j})$ takes in all $\mathbf{y}$ dimensions up to $j$. This isolates sampling to one degree of freedom at a time, and enables scaling to higher dimensional action spaces. For more on autoregressive energy models, see [3].

**Algorithm 2:** Autoregressive Derivative-Free Optimizer

**Result:** $\hat{\mathbf{y}}$

Initialize: $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} \sim \mathcal{U}(\mathbf{y}_{\text{min}},\mathbf{y}_{\text{max}})$, $\sigma = \sigma_{\text{init}}$ ;

**for** *iter in 1, 2, ..., $N_{iters}$* **do**

  **for** *$j$ in 0, 1, ..., $m$* **do**

    $\{E_i\}_{i=1}^{N_{\text{samples}}} = \{E_\theta^j(\mathbf{x},\tilde{\mathbf{y}}_i^{:j})\}_i^{N_{\text{samples}}}$ (compute energies);

    $\{\tilde{p}_i\}_{i=1}^{N_{\text{samples}}} = \{\frac{e^{-E_i}}{\sum_{j=1}^{N_{\text{samples}}} e^{-E_j}}\}$ (softmax);

    $\rightarrow$ *if training, apply $\mathcal{L}_{InfoNCE}$ and update parameters of $E_\theta^j$*

  **if** *iter* < $N_{iters}$ **then**

    $\{\tilde{\mathbf{y}}_i^{:j}\}_{i=1}^{N_{\text{samples}}} \leftarrow \sim \text{Multinomial}(N_{\text{samples}}, \{\tilde{p}_i\}_{i=1}^{N_{\text{samples}}}, \{\tilde{\mathbf{y}}_i^{:j}\}_{i=1}^{N_{\text{samples}}})$ (resample with replacement);

    $\{\tilde{\mathbf{y}}_i^{j}\}_{i=1}^{N_{\text{samples}}} \leftarrow \{\tilde{\mathbf{y}}_i^{j}\}_{i=1}^{N_{\text{samples}}} + \sim \mathcal{N}(0,\sigma)$ (add noise);

    $\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}} = \text{clip}(\{\tilde{\mathbf{y}}_i\}_{i=1}^{N_{\text{samples}}}, \mathbf{y}_{\text{min}}, \mathbf{y}_{\text{max}})$ (clip to $\mathbf{y}$ bounds) ;

    $\sigma \leftarrow K\sigma$ (shrink sampling scale) ;

$\hat{\mathbf{y}} = \text{argmax}(\{\tilde{p}_i\}, \{\tilde{\mathbf{y}}_i\})$

### B.3 Method with Gradient-based, Langevin MCMC

For gradient-based MCMC (Markov Chain Monte Carlo) training we use the approach described in [4, 5] which uses stochastic gradient Langevin dynamics (SGLD) [6]:

$$ {}^{k}\tilde{\mathbf{y}}_i^j = {}^{k-1}\tilde{\mathbf{y}}_i^j - \lambda\left(\frac{1}{2}\nabla_{\mathbf{y}} E_\theta(\mathbf{x}_i, {}^{k-1}\tilde{\mathbf{y}}_i^j) + \omega^k\right), \quad \omega^k \sim \mathcal{N}(0, \sigma) $$

Note that in the conditional case, $\nabla$ is respect to only $\mathbf{y}$, and not $\mathbf{x}$. As in [4, 5] we initialize $\{{}^{0}\tilde{\mathbf{y}}\}$ from the uniform distribution, similar to Sec. B.1, but then optimize these contrastive samples with MCMC. For each $N_{\text{neg}}$, we run $N_{\text{MCMC}}$ steps of the MCMC chain. As recommended in [7] we use a polynomially-decaying schedule for the step-size $\lambda$. Note backpropagation is not performed backwards through the chain, but rather a `stop_gradient()` is used after implicitly generating the samples [4]. Also as in [4] we clip gradient steps, choosing to clip the full $\Delta\mathbf{y}$ value, i.e. after the gradient and noise have been combined. Additionally for inference we run the Langevin MCMC chain a second time, giving twice as many inference Langevin steps as were used during training. Also, for Langevin, all $\{\mathbf{y}\}$ (i.e. $\mathbf{a}$ for actions), in the training dataset are normalized per-dimension to span the range $[\mathbf{y}_{\text{min}} = -1, \mathbf{y}_{\text{max}} = 1]$.

#### B.3.1 Gradient Penalty

For additional stability of training, we use both spectral normalization [8] as in [4], and also add gradient penalties. Gradient penalties are well known in the GAN community, and the form of our gradient penalty is inspired by [9]:

$$ \mathcal{L}_{grad} = \sum_{i=1}^{N} \sum_{j=1}^{N_{\text{neg}}} \sum_{k=\{\cdot\}} \max\left(0\ , (||\nabla_{\mathbf{y}} E_\theta(\mathbf{x}_i, {}^{k}\tilde{\mathbf{y}}_i^j)||_\infty - M)\right)^2 $$

Where the sums over $i$, $j$, $k$, represent respectively the sum over training samples, counter-examples per each data sample, and some subset of iterative chain samples for which we find it is sufficient to use only the final step, $k = \{N_{\text{MCMC}}\}$. $M$ controls the scale of the gradient relative to the noise $\omega$ in SGLD. If $M$ is too large, then the noise in SGLD has little effect; if $M$ is too small, then the noise overpowers the gradient. Empirically we find $M = 1$ is a good setting. On each step of training, the gradient penalty loss is simply added to the InfoNCE loss, i.e. $\mathcal{L} = \mathcal{L}_{\text{grad}} + \mathcal{L}_{\text{InfoNCE}}$. Lastly, we note there are other approaches for improving stability of Langevin-based training, such as loss functions with entropy regularization [10].

To aid intuition on why constraints on the gradients $\nabla E(\cdot)$ are allowable restrictions for the model, Corollary 1.1 shows that the energy model is capable of having an arbitrary Lipschitz constant.

### B.4 Comparison of EBM Variants

A key comparison between these methods is the tradeoff of simplicity for higher-dimensional action spaces. As shown in Fig. 1, with only 2,000 demonstrations in the $N$-D particle environment, the joint-dimensions-optimized derivative-free version (Sec. B.1) fails to solve the environment past $N = 5$ dimensions, due to the curse of dimensionality and its naive sampling. Both the autoregressive (Sec. B.2) and Langevin (Sec. B.3) versions are able to solve the environment reliably up to 16 dimensions, and with nonzero success at 32 dimensions. The autoregressive version requires no new gradient stabilization, and can use only the same loss function, $\mathcal{L}_{\text{InfoNCE}}$, but is memory-intensive, requiring $N$ separate models for $N$ dimensions. The Langevin version scales to high dimensions with only one model, but requires gradient stabilization. For more on autoregressive and Langevin generative EBMs, see [3] and [4, 10]. Which variant is used for each of our evaluation tasks is enumerated in Section D.

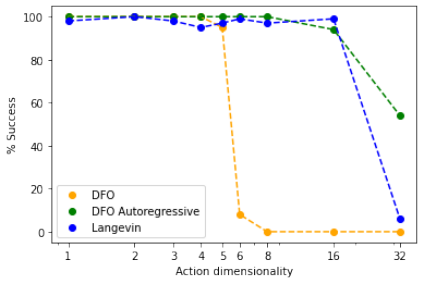



*Figure 1.* Comparison of used EBM methods on the $N$-D particle environment, showing methods using DFO (derivative-free optimization, Sec. B.1), autoregressive DFO (Sec. B.2), or Langevin dynamics (Sec. B.3).

# C Additional Experimental Details and Analysis

## C.1 Per-Task Summary of # Demonstrations and Environment Dimensionalities

In this section, with the table below, we highlight key aspects of the different evaluated policy learning experimental tasks, specifically the # of demonstrations for each task and the dimensionalities of the environments (comprised of the observation spaces, state spaces, and action spaces). As is highlighted in the table, the various tasks cover a wide set of challenges, including: low-data-regime tasks, and tasks with high observation, state, and/or action dimensionalities.

| | | Demonstrations | Dimensionalities | | | | |
|---|---|---|---|---|---|---|---|
| *Domain* | *Task Name* | # | *Observations* | *States* | *Actions* | Results Shown In | Comment |
| D4RL Human-Experts | kitchen-complete | **19** | **60** | **60** | **9** | Table 2 | |
| | kitchen-partial | 601 | **60** | **60** | **9** | | |
| | kitchen-mixed | 601 | **60** | **60** | **9** | | |
| | pen-human | **50** | **45** | **45** | **24** | | |
| | hammer-human | **25** | **46** | **46** | **26** | | |
| | door-human | **25** | **39** | **39** | **28** | | |
| | relocate-human | **25** | **39** | **39** | **30** | | |
| Particle Integrator | "1D"-Particle | 2,000 | 4 | 4 | 1 | Figure 6 | |
| | "2D"-Particle | 2,000 | 8 | 8 | 2 | | |
| | "3D"-Particle | 2,000 | 12 | 12 | 3 | | |
| | "4D"-Particle | 2,000 | 16 | 16 | 4 | | |
| | "5D"-Particle | 2,000 | 20 | 20 | 5 | | |
| | "6D"-Particle | 2,000 | 24 | 24 | 6 | | |
| | "8D"-Particle | 2,000 | **32** | **32** | 8 | | |
| | "16D"-Particle | 2,000 | **64** | **64** | **16** | | |
| | "32D"-Particle | 2,000 | **128** | **128** | **32** | | |
| Simulated Pushing | Single Target, States | 2,000 | 10 | 10 | 2 | Table 3 | |
| | Multi Target, States | 2,000 | 13 | 13 | 2 | | |
| | Single Target, Pixels | 2,000 | **129,600** | 10 | 2 | | 180x240x3 image |
| Planar Sweeping | Image input | **50** | **27,648** | **203** | 3 | Table 4 | 96x96x3 image |
| | State input | **50** | **203** | **203** | 3 | | |
| Bi-Manual Sweeping | Image-and-state input | 1,000 | **27,660** | **372** | **12** | Table 5 | 96x96x3 image |
| Real Robot | Push-Red-Then-Green | **95** | **32,400** | 8 | 2 | Table 6 | 90x120x3 image. |
| | Push-Red/Green-Multimodal | 410 | **32,400** | 8 | 2 | | |
| | Insert-Blue | 223 | **32,400** | 8 | 2 | | |
| | Sort-Blue-From-Yellow | 502 | **32,400** | **26** | 2 | | |

*Table 1.* Summary of the # demonstrations and *observation/state/action*-dimensionalities for each of the environments used in policy learning experiments. Highlighted in color are **(red), low-data-regime tasks** with # demos under 100, **(green), high observation dimensionality** above 25, **(blue), high state dimensionality** above 25, and **(cyan), high action dimensionality** at or above 9.

## C.2 Training and Inference Times, Implicit vs. Explicit Comparison

**D4RL Train+Eval Times.** Table 2 compares example training + evaluation times for the chosen best-performing models on the D4RL tasks. We report both the training steps/second, and then also the full time for running an experiment, which comprises training to 100k steps with intermittently evaluating 100 episodes every 10k steps.

| | **Implicit BC** | **Explicit BC** | Comment |
|---|---|---|---|
| Configuration<br>*Summary:* | As in Section D.1<br>512 batch size<br>512x8 MLP<br>100 Langevin iterations<br>8 counter examples | As in Section D.1<br>512 batch size<br>2048x8 MLP | |
| Device | TPUv3 | TPUv3 | |
| Task | door-human-v0 | door-human-v0 | |
| Training rate (steps/sec) | 17.9 | 101.3 | |
| Total train + eval time (hrs) | 3.4 | 0.66 | 100k train steps, 100 evals every 10k steps |

*Table 2.* Comparison of training+evaluation times for implicit vs. explicit models on an example D4RL task.

As is shown in Table 2, the best-performing implicit models, which are 100-iteration Langevin models, take 5.6x the train+eval time compared to the best-performing explicit models. Note that even the 3.4-hour full train+eval time for the implicit model is considerably faster than what has been reported [11] for completing a train+eval on a comparable D4RL task for CQL: 16.3 hours.

**Real-World Image-based Train and Inference Times.** The following compares relevant training and inference times for our real-world tasks. In contrast to the D4RL scenario discussed above, in this scenario (a) there are large image observations to process, and (b) there are no simulated evaluations run during

training. We report the training steps/sec rate, as well as the total train time, which is performed on a server of 8 GPUs. Once trained, the model is then deployed on a single-GPU machine, for which we report the inference times.

| | **Implicit BC** | **Explicit BC** | Comment |
|---|---|---|---|
| Configuration<br>*Summary:* | As in Section D.5<br>128 batch size<br>90x120 images<br>4-layer ConvMaxPool<br>1024x4 MLP<br>256 counter examples | As in Section D.5<br>128 batch size<br>90x120 images<br>4-layer ConvMaxPool<br>1024x4 MLP | |
| Training Device | 8x V100 GPU | 8x V100 GPU | |
| Task | Push-Red-Then-Green | Push-Red-Then-Green | |
| Training rate (steps/sec)<br>Total train time (hrs) | 4.7<br>5.8 | 5.5<br>5.0 | <br>100k train steps |
| Inference Device | 1x RTX 2080 Ti GPU | 1x RTX 2080 Ti GPU | |
| Inference parameters | 1024 samples<br>3 dfo iterations | | |
| Inference time (ms) | 7.22 | 3.49 | |

*Table 3.* Example comparison of training and inference times for implicit vs. explicit models used for a Real Robot task.

Table 3 shows that for these visual models, the training times are reasonably comparable for the implicit and explicit models – 5.0 and 5.8 hours respectively. Compared to the previous D4RL scenario, this can be explained because the training time is mostly dominated by visual processing. As the implicit models use late fusion (Sec. E), the visual processing time is identical to the explicit models. For inference, the chosen implicit models show a modest increase in inference time, up to 7.22 milliseconds (ms) from 3.49 ms for the explicit model. This can be attributed to time spent on the iterative derivative-free optimization. Note that the inference time of the implicit model can be adjusted by adding/decreasing the number of samples and iterations. For example, using the same trained model but increasing the samples from 1024 to 2048 causes the inference time to increase to 9.25 ms.

## C.3 Additional Real-World Experimental Details

### C.3.1 Robot Hardware Configuration, Workspace, and Objects

Our real-world experiments make use of a UFACTORY xArm6 robot arm with all state logged at 100 Hz. Observations are recorded from an Intel RealSense D415 camera, using RGB-only images at 640x360 resolution, logged at 30 Hz. The cylindrical end-effector is made from a 6 inch long plastic PVC pipe sourced from McMaster-Carr (9173K515). The work surface is 24 x 18 inch smooth wood cutting board. The manipulated objects are from the Play22 Baby Blocks Shape Sorter toy kit (Play22). The targets for the tasks were constructed by hand out of wood and spraypainted black. All demonstrations were provided by a mouse-based interface for providing real-time demonstrations.

The 6DOF robot is constrained to move in a 2D plane above the table. This aids in safety of the robot during operation, since it is constrained to not collide with the table and cannot provide normal forces against objects down into the table either.

### C.3.2 Robot Policy and Controller

The learned visual-feedback policy operates at 5 Hz. On a GTX 2080 Ti GPU, the implicit models (configuration in Sec. D.5) complete inference in under 10 ms (see Sec. C.2), and so could be run faster than 5 Hz, but we find 5 Hz to be sufficient. The learned action space is a delta Cartesian setpoint, from the previous setpoint to the new one. The setpoints are linearly interpolated from their 5 Hz rate to be 100 Hz setpoints to our joint level controller. The joint level controller uses PyBullet [12] for inverse kinematics, and sends joint positions to the xArm6 robot at 100 Hz.

## C.4 Nearest-Neighbor Baseline

This baseline memorizes all training data, and performs inference by looking up the closest observation in the training set and returning the corresponding action. Specifically, given a finite training dataset of pairs $\{(\mathbf{x},\mathbf{y})\}_i$, denote the inputs as $X = \{\mathbf{x}\}_i$ and outputs $Y = \{\mathbf{y}\}_i$, preserving the ordering in both $X$ and $Y$.

Given some new observation $\mathbf{x}'$, the Nearest-Neighbor model, $N(\cdot)$, computes:

$$N(\mathbf{x}') = Y[\operatorname{argmin}_i |\mathbf{x}' - X[i]|]$$

for some norm $|\cdot|$. Specifically we used L2 norm. We experimented with normalizing all observations per-dimension to be unit-variance, but did not find this to improve results. For environments with state-only observations (no images), we can compute this exactly and quickly all in processor memory, but for the image-observation Simulated Pushing task we tested, the dataset did not fit in memory. Accordingly, we used a random linear projection, which is known to be a viable method for nearest-neighbor lookup of image data [13], from the observation space to a 128-dimensional vector. We then stored all these 128-dimensional vectors in memory, and used these for Nearest-Neighbor lookups.

### C.5 $N$-D Particle Environment Description

In this environment, the agent (i.e., particle) moves from its current configuration $q \in \mathbb{R}^N$ to a goal configuration $g_0 \in \mathbb{R}^N$, followed by a second goal configuration $g_1 \in \mathbb{R}^N$. Given its position $q$ and velocity $\dot{q}$, its action is the target position $\hat{q} \in \mathbb{R}^N$ applied to a PD controller which computes acceleration $\ddot{q}$ according to: $\ddot{q} = k_p(\hat{q} - q) + k_d(\hat{\dot{q}} - \dot{q})$ where target velocity $\hat{\dot{q}} = 0$, and $k_p$ and $k_d$ are environment-fixed constant gains. Initial and goal particle configurations are randomized, for each dimension, in the range $[0,1]$ for each episode, and differ between training and testing. To generate demonstrations, a scripted policy returns actions $q = g_0$ until the agent falls within a radius $r$ of $g_0$, then returns actions $q = g_1$ until the agent falls within a radius $r$ of $g_1$. Agent state and goal positions are used as input to the policy, which is trained to imitate the behavior of the scripted policy and tested on its capacity to generalize to new goal configurations. This task can be thought of as modeling an $N$-dimensional step function while dealing with compounding errors. The mode switch between goals presents a discontinuity that needs to be learned.

### C.6 Analysis: Training Data Sparsity in the $N$-D Particle Tasks

To complement other analyses on generalization, sample complexity, and interpolation/extrapolation, we analyze in Fig. 2 another notion of generalization: training data sparsity. In the $N$-D particle experiments, as we increase N but hold the number of demonstrations constant, the training data effectively becomes much sparser over the observation space. New test-time environments for evaluation are accordingly, as $N$ increases, on average farther and farther away from the training set. This helps explain how the Nearest-Neighbor baseline cannot solve this task well past 1D, since memorizing the training data is insufficient, and to succeed in a higher-dimensional environment a model must generalize. This analysis complements our simple 1D->1D figures on extrapolation/interpolation (Fig. 2 and Fig. 3 in the main paper) and our visual generalization and sample complexity analysis (Fig. 4 in the main paper).

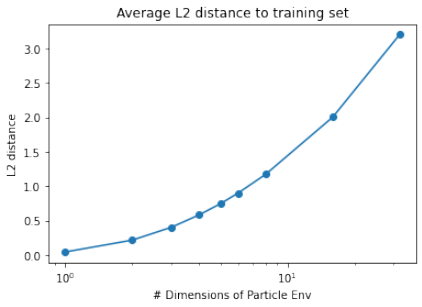



*Figure 2.* Depiction of *training data sparsity* on the $N$-D particle tasks, as $N$ is varied. Shown, for each $N$-D variant of the task, is the average distance of an evaluation episode initialization to the training set of 2,000 demonstrations.

### C.7 Additional D4RL tasks

In the main paper we focused on the human-expert tasks from D4RL, but here provide results on additional D4RL tasks as well. Note that the other tasks shown, except for 'random', use a reinforcement-learning-trained agent for the task, and this reinforcement-learning agent itself has a policy that is a uni-modal continuous, explicit function approximator, and it was optimized as such. Additionally, as expected, supervised imitation learning methods, which do not make use of the additional reward information from the provided demonstrations, perform comparatively worse on tasks with sub-optimal demonstrations. This is true of all tasks with "*medium*" and "*random" in their task name. Additionally, as stated in Section D, we choose the EBM hyperparameters to maximize performance on the human-expert based environments ("Franka" and "Adroit" tasks) at the expense of lower performance on the "Gym"-mujoco tasks. However, for fair comparison with other methods, and according to the standard D4RL evaluation protocol, a single set of hyperparameters was used for all tasks rather than presenting results that maximize each environment.

| Method | | Baselines | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|
| | | BC (from CQL) | CQL [14] | S4RL [15] | *Explicit* BC (MSE) | *Implicit* BC (EBM) | *Explicit* BC (MSE) w/ RWR [16] | *Implicit* BC (EBM) w/ RWR [16] |
| Uses data | | $(\mathbf{o},\mathbf{a})$ | $(\mathbf{o},\mathbf{a},r)$ | $(\mathbf{o},\mathbf{a},r)$ | $(\mathbf{o},\mathbf{a})$ | $(\mathbf{o},\mathbf{a})$ | $(\mathbf{o},\mathbf{a},r)$ | $(\mathbf{o},\mathbf{a},r)$ |
| *Domain* | *Task Name* | | | | | | | |
| Franka | kitchen-complete | 1.4 | 1.8 | 3.08 | 1.76 ±0.04 | **3.37** ±0.19 | 1.22 ±0.18 | **3.37** ±0.01 |
| | kitchen-partial | 1.4 | 1.9 | **2.99** | 1.69 ±0.02 | 1.45 ±0.35 | 1.86 ±0.26 | 2.18 ±0.05 |
| | kitchen-mixed | 1.9 | 2.0 | | **2.15 ±0.06** | 1.51 ±0.39 | 2.03 ±0.06 | **2.25 ±0.14** |
| Adroit | pen-human | 1121.9 | 1214.0 | 1419.6 | 2141 ±109 | **2586 ±65** | 2108 ±58.8 | **2446 ±207** |
| | hammer-human | -82.4 | 300.2 | **496.2** | -38 ±25 | -133 ±26 | -35.1 ±45.1 | -9.3 ±45.5 |
| | door-human | -41.7 | 234.3 | **736.5** | 79 ±15 | 361 ±67 | 17.9 ± 13.8 | 399 ±34 |
| | relocate-human | -5.6 | 2.0 | 2.1 | -3.5 ±1.1 | -0.1 ±2.4 | -3.7 ±0.3 | **3.6±2.5** |
| Gym | halfcheetah-medium | 4202 | 5232 | 5778 | 4273 | 4086 | | |
| | walker2d-medium | 304 | 3637 | 4298 | 822 | 676 | | |
| | hopper-medium | 923 | 1867 | 2548 | 966 | 2430 | | |
| | halfcheetah-medium-replay | | 4934 | 6101 | 4029 | 2766 | | |
| | walker2d-medium-replay | | 970 | 1392 | 480 | 433 | | |
| | hopper-medium-replay | | 940 | 1132 | 543 | 382 | | |
| | halfcheetah-medium-expert | 4164 | 7467 | 9528 | 11758 | 4040 | | |
| | walker2d-medium-expert | 520 | 4533 | 5152 | 640 | 745 | | |
| | hopper-medium-expert | 3621 | 3592 | 3674 | 909 | 876 | | |
| | halfcheetah-expert | 13004 | 12731 | | 12802 | 9436 | | |
| | walker2d-expert | 5772 | 7067 | | 2677 | 3746 | | |
| | hopper-expert | 3527 | 3557 | | 3619 | 3549 | | |
| | halfcheetah-random | -118 | 4115 | 6213 | 0 | -392 | | |
| | walker2d-random | 33 | 323 | 1145 | 145 | -1.63 | | |
| | hopper-random | 308 | 331 | 331 | 284 | 308 | | |

*Table 4.* Baseline comparisons on D4RL [17] tasks, including Mujoco gym tasks. Results shown are the average of 3 random training initialization seeds, 100 evaluations each.

# D Policy Learning Results Overview and Protocol

In each section below we describe the protocols for the individual simulation experiments. Note that Figure 5 was produced by averaging the performance of the best policies, for each type, within each domain across the different tasks of that domain.

For EBM variants that were used for which task: Simulated Pushing and Real World, with action dimensionality of 2, used derivative-free optimization (Sec. B.1). For Planar Sweeping, with action dimensionality 3, and Bi-Manual Sweeping, with action dimensionality 12, we used autoregressive derivative-free optimization (Sec. B.2). D4RL, with action dimensionality between 3 and 30, used Langevin dynamics (Sec. B.3). Particle, with action dimensionality between 1 and 32, used Langevin dynamics as well. See Sec. B.4 for a comparison of variants.

## D.1 D4RL Experiments

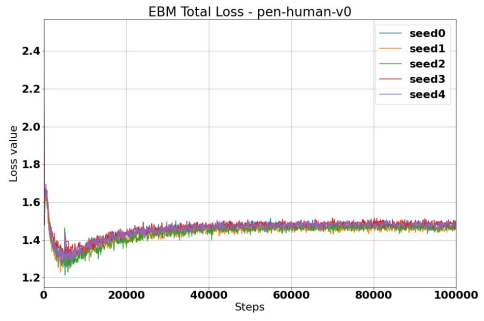
For **D4RL** experiments, we run sweeps over several hyperparameters for both the Implicit BC (EBM) and Explicit MSE-BC models. We choose the final hyperparameters based on max average performance over 3 D4RL environments: hammer-human-v0, door-human-v0, and relocate-human-v0. We use the same final hyperparameters across all D4RL tasks for the final results. Note that we paid closest attention to the human-teleoperation task performance when selecting a single set of hyper parameters for D4RL, particularly at the expense of slightly lower task performance on the gym-mujoco D4RL tasks. For all evaluations, we report average results over 100 episodes for 3 seeds. To calculate the aggregate D4RL performance metric “D4RL Human-Experts" in Figure 5 of the paper, we first calculated the normalized performance metric for the kitchen-complete, kitchen-partial, kitchen-mixed, pen-human, hammer-human, door-human and relocate-human environments, then calculated the average across all these tasks.

The following hyperparameters were used for D4RL evaluation:

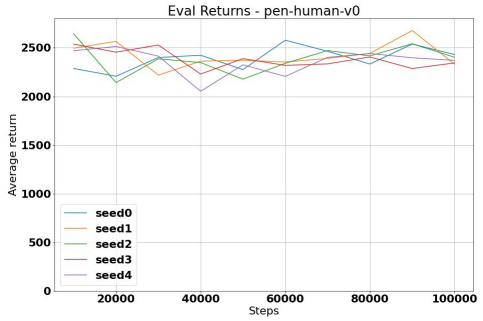
### D4RL Implicit BC (EBM)

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| EBM variant | Langevin | |
| train iterations | 100,000 | |
| batch size | 512 | |
| learning rate | 0.0005 | |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | |
| network size (width x depth) | 512x8 | 128x32, 512x8 |
| activation | ReLU | swish, ReLU |
| dense layer type | spectral norm | regular, spectral norm |
| train counter examples | 8 | 1, 8, 64 |
| action boundary buffer | 0.05 | 0.001, 0.05 |
| gradient penalty | final step only | all steps, final step only |
| gradient margin | 1 | 0.6, 1.0, 1.3 |
| langevin iterations | 100 | 100, 150 |
| langevin learning rate init. | 0.5 | 2.0, 1.0, 0.5, 0.1 |
| langevin learning rate final | 1.00E-05 | 1e-4, 1e-5, 1e-6 |
| langevin polynomial decay power | 2 | 2.0, 1.0 |
| langevin delta action clip | 0.5 | 0.05, 0.1, 0.5 |
| langevin noise scale | 0.5 | 0.5, 1.0 |
| langevin 2nd iteration learning rate | 1.00E-05 | 1e-1, 1e-2, 1e-5 |

Shown also is an indication of training stability, across 5 different seeds, shown for the pen task.




*(a)* A plot of the total EBM loss on the pen-human-v0 D4RL environment for each of 5 seeds. Note that with Langevin sampling, as the sample quality improves, the EBM loss can rise.




*(b)* A plot of the eval returns from the same run on pen-human-v0 for 5 seeds, average of 100 evals.

### D4RL Explicit MSE-BC

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| train iterations | 100,000 | |
| batch size | 512 | |
| sequence length | 2 | |
| learning rate | 0.001 | 1e-3, 0.5e-3 |
| learning rate decay | 0.99 | |
| learning rate decay steps | 200 | |
| dropout rate | 0.1 | 0.0, 0.1 |
| network size (width x depth) | 2048x8 | 128x16, 128x32, 512x16, 512x32, 1024x4, 1024x8, 2048x4, 2048x8 |
| activation | ReLU | |

## D.2 Simulated Pushing Experiments

For **Simulated Pushing** experiments, we run separate sweeps for each model for each of the States and Pixels versions of the task. All chosen hyperparameter sweeps and chosen values are given in tables below, and results are reported as the average of 100 episodes for 3 seeds.

## Simulated Pushing, States, Implicit BC (EBM)

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| EBM variant | DFO | |
| train iterations | 100,000 | |
| batch size | 512 | |
| sequence length | 2 | 2, 4 |
| learning rate | 0.001 | |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | |
| network size (width x depth) | 128x8 | 2048x4, 128x8, 128x16, 128x32 |
| activation | ReLU | |
| dense layer type | regular | |
| train counter examples | 256 | |
| action boundary buffer | 0.05 | |
| gradient penalty | none | |
| dfo samples | 16384 | |
| dfo iterations | 3 | |

## Simulated Pushing, States, Explicit MSE-BC

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| train iterations | 100,000 | |
| batch size | 512 | |
| sequence length | 2 | |
| learning rate | 0.0005 | 4e-3, 2e-3, 1e-3, 0.5e-3, 0.2e-3 |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | 100, 150, 200, 400 |
| dropout rate | 0.1 | |
| network size (width x depth) | 1024x8 | 1024x4, 1024x8, 2048x4, 2048x8 |
| activation | ReLU | |

## Simulated Pushing, States, Explicit MDN-BC

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| train iterations | 100,000 | |
| batch size | 512 | |
| sequence length | 2 | |
| learning rate | 0.001 | |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | |
| dropout rate | 0.1 | |
| network size (width x depth) | 512x8 | 512x8, 512x16 |
| training temperature | 1.0 | 0.5, 1.0, 2.0 |
| test temperature | 1.0 | 0.5, 1.0, 2.0 |
| test variance exponent | 1.0 | 1.0, 4.0 |

## Simulated Pushing, Pixels, Implicit BC (EBM)

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| EBM variant | DFO | |
| train iterations | 100,000 | |
| batch size | 128 | 128, 256 |
| sequence length | 2 | |
| learning rate | 0.001 | |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | |
| image size | 240x180 | 120x90, 240x180 |
| MLP network size (width x depth) | 1024x4 | 512x4, 1024x4, 256x14, 256x26, 1024x14, 1024x26 |
| Conv. Net. | 4-layer ConvMaxPool | |
| activation | ReLU | |
| dense layer type | regular | |
| train counter examples | 256 | |
| action boundary buffer | 0.05 | |
| gradient penalty | none | |
| dfo samples | 4096 | 1024, 4096, 16384 |
| dfo iterations | 3 | |

### Simulated Pushing Pixels MSE-BC

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| train iterations | 100,000 | |
| batch size | 64 | |
| sequence length | 2 | |
| learning rate | 0.001 | |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | |
| image size | 240x180 | 120x90, 240x180 |
| dropout rate (MLP only) | 0.1 | |
| network size (width x depth) | 512x4 | 128x2, 128x4, 512x2, 512x4 |
| Conv. Net. | 4-layer ConvMaxPool | |
| activation | ReLU | |
| coord conv | True | True, False |

### Simulated Pushing Pixels MDN-BC

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| train iterations | 100,000 | |
| batch size | 32 | |
| sequence length | 2 | |
| learning rate | 0.001 | |
| learning rate decay | 0.99 | |
| learning rate decay steps | 100 | |
| dropout rate (MLP only) | 0.1 | |
| image size | 120x90 | 120x90, 240x180 |
| network num components | 26 | |
| network size (width x depth) | 512x8 | 512x8, 512x16 |
| Conv. Net. | 4-layer ConvMaxPool | |
| activation | ReLU | |
| training temperature | 2.0 | 0.5, 1.0, 2.0 |
| test temperature | 2.0 | 0.5, 1.0, 2.0 |
| test variance exponent | 4.0 | 1.0, 4.0 |

## D.3 Simulated $N$-D Particle Environment Experiments

For a detailed description of this environment and its dynamics, see Section C.5. We used the following hyper parameters for evaluation on this environment:

### Particle Implicit BC (EBM)

| Hyperparameter | Chosen Value |
|---|---|
| EBM variant | Langevin |
| train iterations | 50,000 |
| batch size | 128 |
| sequence length | 2 |
| learning rate | 0.001 |
| learning rate decay | 0.99 |
| learning rate decay steps | 100 |
| network size (width x depth) | 128x16 |
| activation | ReLU |
| dense layer type | spectral norm |
| train counter examples | 64 |
| gradient penalty | final step only |
| gradient margin | 1 |
| langevin iterations | 50 |
| langevin learning rate init. | 0.1 |
| langevin learning rate final | 1.00E-05 |
| langevin polynomial decay power | 2 |
| langevin delta action clip | 0.1 |
| langevin noise scale | 1.0 |
| langevin 2nd iteration learning rate | not used |

### Particle Explicit MSE-BC

| Hyperparameter | Chosen Value |
|---|---|
| train iterations | 100,000 |
| batch size | 512 |
| sequence length | 2 |
| learning rate | 0.001 |
| learning rate decay | 0.99 |
| learning rate decay steps | 200 |
| dropout rate | 0.1 |
| network size (width x depth) | 128x16 |
| activation | ReLU |

## D.4 Simulated Sweeping Experiments

For **Planar Sweeping**, for both explicit and implicit models, results are shown for different types of encoders, and different # of Dense ResNet layers (Sec. E) shown in the table, each is the average of 100 evaluations each across 3 different seeds. The best models, for each implicit and explicit, were taken from Planar Sweeping and evaluated on **Bi-Manual Sweeping**.

We used the following hyper parameters for evaluation on the simulated planar sweeping, and bi-manual sweeping environment:

### Planar Sweeping Implicit BC (EBM)

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| EBM variant | Autoregressive | |
| train iterations | 1,000,000 | |
| batch size | 64 | |
| sequence length | 2 | |
| learning rate | 1e-4 | 1e-3, 1e-4 |
| Conv. Net. | ConvResNet | |
| # encoder features | 64 | |
| # Conv ResNet encoder layers | 26 | |
| # spatial softmax heads | 64 | 8, 16, 32, 64 |
| # dense ResNet layers | 20 | 8, 14, 20 |
| activation | ReLU | |
| train counter examples per action dim | 1024 | 128, 256, 512, 1024 |
| inference examples per action dim | 1024 | 128, 256, 512, 1024 |

### Planar Sweeping Explicit MSE-BC

| Hyperparameter | Chosen Value | Swept Values |
|---|---|---|
| train iterations | 1,000,000 | |
| batch size | 64 | |
| sequence length | 2 | |
| learning rate | 1e-4 | 1e-3, 1e-4 |
| Conv. Net. | ConvResNet | |
| # encoder features | 64 | |
| # Conv ResNet encoder layers | 26 | |
| # spatial softmax heads | 64 | 8, 16, 32, 64 |
| # dense ResNet layers | 20 | 8, 14, 20 |
| activation | ReLU | |

### Bi-manual Sweeping Implicit BC (EBM)

| Hyperparameter | Chosen Value |
|---|---|
| EBM variant | Autoregressive |
| train iterations | 1,000,000 |
| batch size | 32 |
| sequence length | 2 |
| learning rate | 1e-4 |
| Conv. Net. | ConvResNet |
| # encoder features | 64 |
| # Conv ResNet encoder layers | 26 |
| # spatial softmax heads | 64 |
| # dense ResNet layers | 20 |
| activation | ReLU |
| train counter examples per action dim | 1024 |
| inference examples per action dim | 1024 |

### Bi-manual Sweeping Explicit MSE-BC

| Hyperparameter | Chosen Value |
|---|---|
| train iterations | 1,000,000 |
| batch size | 32 |
| sequence length | 2 |
| learning rate | 1e-4 |
| Conv. Net. | ConvResNet |
| # encoder features | 64 |
| # Conv ResNet encoder layers | 26 |
| # spatial softmax heads | 64 |
| # dense ResNet layers | 20 |
| activation | ReLU |

## D.5 Real-world Pushing Experiments

For **Real World**, explicit and implicit models were taken from Simulated Pushing, Pixels, and applied to the real world. We used the following hyper parameters for evaluation on the real-world pushing environments:

### Real-world Tasks Pixels Implicit BC (EBM)

| Hyperparameter | Pushing | Pushing Multimodal | Insertion | Sorting |
|---|---|---|---|---|
| EBM variant | DFO | DFO | DFO | DFO |
| train iterations | 100,000 | 100,000 | 100,000 | 100,000 |
| batch size | 128 | 256 | 256 | 256 |
| sequence length | 2 | 2 | 2 | 2 |
| learning rate | 0.001 | 0.001 | 0.001 | 0.001 |
| learning rate decay | 0.99 | 0.99 | 0.99 | 0.99 |
| learning rate decay steps | 100 | 100 | 100 | 100 |
| image size | 120x90 | 120x90 | 120x90 | 120x90 |
| MLP network size (width x depth) | 1024x4 | 1024x4 | 2048x4 | 1024x4 |
| Conv. Net. | 4-layer ConvMaxPool | 4-layer ConvMaxPool | 4-layer ConvMaxPool | 4-layer ConvMaxPool |
| activation | ReLU | ReLU | ReLU | ReLU |
| dense layer type | regular | regular | regular | regular |
| train counter examples | 256 | 256 | 256 | 256 |
| action boundary buffer | 0.05 | 0.05 | 0.05 | 0.05 |
| gradient penalty | none | none | none | none |
| dfo samples | 1024 | 1024 | 2048 | 2048 |
| dfo iterations | 3 | 3 | 3 | 3 |

### Pushing Pixels MSE-BC

| Hyperparameter | Pushing | Pushing Multimodal | Insertion | Sorting |
|---|---|---|---|---|
| train iterations | 100,000 | 100,000 | 100,000 | 100,000 |
| batch size | 128 | 128 | 128 | 128 |
| sequence length | 2 | 2 | 2 | 2 |
| learning rate | 0.001 | 0.001 | 0.001 | 0.001 |
| learning rate decay | 0.99 | 0.99 | 0.99 | 0.99 |
| learning rate decay steps | 100 | 100 | 100 | 100 |
| image size | 120x90 | 120x90 | 120x90 | 120x90 |
| dropout rate (MLP only) | 0.1 | 0.1 | 0.1 | 0.1 |
| MLP network size (width x depth) | 512x4 | 1024x4 | 1024x4 | 1024x4 |
| Conv. Net. | 4-layer ConvMaxPool | 4-layer ConvMaxPool | 4-layer ConvMaxPool | 4-layer ConvMaxPool |
| activation | ReLU | ReLU | ReLU | ReLU |

# E Model Architectures

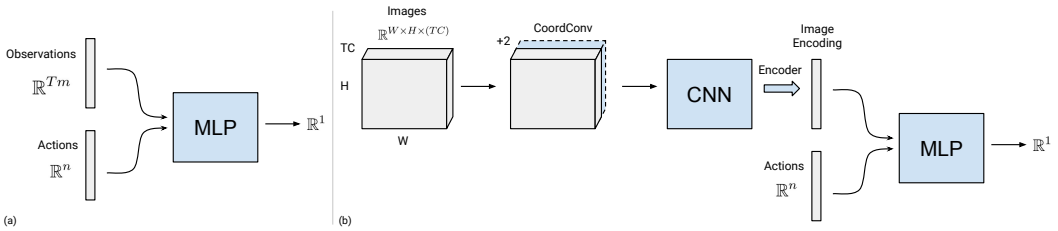



*Figure 4.* Simple depictions of architectures used for state-observation models (a), and visuomotor models (b). $T$ is sequence length, $m$ is observation dimensionality, $n$ is action dimensionality, $W$, $H$, $C$ are image width, height and channels.

.

## E.1 MLPs

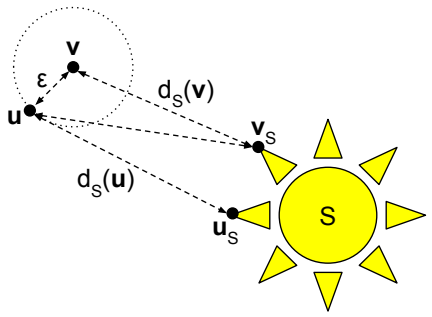
For non-image-observation models, we use MLPs (Multi Layer Perceptrons) that when used as EBMs (Fig. 4a), take in the actions and output an energy in $\mathbb{R}^1$, or when trained as MSE models instead output the actions. All results shown used ReLU activations, although we experimented with Swish as well. Configurable model elements consisted of: Dropout [18], using ResNet skip connections [19], and spectral normalization dense layers instead of regular dense layers [8].

## E.2 ConvMLPs

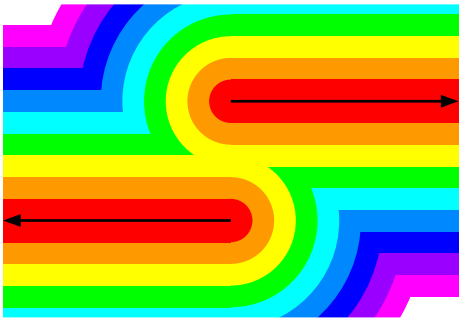
For visuomotor models (Fig. 4b), we use the common ConvMLP [20] style architecture, but when used as an EBM, concatenate actions with image encodings from a CNN model. The MLP portion is identical to the section above. For the CNNs, for all models for the sweeping experiments, we used 26-layer ResNets [21] ("ConvResNets") which maintain full-image spatial resolution before the encoder. For the simulated and real-world pushing experiments, we used a progressively-spatially-reduced model ("ConvMaxPool") composed of interleaving convolutions with max-pooling, with feature dimensions [32, 64, 128, 256]. Both models used 3x3 convolution kernels. Configurable options include: using CoordConv [22], i.e. a pixel coordinate map augmented as input, and either spatial soft (arg)max [20] or global average pooling encoders.

# F Proofs

In this section we prove Theorems 1 and 2 as stated in Section 5 of the paper.




*(a)* Distance to a set, $S$.



*(b)* Distance function isarithms for a step function.

## F.1 Definitions

A function $f$ is *Lipschitz continuous* with constant $L$ if $||f(\mathbf{x})-f(\mathbf{y})|| \leq L||\mathbf{x}-\mathbf{y}||$ for all $\mathbf{x}$,$\mathbf{y}$. We say that $f$ is $L$-Lipschitz, so a 1-Lipschitz function is a function that is continuous with Lipschitz constant 1. The magnitude of the gradient of an $L$-Lipschitz function is always less than or equal to $L$.

The *distance function* from a point $\mathbf{x} \in \mathbb{R}^n$ to a non-empty set, $S \subset \mathbb{R}^n$ is defined as:

$$d_S(\mathbf{x}) = \inf_{\mathbf{x}' \in S} ||\mathbf{x} - \mathbf{x}'||$$

A *closed set* is a set that contains all of its boundary points (points that can be approached from the interior and exterior of the set). Equivalently, a set is closed if and only if it contains all of its limit points (points that are the limit of some sequence of points in the set).

The *power set* of $\mathbb{R}^n$, $P(\mathbb{R}^n)$ is the set of all subsets of $\mathbb{R}^n$ including the empty set and all of $\mathbb{R}^n$.

The *graph*, $\mathcal{G}_F$, of a function $F : \mathbb{R}^m \to \mathbb{R}^n$ is the set of points:

$$\mathcal{G}_F = \{(\mathbf{x}, F(\mathbf{x})) \; \forall \; \mathbf{x} \in \mathbb{R}^m\} \subset \mathbb{R}^{m+n}$$

The *graph*, $\mathcal{G}_F$, of a multi-valued function $F : \mathbb{R}^m \to P(\mathbb{R}^n)$ is the set of points:

$$\mathcal{G}_F = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{m+n} \mid \mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in F(\mathbf{x})\}$$

### F.2 Proofs

**Lemma 1.** *The* distance function *from any point* $\mathbf{v}$ *to a non-empty set,* $S \subset \mathbb{R}^n$*, is well-defined and 1-Lipschitz.*

*Proof.* The distance function from a point $\mathbf{v}$ to a non-empty set, $S$ is defined as:

$$d_S(\mathbf{v}) = \inf_{\mathbf{v}_S \in S} ||\mathbf{v} - \mathbf{v}_S||$$

The set of distance values is a set of positive real numbers, so the infimum exists due to the *completeness* of $\mathbb{R}$. Therefore the distance function is well defined.

For any $\mathbf{v}$, let $\mathbf{v}_S$ be a point in the closure of $S$ with $||\mathbf{v} - \mathbf{v}_S|| = d_S(\mathbf{v})$. Then, to establish continuity using the triangle inequality, we can state that for a given $\mathbf{u}$ at a distance $\epsilon$ from $\mathbf{v}$ (as pictured in Fig. 5a),

$$\begin{aligned}
||\mathbf{u} - \mathbf{v}|| &= \epsilon \\
d_S(\mathbf{u}) &= ||\mathbf{u} - \mathbf{u}_S|| \\
&\leq ||\mathbf{u} - \mathbf{v}_S|| && d_S \text{ is an infimum} \\
&\leq ||\mathbf{u} - \mathbf{v}|| + ||\mathbf{v} - \mathbf{v}_S|| && \text{by the triangle inequality} \\
d_S(\mathbf{u}) &\leq \epsilon + d_S(\mathbf{v}) \\
d_S(\mathbf{v}) &\leq \epsilon + d_S(\mathbf{u}) && \mathbf{v} \text{ and } \mathbf{u} \text{ can be exchanged.} \\
|d_S(\mathbf{u}) - d_S(\mathbf{v})| &\leq \epsilon \\
|d_S(\mathbf{u}) - d_S(\mathbf{v})| &\leq 1 \cdot ||\mathbf{u} - \mathbf{v}||
\end{aligned}$$

Since $\mathbf{u}$ and $\mathbf{v}$ can be reversed we have, $|d_S(\mathbf{u}) - d_S(\mathbf{v})| < \epsilon$ and thus $d_S$ is continuous over $\mathbb{R}^n$ with a Lipschitz constant of 1. □

**Lemma 2.** *If* $d_S : \mathbb{R}^n \to \mathbb{R}$ *is the distance function to a* closed *set* $S \subset \mathbb{R}^n$*, then for every* $\mathbf{x} \in \mathbb{R}^n$ *there exists an element* $\mathbf{x}' \in S$ *such that* $d_S(\mathbf{x}) = ||\mathbf{x} - \mathbf{x}'||$.

*Proof.* Let $B$ be a closed ball of radius $d_S(\mathbf{x}) + 1$ around $\mathbf{x}$. The distance from $\mathbf{x}$ to $B \cap S$ is equal to the distance from $\mathbf{x}$ to $S$. Since $d_S$ is defined as an infimum, there must exist an infinite sequence of points $\{\mathbf{x}_i\} \subset B \cap S$ with distances $d_i = ||\mathbf{x} - \mathbf{x}_i||$ whose limit is $d_S(\mathbf{x})$. The set $B \cap S$ is closed and bounded and, therefore, compact. The infinite sequence $\{x_i\}$ must therefore have at least one sub-sequence that converges to a point $\mathbf{x}' \in B \cap S$. Since the distances of the full series converge to $d_S(\mathbf{x})$, we know that $||\mathbf{x} - \mathbf{x}'|| = d_S(\mathbf{x})$. □

**Lemma 3.** *For any continuous function* $F(\mathbf{x}) : \; \mathbf{x} \in \mathbb{R}^m \to \mathbb{R}^n$*, the distance to the graph of* $F$ *is a continuous function* $g(\mathbf{x}, \mathbf{y}) : \mathbb{R}^{m+n} \to \mathbb{R}$*, such that* $F_g(\mathbf{x}) = \textit{argmin}_{\mathbf{y}} \; g(\mathbf{x}, \mathbf{y}) = F(\mathbf{x})$ *for all* $\mathbf{x}$*.*

*Proof.* Let $g(\mathbf{x},\mathbf{y})$ be the distance in $\mathbb{R}^{m+n}$ from the point $(\mathbf{x},\mathbf{y})$ to the graph of $F$.

The *graph*, $\mathcal{G}_F$, of a function $F:\mathbb{R}^m \rightarrow \mathbb{R}^n$ is the set of points:

$$\mathcal{G}_F = \{(\mathbf{x},F(\mathbf{x})) \,\forall\, \mathbf{x} \in \mathbb{R}^m\} \subset \mathbb{R}^{m+n}$$

Since the graph $\mathcal{G}_F$ is a non-empty set the distance function $g(\mathbf{x},\mathbf{y})$ is well defined and continuous, as shown in Lemma 1.

We must still show that $F_g(\mathbf{x}) = \text{argmin}_{\mathbf{y}}\ g(\mathbf{x},\mathbf{y}) = F(\mathbf{x})$ for all $\mathbf{x}$. We know that $g(\mathbf{x},\mathbf{y}) \geq 0\ \forall \mathbf{x},\mathbf{y}$, because $g$ is a distance function.

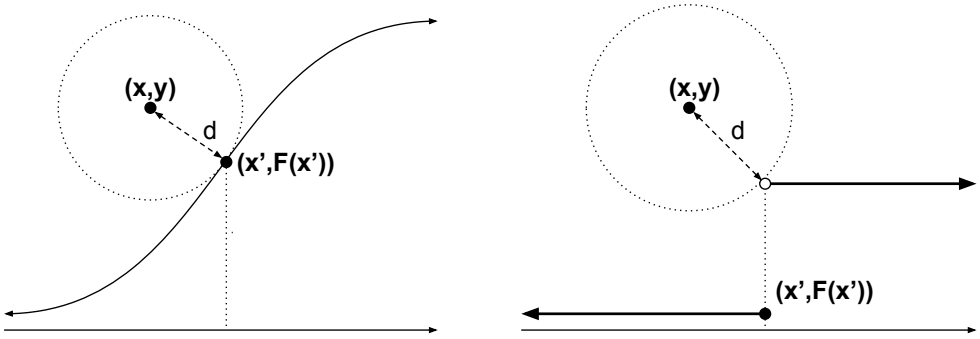
For any $\mathbf{x} \in \mathbb{R}^n$, clearly $g(\mathbf{x},F(\mathbf{x})) = 0$, since the point $(\mathbf{x},F(\mathbf{x})) \in \mathcal{G}_F$ and thus the distance from $(\mathbf{x},F(\mathbf{x}))$ to a point in $\mathcal{G}_F$ is zero.

Consider a point $(\mathbf{x},\mathbf{y})$ where $\mathbf{y} \neq F(\mathbf{x})$ and therefore $(\mathbf{x},\mathbf{y}) \notin \mathcal{G}_F$. Since $F$ is continuous, $\mathcal{G}_F$ is closed and there will exist a point, $(\mathbf{x}',F(\mathbf{x}')) \in \mathcal{G}_F$ that achieves the infimum, $d_{\mathcal{G}}((\mathbf{x},\mathbf{y})) = ||(\mathbf{x},\mathbf{y}) - (\mathbf{x}',F(\mathbf{x}'))||$.

$$||(\mathbf{x},\mathbf{y}) - (\mathbf{x}',F(\mathbf{x}'))|| = ||(\mathbf{x}-\mathbf{x}',\mathbf{y}-F(\mathbf{x}'))|| \leq ||\mathbf{x}-\mathbf{x}'|| + ||\mathbf{y}-F(\mathbf{x}')||$$

At least one of $\mathbf{x} \neq \mathbf{x}'$ or $\mathbf{y} \neq F(\mathbf{x}')$, so $g(\mathbf{x},F(\mathbf{x})) > 0$.

Therefore, for any $\mathbf{x} \in \mathbb{R}^m$, $g(\mathbf{x},\mathbf{y})$ achieves its unique minimum $g(\mathbf{x},\mathbf{y}) = 0$ at $\mathbf{y} = F(\mathbf{x})$ and thus $F_g(\mathbf{x}) = \text{argmin}_{\mathbf{y}}\ g(\mathbf{x},\mathbf{y}) = F(\mathbf{x})$. □




*(a)* Distance to the graph of a continuous function.

*(b)* Distance to the graph of a (single-valued) discontinuous function.

We have shown that for $F : \mathbb{R}^m \rightarrow \mathbb{R}^n$ we can construct a continuous $g(\mathbf{x},\mathbf{y})$ that satisfies $F_g(\mathbf{x}) = \text{argmin}_{\mathbf{y}}\ g(\mathbf{x},\mathbf{y}) = F(\mathbf{x})$ for all $x \in \mathbb{R}^m$ if $F$ is single-valued and continuous. However, the functions we are modeling are often discontinuous or multi-valued. If the single-valued function is discontinuous, there will be open boundaries on the graph where the point that minimizes the distance function is not in the graph of $F$ (Fig. 6b). In that example, there will be two values of $\mathbf{y}$ that minimize $g$ for the same value of $\mathbf{x}$, in which case $F_g(\mathbf{x})$ will not be well defined as a single-valued function. We can disambiguate the two cases to get a well-defined $F_g$, but we cannot reliably recover the original $F$ at the discontinuity.

In order to handle discontinuities and multi-valued functions, we will extend the definition to allow functions that map to multiple values, $F:\mathbb{R}^m \rightarrow P(\mathbb{R}^n)\backslash\{\emptyset\}$. The multi-valued function $F$ maps from $\mathbb{R}^m$ to the *power set* $P(\mathbb{R}^n)\backslash\{\emptyset\}$, which is the set of all subsets of $\mathbb{R}^n$, except the empty set. We no longer require continuity, but instead directly require the one important property of a continuous function that was used in the proof of Lemma 3, namely that the graph of $F$ is closed. In the simple case of a jump discontinuity (as in fig. 6b), the function must include both sides of the discontinuity.

**Theorem 1.** *For any multi-valued (set-valued) function $F(\mathbf{x}) \colon \mathbf{x} \in \mathbb{R}^m \rightarrow P(\mathbb{R}^n)\backslash\{\emptyset\}$ where the graph of $F$ is closed, there exists a 1-Lipschitz function $g(\mathbf{x},\mathbf{y}) \colon \mathbb{R}^{m+n} \rightarrow \mathbb{R}$, such that argmin$_{\mathbf{y}}\ g(\mathbf{x},\mathbf{y}) = F(\mathbf{x})$ for all $\mathbf{x}$.*

*Proof.* The *graph*, $\mathcal{G}_F$, of a multi-valued function $F:\mathbb{R}^m \rightarrow P(\mathbb{R}^n)$ is the set of points:

$$\mathcal{G}_F = \{(\mathbf{x},\mathbf{y}) \in \mathbb{R}^{m+n} \mid \mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in F(\mathbf{x})\}$$

We can again define $g$ as the distance to $\mathcal{G}_F$. Because $\mathcal{G}_F$ is a non-empty set, we know that $g$ is well-defined and uniformly continuous (Lemma 1).

$$g(\mathbf{x},\mathbf{y}) = d_{\mathcal{G}_F}((\mathbf{x},\mathbf{y}))$$

We will now show that $g(\mathbf{x},\mathbf{y})=0$ for all points in $\mathcal{G}_F$ and $g(\mathbf{x},\mathbf{y})>0$ for all points not in $\mathcal{G}_F$.

For any point $(\mathbf{x},\mathbf{y})\in\mathcal{G}_F$, that is $\mathbf{x}\in\mathbb{R}^m$ and $\mathbf{y}\in F(\mathbf{x})$, the distance from $(\mathbf{x},\mathbf{y})$ to $\mathcal{G}_F$ is zero.

$$g(\mathbf{x},\mathbf{y}) = d_{\mathcal{G}_F}((\mathbf{x},\mathbf{y})) = 0$$

For any point $(\mathbf{x},\mathbf{y})\notin\mathcal{G}_F$, that is $\mathbf{x}\in\mathbb{R}^m$ and $\mathbf{y}\notin F(\mathbf{x})$, we must show that the distance to $\mathcal{G}_F$ is strictly positive. Since the graph $\mathcal{G}_F$ is closed (because we require it to be so), by Lemma 2, we know that there exists a point $(\mathbf{x}',\mathbf{y}')\in\mathcal{G}_F$ that achieves the minimum distance exactly.

$$\begin{aligned} g(\mathbf{x},\mathbf{y}) &= d_{\mathcal{G}_F}((\mathbf{x},\mathbf{y})) \\ &= ||(\mathbf{x},\mathbf{y})-(\mathbf{x}',\mathbf{y}')|| \\ &= ||(\mathbf{x}-\mathbf{x}',\mathbf{y}-\mathbf{y}')|| \\ &\leq ||\mathbf{x}-\mathbf{x}'||+||\mathbf{y}-\mathbf{y}'|| \end{aligned}$$

At least one of $\mathbf{x}\neq\mathbf{x}'$ or $\mathbf{y}\neq\mathbf{y}'$, so $g(\mathbf{x},\mathbf{y})>0$.

Because the empty set is excluded from the range of $F$, there will be at least one $\mathbf{y}\in F(\mathbf{x})$ for any $\mathbf{x}$. Therefore, at $\mathbf{x}$, the minimum value of $g(\mathbf{x},\mathbf{y})$ will be zero and $\text{argmin}_{\mathbf{y}} g(\mathbf{x},\mathbf{y})=F(\mathbf{x})$ exactly.

We have shown that the implicit function $g(\mathbf{x},\mathbf{y})$ is well-defined and continuous and has a Lipschitz value of 1, even if $F$ is very badly behaved. □

**Corollary 1.1** *The function $g(\mathbf{x},\mathbf{y})$ in Thm. 1 can be chosen to have an arbitrary positive Lipschitz constant.*

*Proof.* The distance function $d_S(\mathbf{x}):\mathbb{R}^n\to\mathbb{R}$ from any point $\mathbf{x}$ to a non-empty set $S$, has a Lipschitz constant of 1 (Lemma 1). Let $g_1 = d_{\mathcal{G}_F}$, the distance to the graph of $F$. If our desired Lipschitz constant is $L>0$, we can compose $g_1$ with another function $f_L:\mathbb{R}\to\mathbb{R}$ that has a Lipschitz constant of $L$ to get $g_L=f_L\circ g_1$. For example, if $f_L(x)=Lx$, we get $g_L(\mathbf{x},\mathbf{y})=f_L(g_1(\mathbf{x},\mathbf{y}))=L\,d_{\mathcal{G}_F}(\mathbf{x},\mathbf{y})$.

$$\begin{aligned} |g_L(\mathbf{x},\mathbf{y})-g_L(\mathbf{x}',\mathbf{y}')| &= |f_L(g_1(\mathbf{x},\mathbf{y}))-f_L(g_1(\mathbf{x}',\mathbf{y}'))| \\ &\leq L\,|g_1(\mathbf{x},\mathbf{y})-g_1(\mathbf{x}',\mathbf{y}')| \\ &\leq L\cdot 1\,||(\mathbf{x},\mathbf{y})-(\mathbf{x}',\mathbf{y}')|| \end{aligned}$$

Therefore, $g_L$ has a Lipschitz constant of $L$. □

**Theorem 2.** *For any set-valued function $F(\mathbf{x}):\mathbb{R}^m\to P(\mathbb{R}^n)\setminus\{\emptyset\}$, there exists a continuous implicit function $g:\mathbb{R}^{m+n}\to\mathbb{R}$ that has a* continuous function approximator*, $g_\theta$ with arbitrarily small bounded error $\epsilon$, that provides the guarantee that any point in the graph of $F_\theta(\mathbf{x})=\text{argmin}_{\mathbf{y}} g_\theta(\mathbf{x},\mathbf{y})$ is within $\epsilon$ of the graph of $F$.*

*Proof.* Let $g(\mathbf{x},\mathbf{y})=2d_{\mathcal{G}_F}(\mathbf{x},\mathbf{y})$, twice the distance to the graph of $F$. By Thm. 1, this $g$ is continuous and satisfies $\text{argmin}_{\mathbf{y}} g(\mathbf{x},\mathbf{y})=F(\mathbf{x})$.

For an arbitrary $\epsilon>0$, let $g_\theta:\mathbb{R}^{m+n}\to\mathbb{R}$ be a function approximator for $g$ with bounded error $\epsilon$, $|g_\theta-g|<\epsilon$. Since $g$ is a continuous function, the existence of a bounded-error function approximator is guaranteed by well-known results in universal approximation of continuous functions, for example [23].

The question now is whether bounded errors in $g_\theta$ approximating $g$, when composed with argmin, can provide any guarantee on a property of $\text{argmin}_{\mathbf{y}} g_\theta(\mathbf{x},\mathbf{y})$.

Note that $F_\theta$ as an approximator for $F$ is unbounded, since $F$ may be badly behaved. This can be demonstrated at any point where $F$ has a discontinuity. Suppose $||\mathbf{x}-\mathbf{x}'||<\epsilon$ and $F(\mathbf{x})$ and $F(\mathbf{x}')$ have

values that are arbitrarily far apart. Because of the $\epsilon$ error in $g_\theta$, the $\text{argmin}_{\mathbf{y}} g_\theta(\mathbf{x},\mathbf{y})$ may find values in $F(\mathbf{x}')$ that introduce arbitrary error in $F_\theta(\mathbf{x})$.

Let $F_\theta(\mathbf{x}) = \text{argmin}_{\mathbf{y}} g_\theta(\mathbf{x},\mathbf{y})$. For any point $(\mathbf{x},\mathbf{y})$ in the graph of $F_\theta$, we can show that $g_\theta(\mathbf{x},\mathbf{y}) < \epsilon$. Let $\mathbf{y}'$ be any point in $F(\mathbf{x})$. Since $(\mathbf{x},\mathbf{y}')$ is in the graph of $F$, we know that $g(\mathbf{x},\mathbf{y}') = 0$. With the bounded error, $|g_\theta - g| < \epsilon$, we know that $g_\theta(\mathbf{x},\mathbf{y}') < \epsilon$. Thus, although the argmin may be achieved elsewhere, $g_\theta(\mathbf{x},\mathbf{y})$ may not have a value greater than $\epsilon$.

$$\begin{aligned} g_\theta(\mathbf{x},\mathbf{y}) &\leq g_\theta(\mathbf{x},\mathbf{y}') \\ &< g(\mathbf{x},\mathbf{y}') + \epsilon \\ g_\theta(\mathbf{x},\mathbf{y}) &< \epsilon \end{aligned}$$

Also because $g_\theta$ is an approximator for $g$, we know that for $(\mathbf{x},\mathbf{y})$ in the graph of $F_\theta$, $g(\mathbf{x},\mathbf{y}) < g_\theta(\mathbf{x},\mathbf{y}) + \epsilon$ and thus $g(\mathbf{x},\mathbf{y}) < 2\epsilon$. Since $g(\mathbf{x},\mathbf{y})$ is twice the distance to the graph of $F$, the distance from $(\mathbf{x},\mathbf{y})$ to the graph of $F$ is less than $\epsilon$.

$$\begin{aligned} g(\mathbf{x},\mathbf{y}) &\leq g_\theta(\mathbf{x},\mathbf{y}) + \epsilon \leq 2\epsilon \\ g(\mathbf{x},\mathbf{y}) &= 2 d_{\mathcal{G}_F}(\mathbf{x},\mathbf{y}) \leq 2\epsilon \\ d_{\mathcal{G}_F}(\mathbf{x},\mathbf{y}) &\leq \epsilon \end{aligned}$$

Therefore, any point $(\mathbf{x},\mathbf{y})$ in the graph of $F_\theta$ must lie within $\epsilon$ of $\mathcal{G}_F$, the graph of $F$.

Note that this is not symmetric. There are no guarantees on $g_\theta$ other than that it is continuous and within $\epsilon$ of $g$. A point in $\mathcal{G}_F$ may be arbitrarily far from $\mathcal{G}_{F_\theta}$ as tiny variations in $g_\theta$ can eliminate some values from the set $F(\mathbf{x})$. □

# G Theory Implications and Discussion

The practical implications of Theorems 1 and 2 are that they provide a number of favorable properties for real-world modeling tasks, such as robot policy learning, that exhibit discontinuities and multi-modalities. For one, implicit functions can model steep or discontinuous oracle policies without large gradients in the function approximator. We hypothesize this leads to policies with better stability characteristics and fewer generalization issues for out-of-domain samples. Thm. 1 also shows that implicit models can represent multi-valued (set-valued) functions, including ones with discontinuities. With Thm. 2 there is a notion provided of still having guarantees on the output of implicit inference, despite expected errors in the function approximator. One way to consider the guarantee is via the level sets of the function, as shown in Figure 10 of the main paper. To discuss an illustrative example, for a simple step function discontinuity (as in Figure 10) this guarantee provides that, although the precise "decision boundary" of the discontinuity may not be represented perfectly, that decision boundary can be approximated to arbitrary precision. Further, even when the decision boundary is estimated imperfectly, the inferred values will correspond to either side of the discontinuity, and nowhere else (it will not, for example estimate a value somewhere halfway between the two sides of the discontinuity).

Note also that Theorems 1 and 2 do not show that a learning algorithm will actually be able to recover a model with such properties, but only that such a model exists. This is in line with other results in function approximation with neural networks, for example [23]. Additionally, in the implicit model case another consideration is not only whether the parameters $\theta$ of such a parameterized model, $g_\theta$ can be found, but then also whether at inference time the optimization problem $\text{argmin}_{\mathbf{y}} g_\theta(\mathbf{x},\mathbf{y})$ can even be solved. In general, $g(\cdot)$ will be non-convex, and may be a hard global optimization problem. We have shown in practice, however, we are able to perform satisfactory inference on many policy learning tasks of interest.

# H Limitations

Although we have examined performance increases on several tasks using implicit BC policies over explicit BC policies, evaluated as a method for behavioral cloning, there are a few limitations. For one, compared to a simple Mean Square Error (MSE) BC policy, there is both increased training and inference computational

cost. As shown in Sec. C.2, however, the increase in inference time for the models used in the real world is modest, and we have validated the models in the real world to be fast enough for real-time vision-based control. Further, the training times of our presented models are modest compared to reported numbers for offline RL methods (Sec. C.2) A second limitation is the increased implementation complexity of implicit models compared to explicit models. However, we have provided a guide (Sec. B) for how to train the models, which we hope encourages readers to try implicit models.

# References


[1] Y. Song and D. P. Kingma. How to train your energy-based models. *arXiv preprint arXiv:2101.03288*, 2021.

[2] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[3] C. Nash and C. Durkan. Autoregressive energy machines. In *International Conference on Machine Learning*, pages 1735–1744. PMLR, 2019.

[4] Y. Du and I. Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32:3608–3618, 2019.

[5] I. Mordatch. Concept learning with energy-based models. *arXiv preprint arXiv:1811.02486*, 2018.

[6] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[7] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, M. Norouzi, and K. Swersky. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

[8] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[9] A. Jolicoeur-Martineau and I. Mitliagkas. Gradient penalty from a maximum margin perspective. *arXiv preprint arXiv:1910.06922*, 2021.

[10] Y. Du, S. Li, B. J. Tenenbaum, and I. Mordatch. Improved contrastive divergence training of energy based models. In *Proceedings of the 38th International Conference on Machine Learning (ICML-21)*, 2021.

[11] I. Kostrikov, J. Tompson, R. Fergus, and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

[12] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub Repository*, 2016.

[13] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, 2001.

[14] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[15] S. Sinha and A. Garg. S4rl: Surprisingly simple self-supervision for offline reinforcement learning. *arXiv preprint arXiv:2103.06326*, 2021.

[16] J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750, 2007.

[17] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[20] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research (JMLR)*, 2016.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[22] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in Neural Information Processing Systems*, 31, 2018.

[23] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.