7 Appendix

7.1 Proof of Identity 1

We wish to derive the following identity from the main text:

Identity 1. *Let E_1 and E_2 be two energy-based models that respectively define distributions π_1 and π_2 according to Equation 4. Then,*

$$D_J\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) = \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right].$$

Proof. The proof follows from applying the definition of Jeffreys divergence to EBMs:

$$\begin{aligned}D_J\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) &\triangleq D_{KL}\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right) + D_{KL}\left(\pi_2(\cdot|s)\|\pi_1(\cdot|s)\right)\\ &\triangleq \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[\log\frac{\pi_1(a|s)}{\pi_2(a|s)}\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[\log\frac{\pi_2(a|s)}{\pi_1(a|s)}\right]\\ &= \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] - \log Z_1(s) + \log Z_2(s)\\ &\quad + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right] - \log Z_2(s) + \log Z_1(s)\\ &= \mathbb{E}_{a\sim\pi_1(\cdot|s)}\left[E_2(s,a) - E_1(s,a)\right] + \mathbb{E}_{a\sim\pi_2(\cdot|s)}\left[E_1(s,a) - E_2(s,a)\right].\end{aligned}$$

□

7.2 Additional Details on Implicit Models

Implicit BC trains an energy-based model E_θ on samples $\{s_i, a_i\}$ collected from the expert policies π_H. After generating a set of counter-examples $\{\tilde{a}_i^j\}$ for each s_i, Implicit BC minimizes the following InfoNCE [46] loss function:

$$\mathcal{L} = \sum_{i=1}^{N} -\log \hat{p}_\theta(a_i|s_i, \{\tilde{a}_i^j\}), \quad \hat{p}_\theta(a_i|s_i,\{\tilde{a}_i^j\}) := \frac{e^{-E_\theta(s_i,a_i)}}{e^{-E_\theta(s_i,a_i)} + \sum_j e^{-E_\theta(s_i,\tilde{a}_i^j)}}. \tag{5}$$

This loss is equivalent to the negative log likelihood of the training data, where the partition function $Z(s)$ is estimated with the counter-examples. Florence et al. [16] propose three techniques for generating these counter-examples $\{\tilde{a}_i^j\}$ and performing inference over the learned model E_θ; we choose gradient-based Langevin sampling [47] with an additional gradient penalty loss for training in this work as Florence et al. [16] demonstrate that it scales with action dimensionality better than the alternate methods. This is a Markov Chain Monte Carlo (MCMC) method with stochastic gradient Langevin dynamics. More details are available in Appendix B.3 of Florence et al. [16].

We use the following hyperparameters for implicit model training and inference:

Hyperparameter	Value
learning rate	0.0005
learning rate decay	0.99
learning rate decay steps	100
train counter examples	8
langevin iterations	100
langevin learning rate init.	0.1
langevin learning rate final	1e-5
langevin polynomial decay power	2
inference counter examples	512

Table 3: Implicit model hyperparameters.

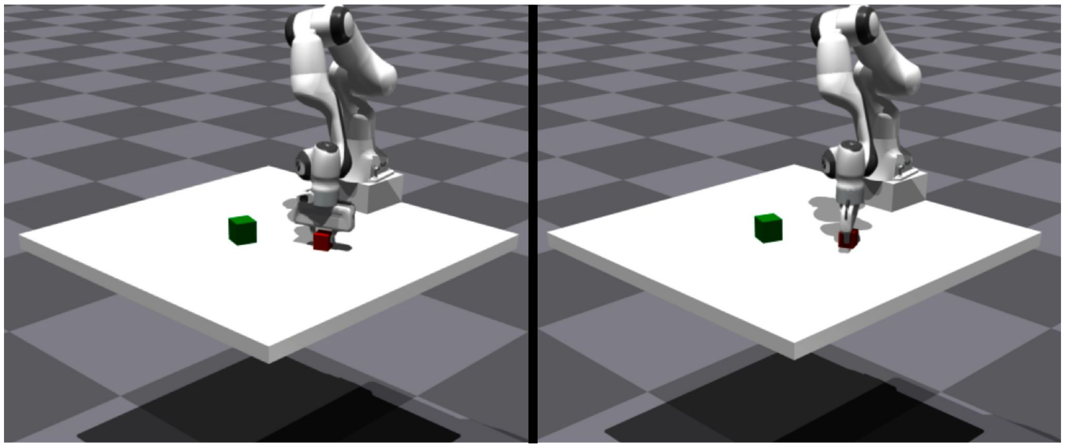

Figure 5: The two scripted heterogeneous supervisors for the FrankaCubeStack Isaac Gym environment pick different faces of the cube for the same cube pose.

7.3 Additional Experimental Details

7.3.1 IFL Benchmark Hyperparameters

Implementations of Implicit Interactive Fleet Learning and baselines are available in the code supplement and are configured to run with the same hyperparameters we used in the experiments. To compute the uncertainty thresholds $\hat{u}$ for Explicit IFL and IIFL (see Section 8.3.1 in [13] for definition), we run Explicit BC and Implicit BC respectively with $N = 100$ robots for $T = 1000$ timesteps and choose the 99th percentile value among all 100×1000 uncertainty values. The FrankaCubeStack environment sets these thresholds to zero since there are no constraint violations (i.e., this sorts robot priority by uncertainty alone). See Table 4 for these values, state and action space dimensionality, and other hyperparameters. The batch size is 512 and all algorithms pretrain the policy for $N/2$ gradient steps, where N is the number of data points in the 10 offline task demonstrations. Finally, as in prior work [13], the Random IIFL baseline is given a human action budget that approximately equals the average amount of human supervision solicited by IIFL. See the code for more details.

Environment	$\lvert S\rvert$	$\lvert A\rvert$	Explicit $\hat{u}$	Implicit $\hat{u}$
BallBalance	24	3	0.1179	0.1206
Ant	60	8	0.0304	0.9062
Anymal	48	12	0.0703	2.2845
FrankaCubeStack	19	7	0.0	0.0

Table 4: Simulation environment hyperparameters.

7.3.2 FrankaCubeStack Environment

The scripted supervisor for FrankaCubeStack is defined in `human_action()` of `env/isaacgym/franka_cube_stack.py` in the code supplement. Using known pose information and Cartesian space control, the supervisor policy does the following, where Cube A is to be stacked on Cube B: (1) move the end effector to a position above Cube A; (2) rotate into a pre-grasp pose; (3) descend to Cube A; (4) lift Cube A; (5) translate to a position above Cube B; (6) place Cube A on Cube B; and (7) release the gripper. Heterogeneity is concentrated in Step 2: while one supervisor rotates to an angle θ that corresponds to a pair of antipodal faces of the cube, the other rotates to $\theta - \frac{\pi}{2}$ to grab the other pair of faces. See Figure 5 for intuition.

7.3.3 Physical Experiment Protocol

We largely follow the physical experiment protocol in Hoque et al. [13] but introduce some modifications to human supervision. We execute 3 trials of each of 4 algorithms (Explicit BC, Implicit BC, Explicit IFL, Implicit IFL) on the fleet of 4 robot arms. Each trial lasts 150 timesteps (synchronous across the fleet) for a total of 3 × 4 × 4 × 150 = 7200 individual pushing actions. The authors provide human teleoperation and hard resets, which differ from prior work due to the continuous action space and the square obstacle in the center of the workspace. Teleoperation is done using an OpenCV (https://opencv.org/) GUI by clicking on the desired end point of the end-effector in the overhead camera view. Hard resets are physical adjustments of the cube to a randomly chosen side of the obstacle. IIFL is trained online with updated data at $t = 50$ and $t = 100$ while IFL is updated at every timestep (with an equivalent total amount of gradient steps) to follow prior work [13].

The rest of the experiment protocol matches Hoque et al. [13]. The 2 ABB YuMi robots are located about 1 km apart; a driver program uses the Secure Shell Protocol (SSH) to connect to a machine that is connected to the robot via Ethernet, sending actions and receiving camera observations. Pushing actions are executed concurrently by all 4 arms using multiprocessing. We set minimum intervention time $t_T = 3$ and hard reset time $t_R = 5$. All policies are initialized with an offline dataset of 3360 image-action pairs (336 samples collected by the authors with 10× data augmentation). 10× data augmentation on the initial offline dataset as well as the online data collected during execution applies the following transformations:

- Linear contrast uniformly sampled between 85% and 115%
- Add values uniformly sampled between -10 and 10 to each pixel value per channel
- Gamma contrast uniformly sampled between 90% and 110%
- Gaussian blur with σ uniformly sampled between 0.0 and 0.3
- Saturation uniformly sampled between 95% and 105%
- Additive Gaussian noise with σ uniformly sampled between 0 and $\frac{1}{80} \times 255$ 80 × 255

7.3.4 Computation Time

In Table 5 we report the mean and standard deviation of various computation time metrics. All timing experiments were performed with $N = 100$ robots and averaged across $T = 100$ timesteps in the Ant environment on a single NVIDIA Tesla V100 GPU with 32 GB RAM. Training time is reported for a single gradient step with a batch size of 512. Note that with default hyperparameters, IFL trains an ensemble of 5 (explicit) models and IIFL trains an ensemble of 2 (implicit) models; hence, we also report the training time per individual model. IFL inference consists of a single forward pass through each of the 5 models, while IIFL inference performs 100 Langevin iterations; both of these are vectorized across all 100 robots at once. IFL uncertainty estimation also consists of a single forward pass through each of the 5 models while IIFL performs both Langevin iterations and 2 forward passes through each of the 2 models. While IIFL can provide policy performance benefits over IFL, we observe that it comes with a tradeoff of computation time, which may be mitigated with parallelization across additional GPUs. Furthermore, while uncertainty estimation is the bottleneck in IIFL, it is performed with sub-second latency for the entire fleet. This is significantly faster than alternatives such as directly estimating the partition function, which is both less accurate and slower; we measure it to take an average of 7.10 seconds per step using annealed importance sampling [48].

Time	IFL	IIFL
Training step (s)	0.0385 ± 0.0205	0.694 ± 0.207
Training step per model (s)	0.0077 ± 0.0041	0.347 ± 0.104
Inference (s)	0.0060 ± 0.0395	0.494 ± 0.045
Uncertainty estimation (s)	0.0029 ± 0.0008	0.988 ± 0.008

Table 5: Computation times for training, inference, and uncertainty estimation for IFL and IIFL.