A Additional Results

A.1 Ablations

In this section we provide a number of additional ablations on the parameters of our method in the MetaWorld environments. Specifically, we vary the amount of total feedback available for both our method and PEBBLE. We train models with PEBBLE using the original amount of feedback in Lee et al. [18], or 20× the amount of feedback used in Section 4.1 and Figure 2. Even with 20× less feedback, our method is at par with PEBBLE. We also train models with our method using only half of the feedback used in Figure 2, and attain nearly the same performance in Window Close, Door Unlock, and Sweep-Into. This indicates that with better parameter tuning, our method could be even more query efficient. Next, we investigate the effects of the disagreement query selection scheme in Figure 7. Disagreement sampling leads to performance improvements in some environments, particularly in Drawer Open, but makes no difference in others.

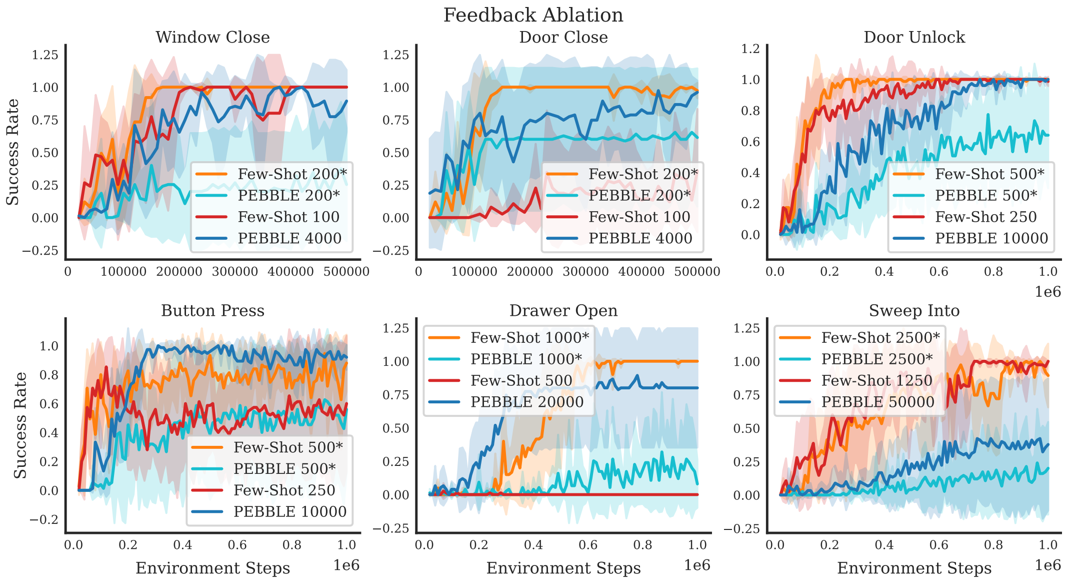

Figure 6: In this ablation, we very the amount of feedback used during training, as indicated by the number in the legend next to each method's name. The "*" indicates the original amount of feedback used in Figure 2. We display results using the same amount of feedback as in [18] for PEBBLE, and using half the amount of feedback for our method. Here we can clearly see that our few-shot method performs better than PEBBLE, even though it uses 20× less feedback. In many tasks, we can half the amount of feedback given to our few-shot method, and still attain the same performance at convergence.

A.2 Plots of Feedback versus Performance

We originally chose to display environment steps on the X-axis of Figures 2 and 3 as was done in prior work [18, 15]. Plotting the environment steps shows the ultimate convergence behavior of each method, as feedback is stopped before the end of training. It also allows us to show SAC on the same graph. Here, we provide versions of Figures 2 and 3 that have the amount of total feedback given on the X-axis. These plots display the same overall trends – our few-shot method out-performs baselines for the amount of feedback provided.

A.3 Locomotion Experiments

We evaluate our few-shot preference learning method on a locomotion task, Cheetah Velocity, from Finn et al. [49] to show its broad applicability, particularly in settings where the agent's goal is temporal and cannot be encapsulated by an environment configuration. The agent is rewarded for moving at a particular unseen target velocity, 1.5m/s. We use 10 other velocities for pretraining. Figure shows our method and PEBBLE using different feedback schedules, with the total feedback

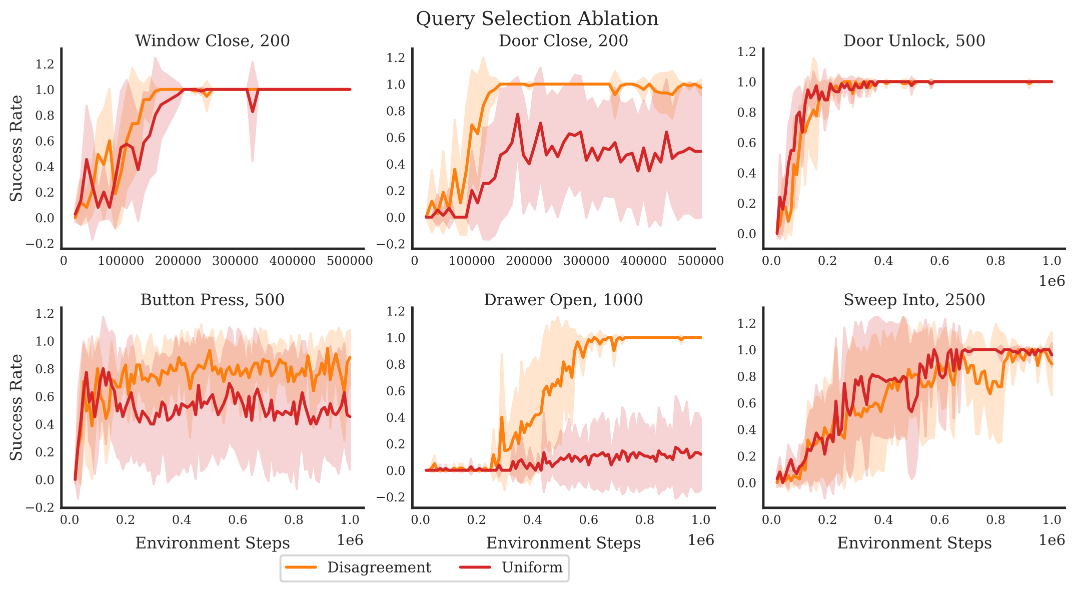

Figure 7: Here we compare using the disagreement query sampling technique versus uniform random query sampling in the MetaWorld environments. We see that for some environments, disagreement sampling is important, but for others it does not have a large effect.

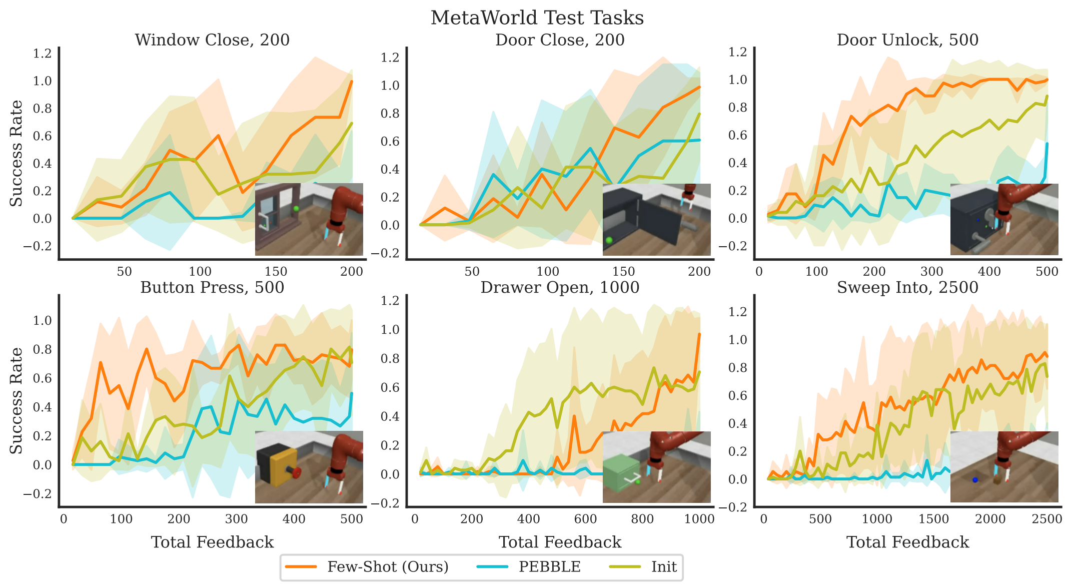

Figure 8: Learning curves for the MetaWorld environments where the x-axis is chosen to be the total feedback given to the agent over the course of training. Note that policies were trained for a bit after all feedback was given, and thus final convergence is not demonstrated as well in this figure, as in Figure 2. In environments where policies obtained decent performance before all feedback was given we were able to further reduce the amount of feedback in the ablation shown in Figure 6.

provided on the X-axis. Each plot corresponds to training over five-hundred thousand environment steps. We find that our method converges after only around 100 queries independent of the feedback schedule, while PEBBLE is unable to attain close to the same performance even with 1000 queries. The "init" baseline described in Section 3 performs similarly, but has slightly worse asymptotic convergence for 2 of 3 feedback schedules. We do minimal hyper-parameter tuning in these environments, and believe the performance of our approach could be further improved. Overall, we find that trends from manipulation environments hold, our few-shot method is able to quickly learn the ground truth reward function.

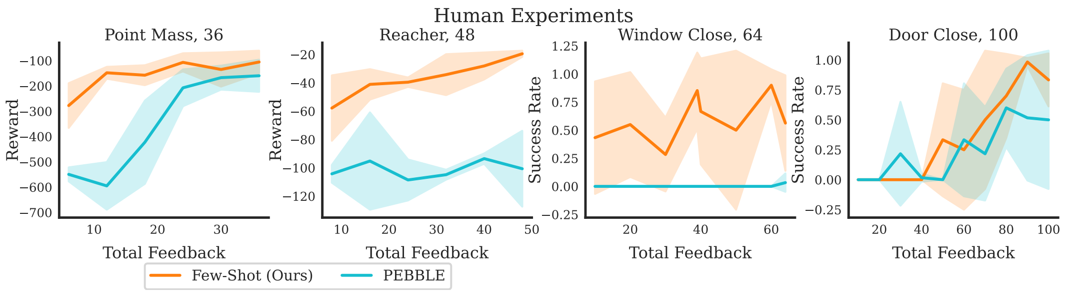

Figure 9: Learning curves for the human user experiments where the x-axis is chosen to be the total feedback given to the agent over the course of training. Again for final convergence, please refer to Figure 3.

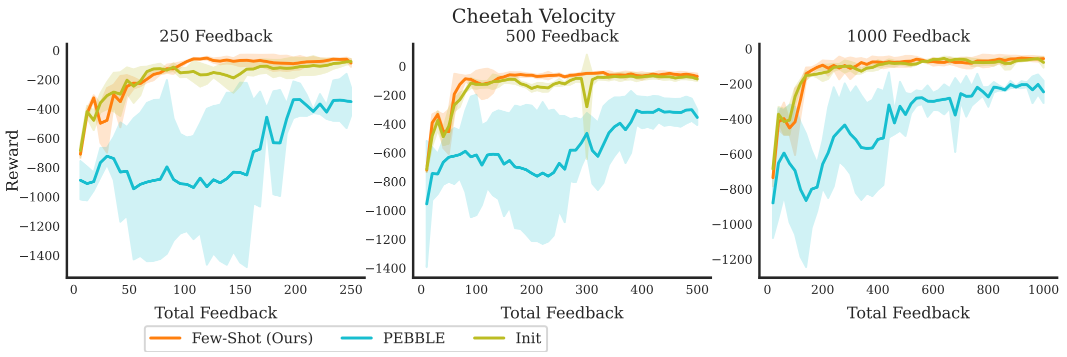

Figure 10: Learning curves for the Cheetah Velocity experiment. The X-axis is given as the amount of feedback provided over 500k environment steps. Each subplot corresponds to a different feedback schedule.

A.4 Comparison with Example Based Methods

One proposed alternative to inferring reward functions via preferences, is inferring them using examples of "success" states to learn reward functions [8] or directly develop new RL algorithms [63]. While such methods have shown success in their chosen domains, they have a number of drawbacks in comparison to preference based methods. First, example based methods often implicitly assume that the underlying reward function for a task is reaching a goal state. While this is amendable to some tasks, it can preclude objectives that cannot easily be classified as satisfying a goal condition. This is particularly evident for tasks that are temporal in nature, like driving, where we might care about intermediate safety and comfort, not just the final destination. For the aforementioned cheetah locomotion task, it might be difficult for humans to provide examples of successful "running" states without a pre-existing oracle policy. While we can easily provide a target velocity, it is difficult to provide target joint positions etc. for a different embodiment. Second, example based methods often optimize sparse-like rewards given for satisfying some learned condition, causing optimization difficulties as horizon scales. This is not the case for preference based methods, which provide consistent dense rewards.

In order to examine these tradeoffs, we compare our Few-Shot method to Recursive Classification of Examples (RCE) from Eysenbach et al. [63] on two environments using 200 examples or 200 pieces of feedback, though in practice it may be harder to collect examples than preferences. In the Cheetah environment, we examine the effect of example quality on performance by training RCE with states from an expert policy pre-trained with SAC and states from a random policy relabeled to have the target velocity. In a sparse Point Mass Barrier environment, we investigate the impact of horizon and sparsity on example based methods. Results can be found in Figure 11. In the Cheetah Velocity environment, we find that even with access to an expert trajectory, RCE does not attain the same asymptotic performance as our method and takes longer to converge. Having access to such data is unrealistic in the real world, as it is impossible to generate success states from a policy if we have not yet solved the task. Even if we had expert demonstrations, it would then perhaps make more sense to directly apply Inverse RL techniques. When we try to train RCE with just states that have been relabeled to the target velocity and do not contain hard-to-specify joint information, performance completely collapses. In the sparse Point Mass Barrier task, we see that despite the 4-dimensional state space RCE is unable to overcome the difficult exploration and long horizon of the task. As our method uses dense rewards learned from preferences, it is almost able to match the oracle SAC policy. While these tasks may be somewhat toy in nature, they demonstrate key areas in which preference based learning excels: when it may be hard to specify temporal behavior via examples, or when tasks are extremely sparse in nature.

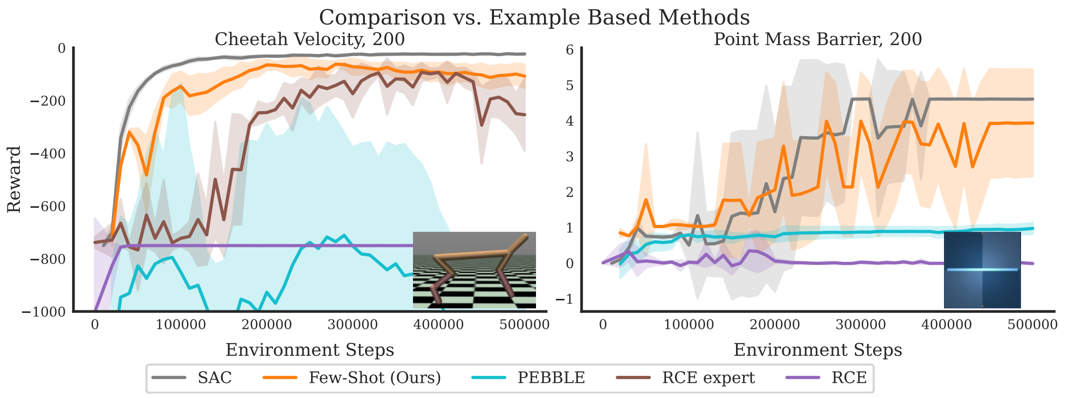

Figure 11: Learning curves for the Cheetah Velocity and Point Maze environment using 200 queries for preference methods and 200 queries for RCE. For the Cheetah environment “expert” denotes that examples were generated using a pretrained oracle policy, otherwise examples were generated by relabeling existing data with the target velocity.

A.5 Human Feedback

In order to better understand the effects of different human users on few-shot preference learning, we compare the performance of four different users on the DM Control reacher task. Each user trained one policy using our Few-Shot method and one policy using PEBBLE. The results are shown in Figure 12. Each users provided 48 preferences for each policy. We find that across all users, our few-shot method out performs PEBBLE. Consistent with results in Figure 3, we did not find a significant difference in the difficulty of providing feedback for this task between our method and PEBBLE, unlike in the MetaWorld tasks. Results on the right hand side of Figure 12 show that when users preferences do not agree with the ground truth reward function as often, performance declines as expected. Our method is relatively robust until query accuracy, or the amount of time the users preferences agreed with the ground truth reward, dropped below 75%. At this point, performance began to decline. While these results indicate that our method is robust to human users, it shows a limitation of our work: if users are unable to accurately provide feedback, reward adaptation will suffer.

A.6 Franka Panda Experiments

Figure 13 shows the learning curves for the Franka Panda models that could not be fit in the main paper due to space constraints.

B Experiment Details

In this section, we enumerate the specifics of the experiments we use to evaluate few-shot preference based RL. As our method requires generating datasets from past experience, we include dataset generation specifics in addition to environment and evaluation details.

B.1 Meta-World

Environments. For the MetaWorld experiments, we adopt the ML10v2 Benchmark for MetaWorld [20]. We keep environments in the “goal unobserved” mode, where the agents must infer the final

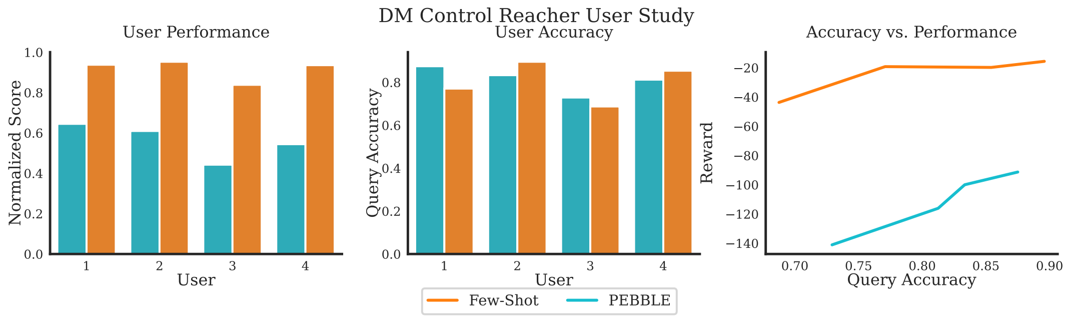

Figure 12: A study of four different users on the DM Control Reacher task. **Left:** The performance of policies trained by each user expressed as a normalized score between a random policy and a fully trained SAC policy on the task. This is computed as (method reward $-$ random reward)/(SAC reward$-$random reward). **Center:** The percentage of each users preferences that aligned with the ground truth reward function for the task. This information was unavailable to the users and is designed to indicate how accurate the human users were. **Right:** A comparison of final ground truth reward against the alignment of the users preferences with the ground truth reward function.

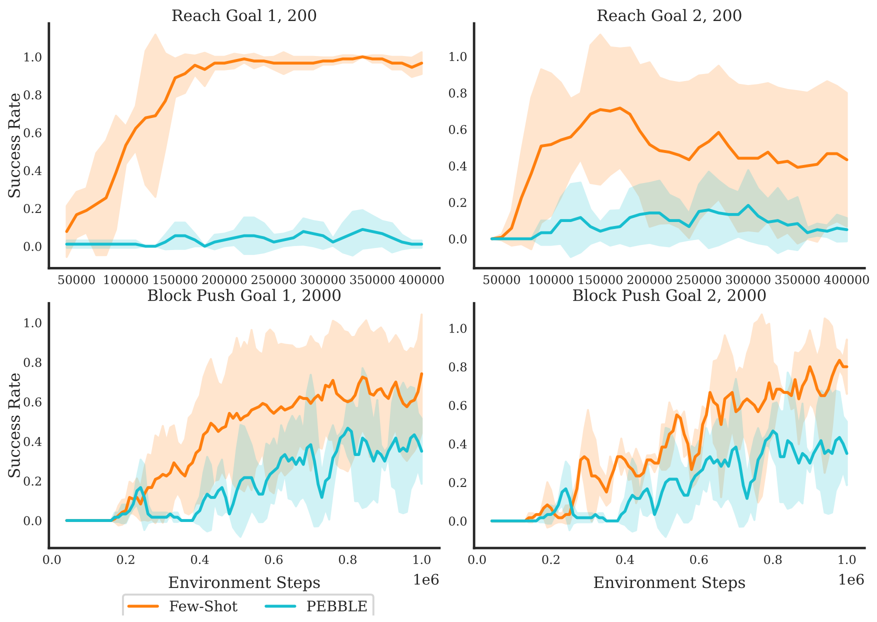

Figure 13: Learning Curves for the Panda experiments in simulation.

desired position of an object (i.e. door handle) from the reward function alone. MetaWorld environments have both parametric and non-parametric task variations. Parameteric variations refer to changes in the initial and final object positions. Non-parametric variations refer to changes in the objects and their desired conditions, like open door vs close window. Because we wanted to directly compare with the hardest environments used in PEBBLE (Sweep Into, Drawer Open, Button press), we slightly modified the set of environments used in ML10 . This amounts to collecting pre-training data on the 10 tasks shown at the top of Figure 2.

Dataset. The datasets for Metaworld are generated by running ground-truth policies from the 10 prior tasks with some additional Gaussian noise. For each of the 10 tasks, we consider 25 parameteric variations, amounting to 250 different reward functions in the training set, though they each belong to one of only 10 overarching categories. For each of these variations, we collect a dataset of (s, a, s', r) tuples by running different policies in the given tasks environment with 0 mean, 0.1 standard deviation Gaussian action noise. Specifically, we run 15 episodes with actions from the expert policy, 25 episodes with actions from parametric variations of the same task family, 10 episodes with actions from the expert policy of completely different task family, and 2 episodes with completely uniform random actions. In order to do this we use the scripted policies provided with the MetaWorld benchmark. From each of these datasets, we sample 6000 queries uniformly at random and assign them labels using the ground truth reward. In summary, we use 10 tasks, each with 25 variations of 6000 queries each.

Evaluation. For MetaWorld, we report the success rate as defined by the MetaWorld benchmark. The test environments are obtained in the same way as PEBBLE, using the standalone versions from MetaWorld. These environments have some parametric variations not included in the prior task environments which makes the test setting slightly more difficult.

B.2 DM Control

Environments. We created custom versions of the standard Point Mass and Reacher environments in DM Control [61]. The default Point Mass environment has a randomly initialized agent attempt to reach the center of a square environment. We modify the point mass environment so that the goal position is randomly chosen, and use the negative L2 distance to the goal as the ground truth reward function. The default sigmoid style reward function would assign zero reward to a large part of the state space, making artificial query generation difficult. For the reacher environment we mask the goal from the observation space an also use the negative L2 distance as the reward function. All other aspects of the state and action space are left the same. The point mass environment terminates when the agent reaches the goal position, and the default time limit of the reacher environment was halved to make learning easier. In both of these environments the task distribution is given by the distribution of unknown goal locations. Additionally when comparing to example-based methods we develop a custom Point Mass Barrier environment on top of the standard point mass. We double the size of the point mass environment in both x and y directions, then place a horizontal barrier at $y = 0$. The task distribution is also given by different goal locations. The ground truth reward is given by the decrease in L2 distance to the barrier crossing point and then the goal location in sequence (max < 2 across the whole trajectory) in addition to a sparse reward of three for reaching the goal. Consequently, the task is considered solved if the agent receives a reward larger than 3. The task distribution is given by goal locations at $y > 0$.

Dataset. For the Point Mass and Reacher DM control environments we use completely randomly generated dataset. For the Point Mass environment we collect 25,000 random time-steps of the environment 16 X-Y goal positions, which include permutations values in the set $\{0, 0.5, -0.5, 1, -1\}^2$. For reacher environment we also collect 25,000 random time-steps of the environment, but over 12 goals each defined by different angle θ and radius r values, include goals at radius one for each of the four cardinal directions, goals at radius 0.66 for the cardinal directions rotated by 45 degrees, and goals at radius 0.33 for the cardinal directions shifted by 22.5 degrees. From each of the tasks datasets we generate 4000 artificial queries for pre-training uniformly at random. For the Point Mass Barrier task we use 10 pretraining tasks. We then sample 40k queries uniformly at random from the replay buffers of agents train with SAC for 100k steps.

Evaluation. We evaluate the point mass environment on the unseen goal of (-0.75, 0.8) and the reacher environment on the unseen goal of (5.5, 0.8). The Point Mass Barrier task is evaluated on the goal (0, 1) at the top middle of the environment.

B.3 Franka Panda

Environments. We design two tasks for the Franka robot. For both tasks we use end-effector delta control, ie the agent chooses x, y, z deltas for the end effector to move to. The first task is the Reach task, where the robot is tasked with simply moving its end-effector towards a target goal position g. The reward function is again the negative L2 distance to the goal position, or $-||e - g||_2^2$ where e is the absolute position of the end effector. The second task is a block pushing task where the agent wants to push a block from a randomized starting location to a fixed goal position g. The reward function for this task is $-0.1||e - b||_2^2 - ||b - g||_2^2$ where e is defined as before and b is the absolute position of the center of the block. The goal positions always have a z value of half the block's height. The agent observes the (x, y) position of the block, but does not know the goal location. The block is 5cm across. Again the task distribution for both environments is given by the distribution of unknown goal locations. We use the PyBullet simulator for our training environments. When transferring the policies to the real world, use two Intel Realsense cameras and OpenCV Aruco tag tracking to compute the estimated (x, y) position of the center of the block. An image of our setup can be found in Figure 14. We also add zero mean, 0.001 standard deviation noise to the state to aid in sim to real transfer. For the Reach task we define success as being within 2.5cm of the goal and for the Block push task we define it to be within 5 cm.

Dataset. We generate behavior datasets for the reach task by simply collecting random rollouts of 10,000 timesteps for 75 randomly sampled goals. We generate behavior datasets for the block push task by training polices to push blocks to 16 different locations, then applying a similar strategy to the MetaWorld environments: for each task we run 8 random episodes, 50 expert episodes, and 5 episodes using actions from each of the other tasks (80 total), all with zero mean standard deviation 0.3 Gaussian noise. Unlike in meta-world, we did not spend time tuning data generation for the Panda experiments. We then generated 6000 artificial queries for each of the 75 reach tasks, and 20,000 artificial queries for each of the 16 block pushing tasks, leaving one out for validation.

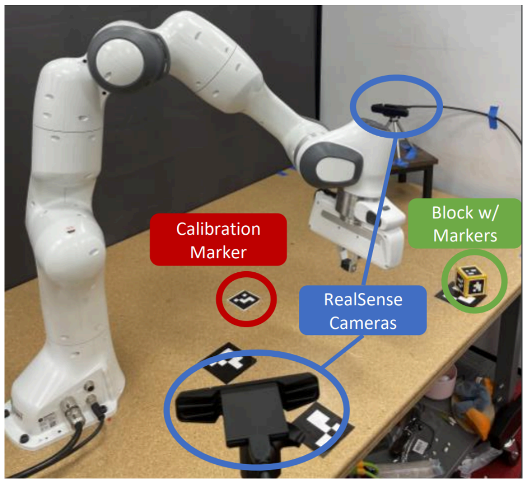

Figure 14: Depiction of the real world robot setup with a Franka Panda arm. We use ArUco tags for tracking the position of objects in combination with Intel RealSense cameras. For the reach task, the robot just needs to move its end effect to a target position. For the block push task, the marked block must be moved to a specific location. The blocks position is computed using two Intel RealSense Cameras.

Evaluation. We evaluate each of the policies by transferring them from simulation to a real Franka-Panda robot. For control, we use the PolyMetis library [64]. We train policies on two different unseen goal positions, which are listed in Table 1. We evaluate each run of the reach task using four initial robot configurations and each run of the Block Push task using four initial block locations. Results are reported in final meters to the goal. We found that the second block push location of (0.35, -0.3) was much easier for the robot regardless of method. This is likely because block state estimation was more accurate on that side of the table due to the camera setup.

B.4 Locomotion

Environments. We take the Cheetah Velocity environment from Finn et al. [49], but use a horizon of length 500. The ground truth reward function is given by $-|v - \text{target}| - ||a||_2^2$, where "target" is a target velocity. Thus, the agent is rewarded for running at a certain speed, and we vary the target speed across tasks. Unlike in manipulation environments, reward functions for locomotion environments cannot be specified through any type of "goal condition" as behavior across time matters.

Dataset. We generate behavior datasets by taking the replay buffers of policies trained with SAC for 150k environment steps using different target velocities in increments of 0.25m/s, starting with 0.25m/s and ending with 2.75 m/s. We leave out 1.5m/s for the test , making for 10 total training tasks, which is far less than the upwards of 100 training tasks used in Finn et al. [49]. We generate 40k artificial queries uniformly at random from each replay buffer for the training dataset.

Evaluation. We evaluate all approaches on the unseen velocity of 1.5m/s.

B.5 Human Experiments

Here we provide an overview of the procedure used in our human experiments in Section 4.2. We use a single expert human subject for experiments in Figure 3, who was familiar with preference based RL and both the MetaWorld and DM Control benchmarks. The user completed experiments on PointMass, Reacher, Window Close, and Door Close in that order. The human results in Figure 12

Table 2: Hyperparameters used for pre-training with the MAML Algorithm.

Parameter	**Value**
Outer LR	0.0001
Inner LR	0.001
Support Set Size	32
Query Set Size	32
Task Batch Size	4
Learn Inner LR	True
Ensemble Size	3
Reward Arch	3x 256 Dense
Activation	ReLU
Output Activation	Tanh
Segment Size	25 (MW, FP, C), 10 (DM)

are from three additional users familiar with learning for robotics, who followed the same procedure. Each environment required training four policies – two for PEBBLE and two for our Few-Shot method. The user trained all four policies in parallel on a single computer with a user interface that looked similar to the query visualizations shown in Figure 5. As feedback was elicited intermittently through the course of training, we cannot fully separate the time it took for users to answer queries with the time used to train the policy. However, we know that the total time before all queries were answered was around 22 minutes for Point Mass, around 28 minutes for Reacher, around 45 minutes for Window Close, and around 1 hour for Door Close. Whenever the user could not make a determination about the query, they were asked to skip it. We count skip queries in the total feedback budget and measured the practicality of the user interactions by the number of such skip queries as shown in Figure 3. There we see that human users did not need to skip queries that frequently, and were able to be relatively accurate with respect to the ground truth reward function. Moreover, we found that in the more difficult environments, the human user skipped fewer queries and was more accurate when training a policy using our few-shot method. This is backed up by the visualizations in Appendix D, which qualitatively demonstrates that the few-shot method asks easier to distinguish queries in the robotics environments, likely due to pretraining.

C Hyperparameters

In this section, we detail the hyper-parameters used for our method and baselines. We first give hyperparameters used in pre-training, then provide the hyperparameters used for online experiments. In the following tables we use MW for MetaWorld, DM for DM Control, and FP for Franka Panda. For MetaWorld artificial feedback experiments, we run five random seeds for each method. For human feedback experiments we run two seeds for each method, as it takes a large amount of time to collect human feedback. For real world experiments, we run four seeds for each reaching task, and two seeds for each block pushing task for 8 and 4 seeds total, respectively.

Pretraining. We use the MAML algorithm in combination with the Adam Optimizer. We used learned inner learning rates as in Antoniou et al. [65].

Online Adaptation. Here we list the hyperparameters and network architectures used for SAC, PEBBLE, and our method in Table 3. In comparison to the original PEBBLE algorithm, we change the segment size to 25 and increased the reward frequency. We found that these changes improved performance for PEBBLE as well. We also train reward models until they achieve 95% accuracy, instead of training them for a fix number of epochs or until they reach 97% accuracy as done in the PEBBLE codebase. We run a maximum of 40 MAML adaptation steps. If at that point the reward model has not reached 95% accuracy, we train it again with the Adam Optimizer. For all methods we did not run unsupervised exploration prior to beginning training. While unsupervised exploration leads to improvements in locomotion environments as shown in Lee et al. [18], we found that it did not offer a large improvement in robotics environments. This is likely because a sufficient portion of the state space can be explored quickly in locomotion environments like Cheetah and Quadruped, but not in MetaWorld, where task are longer horizon and require both reaching and interacting with specific parts of the state space. For all runs we use a constant feedback schedule, ie the same amount of feedback each session. We list the exact feedback specifications in Table 4. Feedback

Table 3: Hyper-parameters for preference learning algorithms.

Parameter	**Artificial Feedback**	**Human Feedback**
Init Temp	0.1	0.1
Discount	0.99	0.99
EMA τ	0.995	0.995
Learning Rate	0.0003	0.0003
Target Update Freq	2	2
(β_1, β_2)	0.9, 0.999	0.9, 0.999
Actor and Critic Arch	3x 256 Dense MW, FP, C	2x 256 DM, 3x 256 Dense MW
Actor and Critic Activation	ReLU	ReLU
SAC Batch Size	512	512
Reward Net Batch Size	256	256
Disagreement Sample Multiplier	10	10

Table 4: Specific feedback schedule for each environment. For all environments, the first session always sampled queries at uniform. For the MetaWorld human experiments, the first half of all queries were asked uniformly at random.

Environment(s)	**Max Feedback**	**Feedback Per Session**	**Session Frequency** (K)
Window Close, Door Close	200	8	5000
Door Unlock, Button Press	500	8	5000
Drawer Open	1000	10	5000
Sweep Into	2500	20	5000
Point Mass (Human)	36	6	20000
Reacher (Human)	48	8	20000
Window Close (Human)	64	8	10000
Door Close (Human)	100	10	10000
Reach Panda	200	8	5000
Block Push Panda	2000	20	5000
Cheetah Velocity (vs. RCE)	200	4	6000
Cheetah Velocity	250	3	5000
Cheetah Velocity	500	5	5000
Cheetah Velocity	1000	10	5000
Point Mass Barrier	200	5	10000

schedules used in the ablation experiments in Appendix A were constructed by multiplying the "Max Feedback" and "Feedback per Session" values by 20 for PEBBLE and 0.5 for our method.

RCE. For our comparisons against RCE in Figure 11, we left all parameters at their defaults. Example states for the Cheetah Velocity environment were given via an expert demonstration, or by relabeling random states with the target velocity. Example states for the Point Mass Barrier environment were created by sampling positions within the target location with feasible velocities.

D Additional Visualizations

Here we provide select queries shown to users when training from real human feedback using our Few-Shot method. We compare queries asked by each method at the same point in training. The set of nearly all queries used to train agents from human feedback is included in the supplementary material download on OpenReview. Note that the segment size used in MetaWorld was 25, but we showed users every other frame as the changes between individual frames were minimal. In each figure the trajectory segment with the check mark was selected by the user.

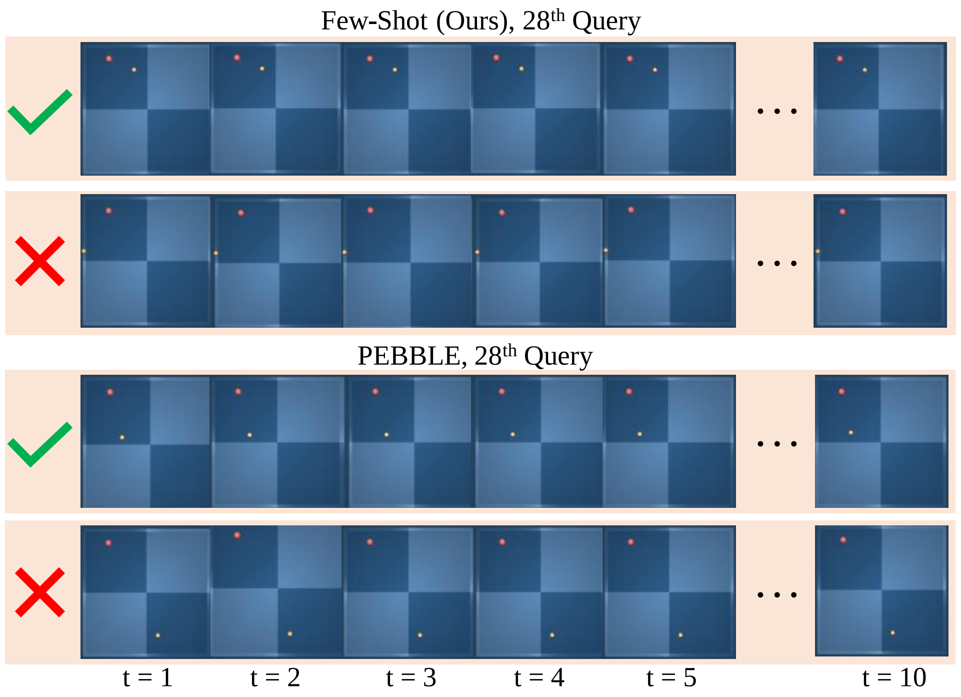

Figure 15: A depiction of the 28th query asked to users when training the Point Mass Agent from human feedback. The winning query was chosen based on proximity of the agent (yellow) to the goal position (red). At this point in training, our Few-Shot method sampled queries closer to the goal position than PEBBLE.

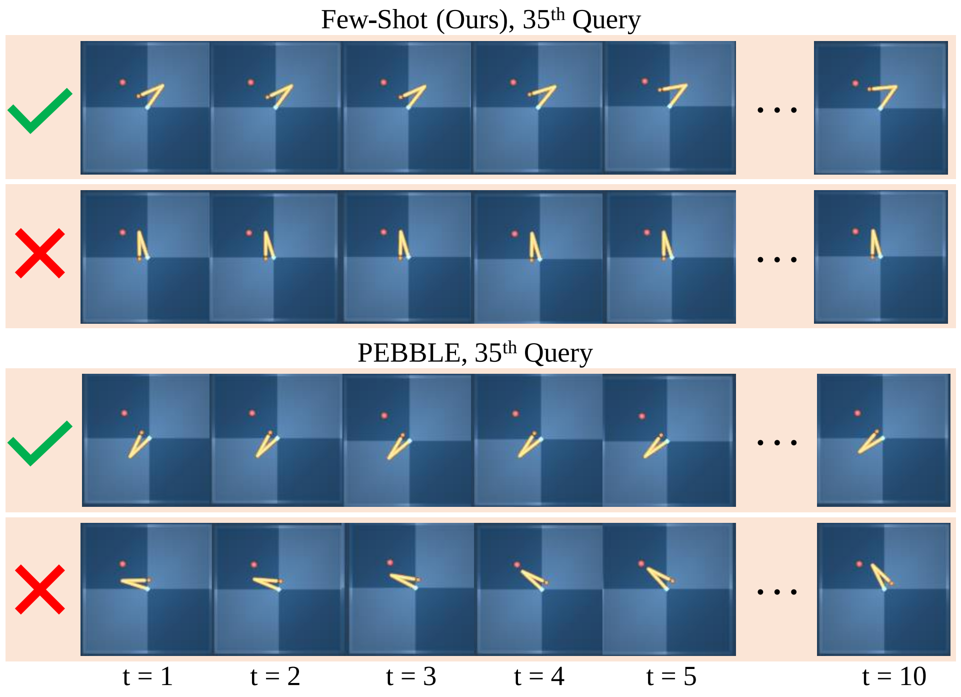

Figure 16: A depiction of the 35th query asked to users when training the reacher from human feedback. Our method's query (top) was easier to answer because the top trajectories' arm was clearly closer to the target position.

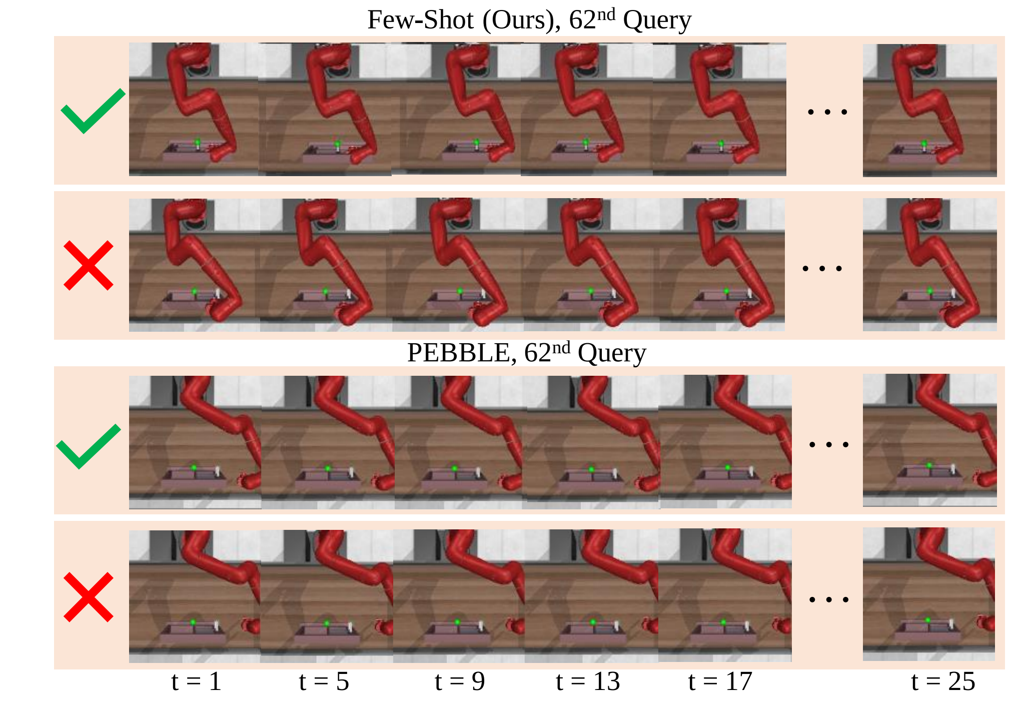

Figure 17: This shows one of the last queries asked for the Window Close environment. Here we see that our method's query asks the user to choose between a closed and unclosed window (top), while PEBBLE asked the user to choose between two different, hard to distinguish, arm positions.

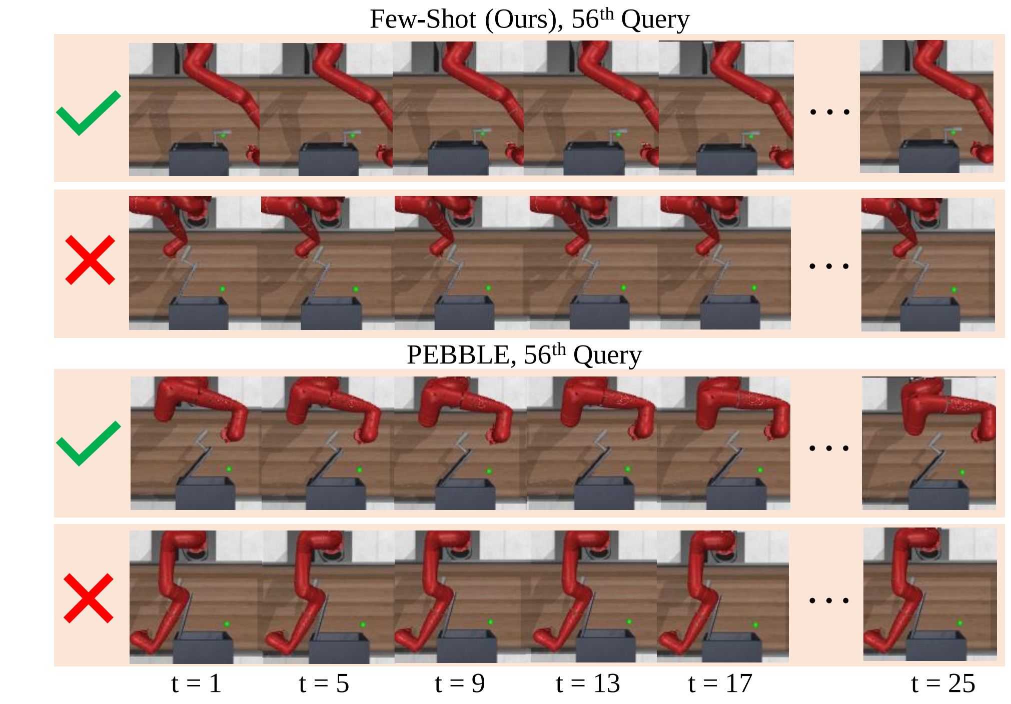

Figure 18: This shows a query towards the middle of training for the Door Close environment. At this point, the few-shot method is asking the user to compare a completely closed door (better) versus an open one, while PEBBLE's query only includes a partially closed door.