Leveraging Sub-Optimal Data for Human-in-the-Loop Reinforcement Learning

Calarina Muslimani[1] , **Matthew E. Taylor**[1,2]
[1]University of Alberta
[2]Alberta Machine Intelligence Institute (Amii)
{musliman, matthew.e.taylor}@ualberta.ca

Abstract

To create useful reinforcement learning (RL) agents, step zero is to design a suitable reward function that captures the nuances of the task. However, reward engineering can be a difficult and time-consuming process. Instead, human-in-the-loop (HitL) RL allows agents to learn reward functions from human feedback. Despite recent successes, many of the HitL RL methods still require numerous human interactions to learn successful reward functions. To improve the feedback efficiency of HitL RL methods (i.e., require less feedback), this paper introduces Sub-optimal Data Pre-training, SDP, an approach that leverages reward-free, sub-optimal data to improve scalar- and preference-based HitL RL algorithms. In SDP, we start by pseudo-labeling all low-quality data with rewards of zero. Through this process, we obtain "free" reward labels to pre-train our reward model. This pre-training phase provides the reward model a head start in learning, whereby it can identify that low-quality transitions should have a low reward, all without any *actual* feedback. Through extensive experiments with a simulated teacher, we demonstrate that SDP can significantly improve or achieve competitive performance with state-of-the-art (SOTA) HitL RL algorithms across nine robotic manipulation and locomotion tasks.

1 Introduction

In reinforcement learning (RL), an agent's goal is to interact with an environment to maximize its total reward [Sutton and Barto, 2018]. It is assumed that the environment provides an agent with a well-defined reward function that captures all task complexities. But where does this reward function actually come from? Reward functions are hand-engineered by humans in what often can be a tedious and non-trivial pursuit [Booth *et al.*, 2023]. As the complexity of tasks increases, so does the time and effort required to design a suitable reward function. Further, there have been notable examples of reward misspecification, in which RL agents discovered and exploited unintended shortcuts in the reward function [Skalse *et al.*, 2022]. One notorious example is the CoastRunners game, in which the objective should be to finish a boat race as fast as possible. Instead, an RL agent gained the most reward by spinning its boat in a circle despite concurrently catching on fire and crashing into other boats [Clark and Amodei, 2016].

A promising alternative is to learn reward functions directly from human feedback. In this paradigm, humans can provide feedback in the form of preferences or scalar signals, which can then used to learn a reward function that is consistent with human desires [Daniel *et al.*, 2014; Christiano *et al.*, 2017; Lee *et al.*, 2021; White *et al.*, 2023]. Despite recent progress, existing preference- and scalar-based HitL RL methods still suffer from high human labeling costs that can require thousands of human queries to learn an adequate reward function [Christiano *et al.*, 2017]. This may invalidate the original goals of human-in-the-loop for reward design. Prior work attempts to mitigate this issue through several mechanisms, including active learning [Settles, 2009; Lee *et al.*, 2021], data augmentation [Shorten and Khoshgoftaar, 2019; Park *et al.*, 2022], semi-supervised learning [Zhu, 2005; Park *et al.*, 2022], and meta-learning [Muslimani *et al.*, 2022; Hejna III and Sadigh, 2023].

Alternatively, this work takes inspiration from data sharing in offline RL, in which the goal is to leverage prior data to improve performance on a given target task. In settings where there is an abundance of unlabeled (i.e., reward-free), low-quality data, one way to use this data is to simply label all such data with a reward of zero (i.e., the minimum reward for the task) [Yu *et al.*, 2021]. This simple approach achieved promising results on a variety of robotic tasks in the offline RL setting. As mediocre or low-quality data is arguably the easiest type of data to obtain, and prior work has demonstrated how to leverage it in offline RL, this paper asks the question:

Can we leverage abundant sub-optimal, unlabeled data to improve learning in HitL RL methods?

To that end, we present Sub-optimal Data Pre-training, *SDP*, a tool for HitL RL algorithms to increase human feedback efficiency. SDP leverages sub-optimal target task data by pseudo-labeling all transitions with the minimum reward for the task, which we assume to be zero (without loss of generality). The now pseudo-labeled sub-optimal data is used in two ways. First, we pre-train a regression-based reward model by applying standard supervised learning to minimize

the mean squared loss. Intuitively, the pre-training component provides a "free" head start to the reward model, whereby the reward model can bias these low-quality transitions to have a lower reward value than other transitions. Second, we initialize the RL agent's replay buffer with the sub-optimal data and make learning updates to the RL agent. This process changes the RL agent's policy and provides new behaviors for the human to provide feedback on, relative to learning with no initial sub-optimal data. This ensures that when it is time for the human teacher to provide feedback, they will not be providing redundant feedback to the existing sub-optimal data. Afterward, we follow standard preference- or scalar-based HitL RL.

This paper's core contribution is showing that we can harness the availability of low-quality, reward-free data for HitL RL approaches by pseudo-labeling it with zero rewards and treating it as a free bias for learning reward models. Extensive experiments combining SDP with both scalar- and preference-based reward learning RL algorithms show that the addition of SDP can significantly improve feedback efficiency across complex tasks from both DeepMind Control (DMControl) [Tassa *et al.*, 2018] and Meta-world [Yu *et al.*, 2020] suites. We additionally show that SDP is not limited to leveraging sub-optimal data from the target task — sub-optimal data from different tasks can also be useful, highlighting the generality of SDP. Overall, this work takes an important step toward considering how HitL RL approaches can take advantage of readily-available sub-optimal data.

2 Related Work

This section highlights related work in HitL RL and in leveraging sub-optimal data.

2.1 Human-in-the-Loop RL

Several approaches in HitL RL allow agents to leverage human feedback to adapt or learn new behavior. Learning from demonstration (LfD) [Argall *et al.*, 2009] is one such methodology that allows a human to provide examples of desired agent behavior. Human demonstration data has been used to shape the environment's reward function [Brys *et al.*, 2015], develop a reward function from scratch [Abbeel and Ng, 2004], and bias the agent's policy towards certain actions [Taylor *et al.*, 2011]. Although demonstrations can be a rich source of feedback, they are often expensive to obtain and may require domain experts [Dragan and Srinivasa, 2012].

Another approach is learning from preference-based feedback where a teacher provides preferences between two or more sets of agent behavior [Christiano *et al.*, 2017]. Preference learning has been popularized in recent years as it can require less effort and expertise compared to providing demonstrations. To further reduce the amount of human interaction required, several strategies have been introduced. Recent work has combined preferences with demonstrations [Ibarz *et al.*, 2018; Bıyık *et al.*, 2022], used unsupervised pre-training for policy initialization [Lee *et al.*, 2021], integrated semi-supervised learning and data augmentation techniques [Park *et al.*, 2022], applied uncertainty-based exploration strategies [Liang *et al.*, 2022], and leveraged labeled data from multiple tasks via a meta-learning approach [Hejna III and Sadigh, 2023]. Despite its popularity, some argue that comparison feedback might not capture the full intricacies of human preferences, as oftentimes the human is limited to choosing between two options [Daniel *et al.*, 2014; White *et al.*, 2023].

As a result, another body of work focuses on learning from scalar feedback where human teachers can provide scalar signals to evaluate an agent's behavior [Knox and Stone, 2009; Griffith *et al.*, 2013; Loftin *et al.*, 2016; MacGlashan *et al.*, 2017; Warnell *et al.*, 2018; White *et al.*, 2023]. Several works use scalar feedback to learn a reward model via regression [Daniel *et al.*, 2014; Cui and Niekum, 2018; Cabi *et al.*, 2020]. Similarly, other works learn a human reinforcement function via regression [Knox and Stone, 2009; Knox and Stone, 2013; Warnell *et al.*, 2018]. In this setting, a human is assumed to provide scalar feedback that is representative of a behavior's long-term value. A human reinforcement function is distinct from a reward function, and more closely resembles an action-value function in RL.

Despite the growing advances in HitL RL, there is no work, to the best of our knowledge, in understanding how reward-free, sub-optimal data can be leveraged to improve the feedback efficiency of preference- and scalar-based approaches.

2.2 Learning from Sub-Optimal Data

SDP aims to leverage sub-optimal data for scalar- and preference-based HitL RL algorithms. However, learning from low-quality data or negative examples has been applied in other areas of reinforcement learning and imitation learning [Chen *et al.*, 2021; Tangkaratt *et al.*, 2021]. In standard RL, several works use sub-optimal demonstrations to initialize a policy [Taylor *et al.*, 2011; Hester *et al.*, 2018; Gao *et al.*, 2019]. In offline RL, some approaches leverage sub-optimal transitions from multiple tasks and either assign reward labels according to the reward function of the target task [Singh *et al.*, 2020] or simply label them the minimum possible environmental reward [Yu *et al.*, 2022]. In goal-conditioned RL, Hindsight-Experience-Replay (HER) [Andrychowicz *et al.*, 2017] uses failed episodes by treating them as a success with respect to a different goal. In inverse reinforcement learning (IRL) [Arora and Doshi, 2021], a constrained optimization formulation has been proposed that can accommodate both successful and failed demonstrations [Shiarlis *et al.*, 2016]. Trajectory-ranked Reward Extrapolation (T-REX) [Brown *et al.*, 2019] can leverage sub-optimal demonstrations in IRL, making use of ranked demonstrations to learn a reward function. Later work improves upon T-REX by learning a policy via behavioral cloning on the demonstrations. They then automatically generate ranked trajectories by adding increasing amounts of noise to the learned policy [Brown *et al.*, 2020].

3 Background

In the RL paradigm, agents interact with an environment with the goal of maximizing a scalar reward signal. This interaction process is modeled as a Markov Decision Process (MDP) which consists of $\langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$. At every time-step t, the agent receives a state $s_t \in \mathcal{S}$ from the environment and

chooses an action $a_t \in \mathcal{A}$. The environmental transition function, T, determines the probability of transitioning to state s_{t+1} and receiving reward r_{t+1}, given the agent was in state s_t and executed action a_t. The environment then provides the agent with this scalar reward r_{t+1}. The agent attempts to learn a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that maximizes the return $G = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, which is defined as the expected sum of discounted future rewards with discount factor $\gamma \in [0, 1)$.

3.1 Reward Learning from Human Feedback

This paper assumes that we are in a reward-free paradigm, an MDP/R setting, where our goal is to learn a good policy while also learning a reward function from human feedback. We follow the standard reward learning framework that uses supervised learning to learn such a reward function, $\hat{r}_\theta$ [Daniel *et al.*, 2014; Christiano *et al.*, 2017]. In both scalar- and preference-based feedback settings, we consider segments of trajectories σ. Each segment consists of a sequence of states and actions $\{s_t, a_t, s_{t+1}, a_{t+1}, ..., s_{t+k}, a_{t+k}\}$.

Preference-based Reward Learning In preference-based learning, two segments, σ^0 and σ^1, are compared by a teacher, yielding $y \in \{\ 0, .5, 1\}$. Specifically, if the teacher preferred segment σ^1 over segment σ^0, then y is set to 1, and if the converse is true y is set to 0. If both segments are equally preferred, then y is set to .5. As feedback is collected, it is stored as tuples (σ^0, σ^1, y) in the reward model (RM) data set D_{RM}. Then, we follow the Bradley-Terry model [Bradley and Terry, 1952] to define a preference predictor using the reward function $\hat{r}_\theta$:

$$P_\theta(\sigma^1 > \sigma^0) = \frac{\exp(\sum_t \hat{r}_\theta(s_t^1, a_t^1))}{\sum_{i \in \{0,1\}} \exp(\sum_t \hat{r}_\theta(s_t^i, a_t^i))} \qquad (1)$$

In general, if $\sigma^i > \sigma^j$ in equation 1, then the segment σ^i is preferred by the teacher over segment σ^j. Intuitively, this model assumes that the probability of the teacher preferring a segment depends exponentially on the total sum of predicted rewards along the segment. To train the reward function, we can use supervised learning where the teacher provides the labels y. More specifically, we update $\hat{r}_\theta$ by minimizing the standard binary cross-entropy objective:

$$L^{CE}(\theta, D) = -\, E_{(\sigma^0, \sigma^1, y) \sim D}\big[(1 - y)\, \log P_\theta(\sigma^0 > \sigma^1) + \ y\, \log P_\theta(\sigma^1 > \sigma^0)\big] \qquad (2)$$

Scalar-based Reward Learning The primary difference between scalar and preference-based reward learning is that in scalar-based learning, the human teacher assigns numerical ratings to segments of trajectories one at a time. In this setting, the comparisons between segments are implicit. More concretely, a teacher assigns a scalar value y to a segment σ^i, and as feedback is collected, it is stored as tuples (σ^i, y) in the reward model data set D_{RM}. We then apply standard regression and update $\hat{r}_\theta$ by minimizing the mean squared error:

$$L^{MSE}(\theta, D) = E_{(\sigma^i, y) \sim D}\big[(y - \hat{r}_\theta(\sigma^i))^2\big] \qquad (3)$$

4 Sub-optimal Data Pre-training

In this section, we present SDP, a tool to improve the feedback efficiency for HitL RL. We take the approach of pseudo-labeling all transitions from a set of sub-optimal demonstrations with the lowest possible environmental reward (we assume to be zero). The goal of SDP is then to use this pseudo-labeled data to create a pessimistic prior for rewards models in HitL RL methods (see Figure 1). Its simplicity enables SDP to be used in conjunction with any off-the-shelf HitL RL algorithm that learns a reward function from feedback. Algorithm 1 contains the complete pseudocode.

4.1 Pessimistic Prior for Reward Learning

SDP comprises two phases: (1) the reward model pre-training phase and (2) the agent update phase. In the reward model pre-training phase, we first gather a data set, D_{sub}, of N sub-optimal state, action transitions. We then pseudo-label all transitions in D_{sub} with rewards of 0, resulting in $D_{\text{sub}} = \{s_i, a_i, 0\}_{i=1}^{N}$. D_{sub} is then used to optimize the reward model $\hat{r}_\theta$ with the mean squared loss in Equation 3 (see lines 2-4 in Algorithm 1). As a result, the reward model $\hat{r}_\theta$ becomes pessimistic because it learns to associate all sub-optimal transitions with a low reward. Without such a prior, the reward model initially has random estimates for the sub-optimal transitions. The only way to improve such estimates is to obtain teacher feedback. Therefore, by pseudo-labeling the sub-optimal transitions with 0, we obtain a helpful reward bias before receiving any teacher feedback.

Next, in the agent update phase, we initialize the RL agent's replay buffer D_{agent} with D_{sub} (see line 6 in Algorithm 1). The RL agent then briefly interacts with its environment and performs gradient updates according to its loss functions (see lines 7-12 in Algorithm 1). The agent update process changes the RL agent's policy and generates new transitions, which are then stored in both the agent's replay buffer D_{agent} (see line 9 in Algorithm 1) and the reward model's data set D_{RM} (see line 10 in Algorithm 1). It is important to note that in standard scalar- and preference-based reward learning, we query the teacher for feedback on trajectory segments sampled from D_{RM}. Therefore, adding new transitions into D_{RM} during the agent update phase is necessary to ensure that the teacher does not provide redundant feedback to the original sub-optimal transitions (as D_{RM} was empty prior to the agent update phase). When it is time for the teacher to provide their first set of feedback (see line 13 in Algorithm 1), the feedback can cover a different region of the state and action space, relative to the original sub-optimal data. We empirically show that the agent update phase changes the RL agent's policy by performing policy rollouts and analyzing the differences in state distributions (see Figure 5 in Appendix B).

One may immediately think to ask: is labeling sub-optimal transitions with an incorrect reward problematic? Using incorrect reward labels does induce some incorrect bias for both the reward model and the RL agent's value network. However, as the transitions are sub-optimal, the bias for using an incorrect reward is low and does not greatly influence learning. Figure 6 in Appendix B shows that the majority of the reward values for sub-optimal transitions gathered through a random policy do have true rewards close to 0. In addition,

by using the sub-optimal transitions, we increase the overall amount of data used by both models, which can *decrease* the models' variance [Yu *et al.*, 2022].

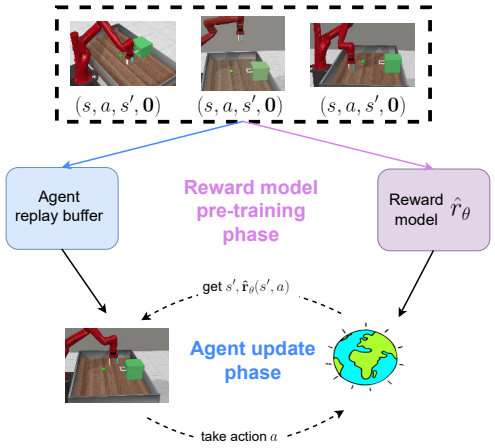


Figure 1: Overview of SDP: After obtaining sub-optimal target task data, we pseudo-label this data with rewards of zero. We then pre-train our reward model $\hat{r}_\theta$ using this data set. During the agent update phase, we initialize our RL agent's replay buffer with the same pseudo-labeled data set. We then interact in the environment and make learning updates to obtain new behaviors for a teacher to give feedback.

Algorithm 1 SDP

Require: Reward model $\hat{r}_\theta \leftarrow \theta$ randomly initialized, Reward model data set $D_{\text{RM}} \leftarrow \emptyset$, RL agent with replay buffer $D_{\text{agent}} \leftarrow \emptyset$, Sub-optimal data set D_{sub} with reward labels 0

// REWARD MODEL PRE-TRAIN PHASE
for each gradient step **do**
    Optimize $\hat{r}_\theta$ on D_{sub} with L^{MSE} (equ. 3)
end for
// AGENT UPDATE PHASE
$D_{\text{agent}} \leftarrow D_{\text{sub}}$
for each time-step t **do**
    Collect s_{t+1} by taking action $a_t \sim \pi(s_t)$
    Store $(s_t, a_t, \hat{r}_\theta, s_{t+1})$ in D_{agent}
    Store (s_t, a_t) in D_{RM}
    Update RL agent with D_{agent}
end for
Begin scalar- or preference-based reward learning using pre-trained $\hat{r}_\theta$, RL agent, and D_{agent}, D_{RM}

4.2 Implementation Details of SDP

For ease of obtaining sub-optimal data for SDP, we used state, action transitions from a random policy. We further note that we do not require explicit access to a sub-optimal policy; we only require state, action transitions from said policy. For all the experiments in Section 5.2, we used 50,000 transitions. Moreover, all reward model hyperparameters remained the same during both SDP phases (i.e., the reward model pre-train and agent update phases) as well as during the standard scalar- or preference-based reward learning that followed.

5 Experiments

This section considers the following five research questions:

- In the low feedback regime, can SDP improve upon existing scalar- and preference-based HitL RL methods?
- Can SDP effectively leverage sub-optimal data from tasks other than the target task?
- What is the contribution of each phase of SDP?
- How does the amount of sub-optimal data affect the performance of SDP?
- How does the amount of feedback affect the performance of SDP?

5.1 Experimental Design

To demonstrate the generality and effectiveness of SDP, we apply SDP to (1) scalar-based and (2) preference-based HitL RL approaches. For the scalar-based experiments, we combine SDP with R-PEBBLE (a regression variant of PEBBLE [Lee *et al.*, 2021]). We compare SDP + R-PEBBLE against the following benchmarks: R-PEBBLE, Deep TAMER [Warnell *et al.*, 2018] (a scalar feedback HitL RL algorithm), and SAC. For the preference-based experiments, we further show the robustness of SDP by combining it with three SOTA preference-based algorithms. This results in the following benchmarks: PEBBLE [Lee *et al.*, 2021], RUNE [Liang *et al.*, 2022], SURF [Park *et al.*, 2022], and SAC. We treat SAC [Haarnoja *et al.*, 2018] as an oracle baseline because it learns while accessing the true reward function, which is unavailable to the other algorithms (SAC is the core RL algorithm used across all baselines). We include it as a baseline only to show an upper bound of task performance. See Appendix A for full implementation and hyperparameter details.

For evaluation, we show average offline performance (i.e., freeze the policy and evaluate it with no exploration) over five episodes using either the ground truth reward function (DMControl experiments) or the success rate (Meta-world experiments). We perform this evaluation every 10,000 training steps. To systemically evaluate performance, we use a scripted teacher that provides either a scalar rating of a single trajectory segment or preferences between two trajectory segments according to the true reward function. To thoroughly test the effectiveness of SDP, we perform evaluations on three robotic locomotion tasks from the DMControl Suite: Walker-walk, Cheetah-run, and Quadruped-walk, and six robotic manipulation tasks from Meta-world: Hammer, Door-unlock, Door-lock, Drawer-open, Door-open, and Window-open.

In our experiments, the results are averaged over five seeds with any shaded regions or error bars indicating 95% confidence intervals. To test for significant differences in final performance and learning efficiency (e.g., area under the curve, AUC), we perform Welch t-tests (equal variances not assumed) with a p-value of 0.05. See Appendix B, Tables 4, and 5 for a summary of final performance and AUC across all tasks.

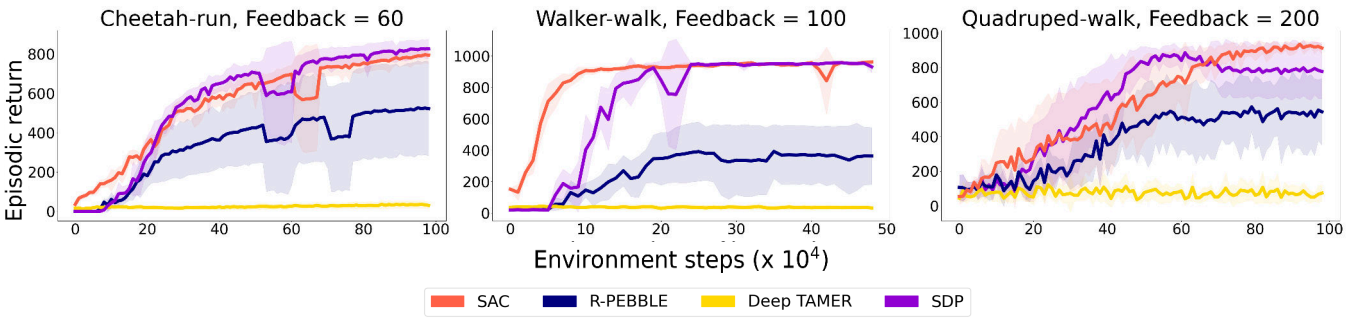


Figure 2: Scalar feedback experiments. In all DMControl experiments, SDP significantly outperforms R-PEBBLE and Deep TAMER ($p < 0.05$) and achieves comparable performance to the SAC oracle.

5.2 Locomotion and Manipulation Results

Scalar Feedback Experiments We first evaluate the utility of SDP in the scalar-based HitL RL setting. Across all tested DMControl environments, we found that SDP (purple curve) significantly improves either the final performance or the learning efficiency (e.g., AUC) compared to R-PEBBLE (navy curve) and Deep TAMER (yellow curve) (see Figure 2). More impressively, we found that SDP achieves comparable performance to SAC (red curve), which uses the ground truth reward function, using as little as 60 feedback queries.

Preference Feedback Experiments To further showcase the effectiveness of SDP, we perform preference learning experiments in both DMControl and Metaworld suites. Considering all three preference-based algorithms in the nine environments, SDP significantly ($p < 0.05$) improved learning (i.e., either final performance or AUC) in 15 out of the 27 experiments (see Figure 3 and Table 4 in Appendix B). In the remaining experiments, there were no significant differences in the performance between SDP and the baseline algorithms. The addition of SDP (i.e., SDP + base algorithm) *never* hurt performance. Furthermore, we performed additional experiments using an imperfect teacher and the results are consistent (see Figure 11 in Appendix B).

Experiments with Sub-Optimal data from Different Tasks Our earlier experiments demonstrated that if we have sub-optimal data for our task of interest (i.e., target task), then SDP can leverage this data to improve reward learning HitL RL methods. However, in many cases, we may also have sub-optimal data from other related tasks. This section considers whether SDP can improve performance on a target task if it leverages sub-optimal data from a different task.

We perform three preference learning experiments, comparing PEBBLE with SDP + PEBBLE, using sub-optimal data from a different prior task that has the same virtual robot (i.e., only reward function differs): (1) Walker-stand for the target task of Walker-walk, (2) Quadruped-walk for the target task of Quadruped-run, and (3) Drawer-open for the target task of Door-open. To obtain the sub-optimal data for the prior tasks, we gathered transitions from partially trained policies as opposed to using random policies. This ensured that the distribution of sub-optimal data differed between the prior and target tasks. See Appendix A.2 for further details on the experiment setup. Figure 4a demonstrates the flexibility of SDP as it can successfully leverage sub-optimal data from related tasks (green curve) and still outperform PEBBLE (blue curve) in two out of three of the environments.

5.3 Ablation Studies

To further understand the effectiveness of SDP, we perform further analysis of SDP across three dimensions: (1) the phases of SDP, (2) the amount of sub-optimal data, and (3) the number of feedback queries. To cover scalar- and preference-based HitL RL settings in both DMControl and Meta-world, we consider SDP + R-PEBBLE (i.e., scalar feedback) in Walker-walk and SDP + PEBBLE (i.e., preference feedback) in Cheetah-run and Door-open.

SDP Component Analysis To understand the effect of each component of SDP, we perform three ablations. First, we evaluate the effect of each phase of SDP individually (i.e., the reward model pre-train phase and the agent update phase). Figure 4b–leftmost demonstrates the importance of using both phases in SDP for scalar-based HitL RL approaches. We found that the SDP variants that only use one of the phases (green and gray curves) result in worse performance than the full SDP (purple curve). We observed similar results in the Cheetah-run and Door-open preference learning experiments (see Figures 8a and 8b in Appendix B).

Second, in the reward model pre-training phase, the goal is for the reward model to learn to output zero. However, a trivial means to achieve an output of zero is to set all weights and biases in the neural network to zero. Therefore, we compare the full SDP to SDP using a zero-weight initialization as a replacement for the reward model pre-training. We found that only using a zero-weight initialization for the reward model instead of the pre-training phase results in significantly degraded performance (see the green curve in Figure 4b–rightmost). This is not surprising, as previous works have found that a zero-weight initialization can negatively affect the training of neural networks [Blumenfeld *et al.*, 2020; Zhao *et al.*, 2022]. Figure 7 in Appendix B further shows that the reward model pre-training phase does not result in zero-weight values for the reward model. This explains why our

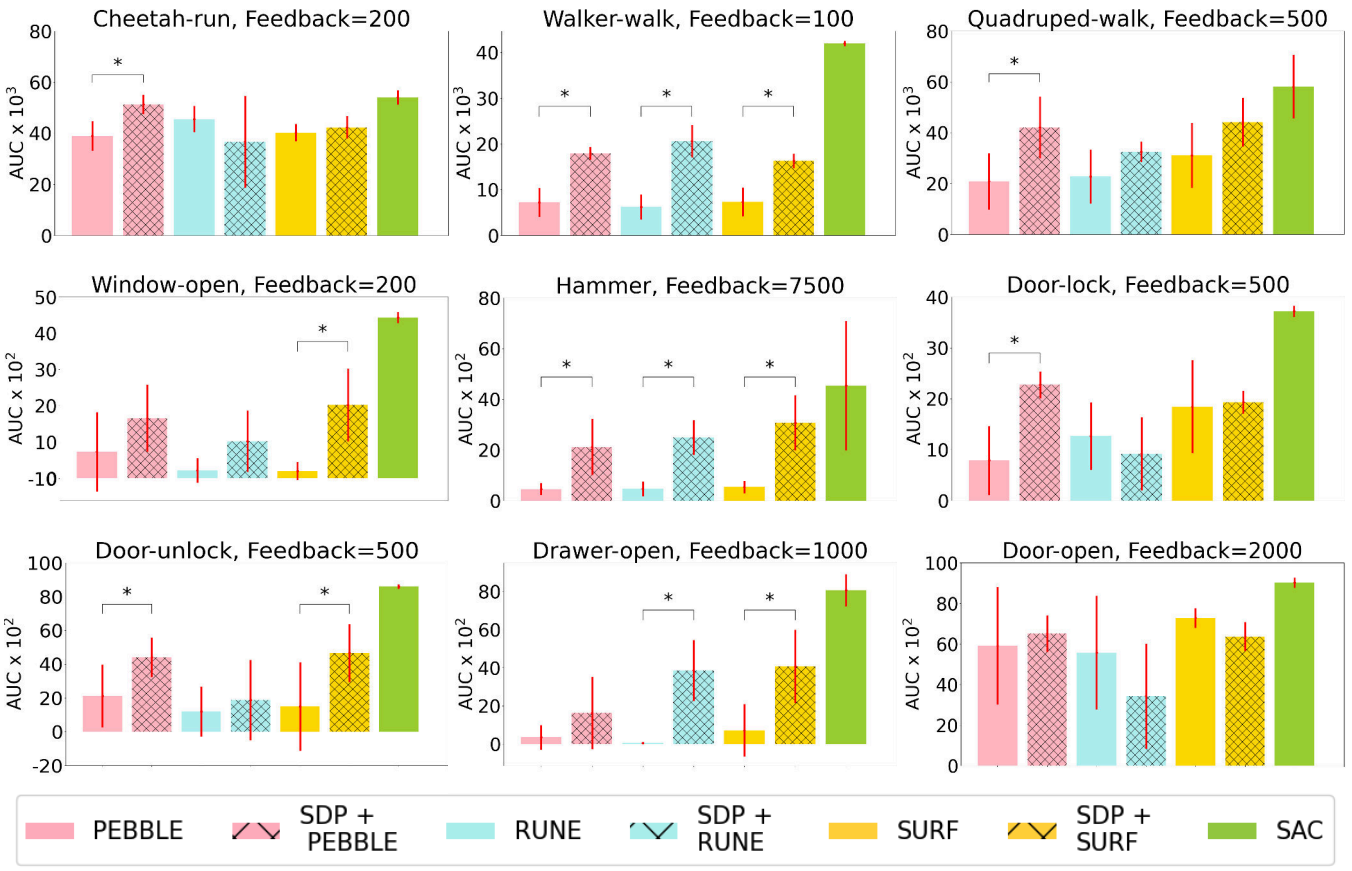


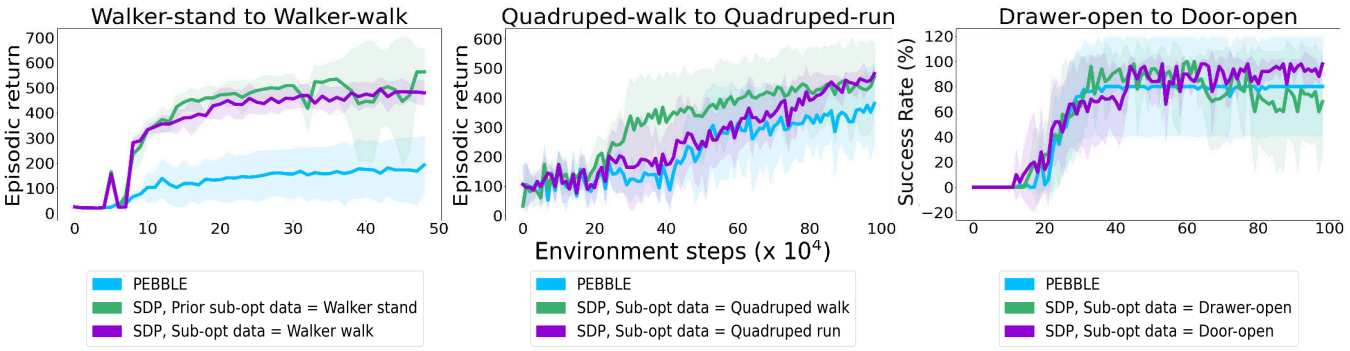
Figure 3: Preference feedback experiments in DMControl and Metaworld suites. The bar plots show AUC +/- 95% confidence intervals. * indicates that SDP + the base preference learning algorithm achieves a statistically greater score ($p < 0.05$) than the base preference learning algorithm alone (i.e., without SDP).

pre-training does not lead to the same poor performance as the zero-weight initialization.

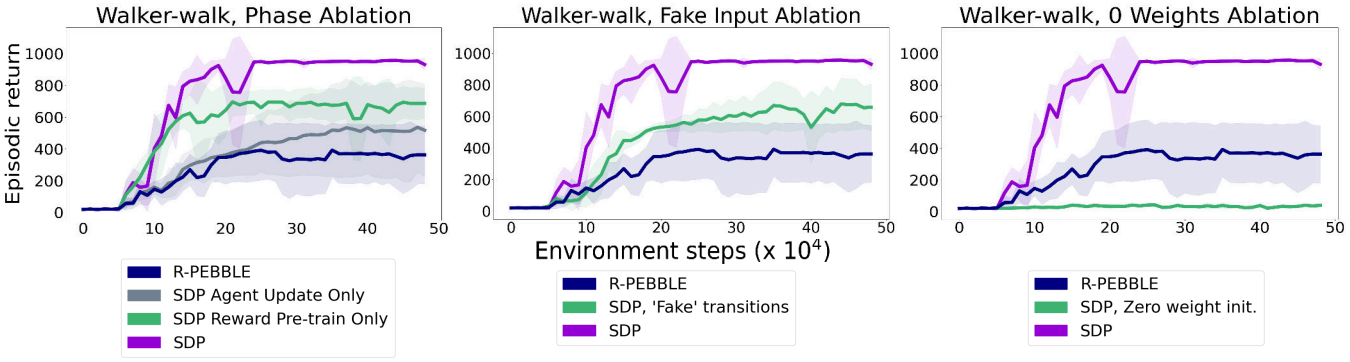
Third, SDP makes use of state, action transitions that are gathered through a sub-optimal policy. Therefore, these are transitions that an agent experiences while interacting with its environment. However, as previously noted, the goal of the reward model pre-training phase is for the reward model to learn to output zero. Therefore, is it necessary for the reward model to pre-train on inputs that are real environment transitions? Instead, can we pre-train the reward model on transitions that did not result from an agent-environment interaction? To test this, we created "fake" inputs of size dim(state) + dim(action), and for each input dimension, we randomly sampled a value from $\mathcal{N}(0, 1)$. We obtained 50,000 "fake" transitions and used this data for the reward model pre-training phase. In this experiment, our goal is to understand the effect of the type of inputs on the reward model pre-training phase, therefore we kept the agent update phase as is (i.e., provided the true sub-optimal data for this phase). We then compared SDP using true transitions to SDP using "fake" transitions in Figures 4b–middle and Figure 8c in Appendix B. We found that the full SDP (purple curve) in which we pre-train the reward model on true sub-optimal transitions yields significantly greater final performance than SDP using "fake" transitions (green curve). This highlights the importance of using true sub-optimal transitions in SDP.

Effect of Sub-Optimal Data Amount To analyze the effect of the amount of sub-optimal data on the performance of SDP, we evaluated SDP using three different sub-optimal data budgets: $\{5000, 15000, 50000\}$. In Figure 4c–leftmost, we found that for the Walker-walk scalar feedback experiment, increasing the amount of sub-optimal data in SDP leads to better final performance. We observed a similar pattern in the Cheetah-run and Door-open preference learning experiments (see Figures 9a and 9b in Appendix B).

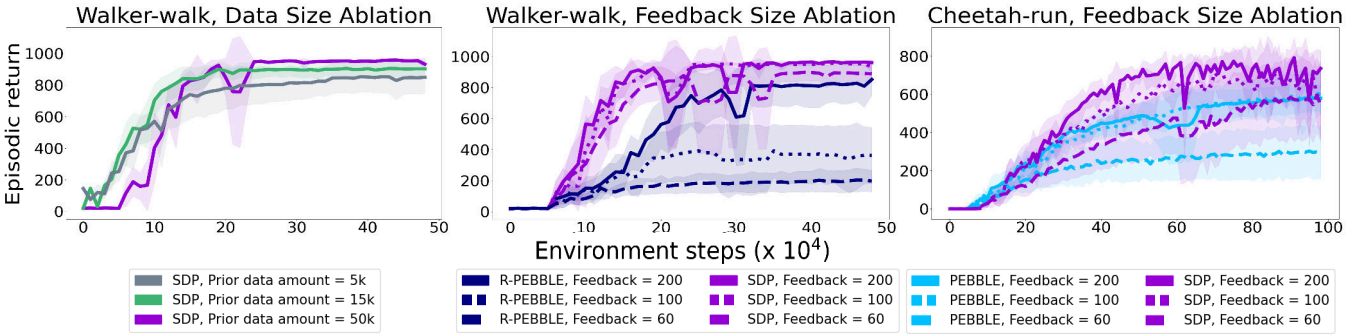
Effect of Feedback Amount To investigate the effect of the amount of feedback on the performance of SDP, we evaluated SDP and PEBBLE (and their scalar feedback variants) using three different feedback budgets. For both Cheetah-run and Walker-walk environments, we considered budgets $N \in \{60, 100, 200\}$. In Figure 4c (middle and right-most), we show that SDP (purple curves) consistently outperforms PEBBLE (blue curves) and R-PEBBLE (navy curves). This further supports SDP's effectiveness across a range of feedback queries. We achieved similar results in the Door-open environment (see Figure 9c in Appendix B).



(a) These figures highlight that SDP can leverage sub-optimal data from tasks other than the target task and still outperform PEBBLE.



(b) These figures demonstrate that the complete SDP outlined in Algorithm 1 achieves significantly greater performance than various SDP variants.



(c) These figures highlight that SDP is effective with differing feedback budgets but is more successful with larger amounts of prior data.

Figure 4: This figure showcases several SDP ablations: SDP transfer ablations (Top), SDP component ablations (Middle), and amount of prior pre-training data and feedback ablations (Bottom).

6 Conclusion

In this work, we present SDP, an approach that improves the feedback efficiency for HitL RL algorithms. SDP is specifically designed to leverage reward-free, sub-optimal data for scalar- and preference-based HitL RL approaches. By pseudo-labeling low-quality data with zero, we secure "free" reward labels to pre-train the reward model, giving the reward model a head start in learning. Through this process, the pre-trained reward model can identify that low-quality transitions should have a low reward value, all without having acquired any actual feedback. Our extensive experiments in both DM-Control and Metaworld suites demonstrate that SDP can significantly improve both preference- and scalar-based reward learning algorithms. This work takes an important step towards considering how sub-optimal data can be leveraged for HitL RL. Future work should consider other mechanisms for leveraging low-quality data. One interesting direction could be using sub-optimal data from multiple tasks to improve HitL RL algorithms. Another possibility is to consider approaches that can leverage *both* sub-optimal and expert data.

References

[Abbeel and Ng, 2004] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Twenty-First International Conference on Machine Learning*, 2004.

[Andrychowicz *et al.*, 2017] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Thirty-First Conference on Neural Information Processing Systems*, 2017.

[Argall *et al.*, 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

[Arora and Doshi, 2021] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 2021.

[Bıyık *et al.*, 2022] Erdem Bıyık, Dylan P Losey, Malayandi Palan, Nicholas C Landolfi, Gleb Shevchuk, and Dorsa Sadigh. Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences. *The International Journal of Robotics Research*, 2022.

[Blumenfeld *et al.*, 2020] Yaniv Blumenfeld, Dar Gilboa, and Daniel Soudry. Beyond signal propagation: Is feature diversity necessary in deep neural network initialization? In *Thirty-Seventh International Conference on Machine Learning*, 2020.

[Booth *et al.*, 2023] Serena Booth, W. Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 2023.

[Bradley and Terry, 1952] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 1952.

[Brown *et al.*, 2019] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *Thirty-sixth International Conference on Machine Learning*, 2019.

[Brown *et al.*, 2020] Daniel S. Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Third Proceedings of the Conference on Robot Learning*, 2020.

[Brys *et al.*, 2015] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[Cabi *et al.*, 2020] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *Robotics: Science and Systems*, 2020.

[Chen *et al.*, 2021] Letian Chen, Rohan Paleja, and Matthew Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. In *Forth Conference on Robot learning*, 2021.

[Christiano *et al.*, 2017] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Thirty-First Conference on Neural Information Processing Systems*, 2017.

[Clark and Amodei, 2016] Jack Clark and Dario Amodei. Faulty reward functions in the wild, 2016.

[Cui and Niekum, 2018] Yuchen Cui and Scott Niekum. Active reward learning from critiques. In *International Conference on Robotics and Automation*, 2018.

[Daniel *et al.*, 2014] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Robotics: Science and Systems*, 2014.

[Dragan and Srinivasa, 2012] Anca Dragan and Siddhartha Srinivasa. Formalizing assistive teleoperation. *Robotics: Science and Systems*, 2012.

[Gao *et al.*, 2019] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. In *Thirty-Fith International Conference on Machine Learning*, 2019.

[Griffith *et al.*, 2013] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Twenty-Sixth Conference on Neural Information Processing Systems*, 2013.

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Thirty-Fifth International Conference on Machine Learning*, 2018.

[Hejna III and Sadigh, 2023] Donald Joseph Hejna III and Dorsa Sadigh. Few-shot preference learning for human-in-the-loop rl. In *Sixth Conference on Robot Learning*, 2023.

[Hester *et al.*, 2018] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[Ibarz *et al.*, 2018] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. In *Thirty-Second Conference on Neural Information Processing Systems*, 2018.

[Knox and Stone, 2009] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement. In

Fifth International Conference on Knowledge Capture, 2009.

[Knox and Stone, 2013] W Bradley Knox and Peter Stone. Learning non-myopically from human-generated reward. In *International Conference on Intelligent User Interfaces*, 2013.

[Lee *et al.*, 2021] Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. In *Thiry-Eighth International Conference on Machine Learning*, 2021.

[Liang *et al.*, 2022] Xinran Liang, Katherine Shu, Kimin Lee, and Pieter Abbeel. Reward uncertainty for exploration in preference-based reinforcement learning. In *Tenth International Conference on Learning Representations*, 2022.

[Loftin *et al.*, 2016] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous Agents and Multi-Agent Systems*, 2016.

[MacGlashan *et al.*, 2017] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, David Roberts, Matthew E Taylor, and Michael L Littman. Interactive learning from policy-dependent human feedback. In *Thirty-Forth International Conference on Machine Learning*, 2017.

[Muslimani *et al.*, 2022] Calarina Muslimani, Alex Lewandowski, Dale Schuurmans, Matthew E Taylor, and Jun Luo. Reinforcement teaching. *Transactions on Machine Learning Research*, 2022.

[Park *et al.*, 2022] Jongjin Park, Younggyo Seo, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. Surf: Semi-supervised reward learning with data augmentation for feedback-efficient preference-based reinforcement learning. In *Tenth International Conference on Learning Representations*, 2022.

[Settles, 2009] Burr Settles. Active learning literature survey. 2009.

[Shiarlis *et al.*, 2016] Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. Inverse reinforcement learning from failure. In *Fifteenth International Conference on Autonomous Agents and Multiagent Systems*, 2016.

[Shorten and Khoshgoftaar, 2019] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 2019.

[Singh *et al.*, 2020] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *Conference on Robot Learning*, 2020.

[Skalse *et al.*, 2022] Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[Tangkaratt *et al.*, 2021] Voot Tangkaratt, Nontawat Charoenphakdee, and Masashi Sugiyama. Robust imitation learning from noisy demonstrations. In *Twenty-Forth International Conference on Artificial Intelligence and Statistics*, 2021.

[Tassa *et al.*, 2018] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[Taylor *et al.*, 2011] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *Tenth International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[Vien *et al.*, 2013] Ngo Anh Vien, Wolfgang Ertel, and Tae Choong Chung. Learning via human feedback in continuous state and action spaces. *Applied intelligence*, 2013.

[Warnell *et al.*, 2018] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[White *et al.*, 2023] Devin White, Mingkang Wu, Ellen Novoseller, Vernon Lawhern, Nick Waytowich, and Yongcan Cao. Rating-based reinforcement learning. *arXiv preprint arXiv:2307.16348*, 2023.

[Yu *et al.*, 2020] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Third Conference on Robot Learning*, 2020.

[Yu *et al.*, 2021] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

[Yu *et al.*, 2022] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Chelsea Finn, and Sergey Levine. How to leverage unlabeled data in offline reinforcement learning. In *Thiry-Ninth International Conference on Machine Learning*, 2022.

[Zhao *et al.*, 2022] Jiawei Zhao, Florian Tobias Schaefer, and Anima Anandkumar. Zero initialization: Initializing neural networks with only zeros and ones. *Transactions on Machine Learning Research*, 2022.

[Zhu, 2005] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.

Appendix

A Experimental Details

A.1 Benchmarks

PEBBLE PEBBLE has two primary components: unsupervised exploration and off-policy learning with relabeling. The purpose of the unsupervised exploration phase is to collect diverse experiences for a human teacher to provide feedback on. More specifically, PEBBLE optimizes state entropy to explore the environment. Furthermore, PEBBLE uses off-policy reinforcement learning to learn a policy. PEBBLE specifically uses an off-policy RL algorithm as they are more sample efficient compared to their on-policy counterparts. Then as the reward model changes, PEBBLE relabels all transitions in the RL agent's replay buffer with the latest reward model. This is integral as the reward model is non-stationary, and relabeling the transitions stabilizes the learning process.

To adapt PEBBLE to the scalar feedback setting, we make one minor change to the reward model. In the scalar feedback setting, we use a scripted teacher that provides a scalar rating of a single trajectory segment. Therefore, the only update to PEBBLE is with respect to the loss function. Instead of using the cross-entropy loss in Equation 2, we use the mean-squared error loss in Equation 3.

RUNE RUNE is a preference learning algorithm (built on top of PEBBLE) that uses an uncertainty-based exploration strategy to improve feedback efficiency. To encourage exploration for the SAC agent, RUNE adds an intrinsic reward component based on the standard deviation in the reward model.

SURF SURF is another preference learning algorithm (built on top of PEBBLE) that improves feedback efficiency by using semi-supervised and data augmentation approaches. To incorporate semi-supervised learning, SURF generates pseudo-labels for unlabeled trajectories by querying the learned reward model. If the reward model confidently (e.g., low output standard deviation) predicts the pseudo-label, then the trajectory, label pair is added to the reward model training data set. Further, SURF proposes a new data augmentation technique that crops sub-sequences of trajectories.

Deep TAMER In our scalar feedback experiments, we also consider the Deep TAMER benchmark. In Deep TAMER, scalar feedback is used to learn a human reward function via regression. Then the agent acts greedily with respect to this reward function. Furthermore, the original implementation of Deep TAMER was built on top of DQN. Therefore, there was no separate actor-network. In addition, Deep TAMER only used discrete feedback $\in [-1, 0, 1]$.

To make Deep TAMER a fair benchmark, we made a few adjustments. To start, we allow Deep TAMER to learn from real-valued feedback as done in the other scalar-based experiments. However, instead of using the ground truth reward function as feedback, we use the state-action values from a fully-trained SAC agent. We do this because, in TAMER (and Deep TAMER), the teacher is intended to provide feedback representative of the return. Secondly, in Deep TAMER, feedback is provided per (state, action) pair. Therefore, to make sure Deep TAMER received the same amount of feedback as the other baselines, we used *trajectory segment size* $\times$ *feedback amount* for Deep TAMER only. The other benchmarks receive a scalar feedback value that is the sum of rewards along a trajectory segment. Third, to learn the reward model we use standard regression as described in Section 3.1. Lastly, as our testing environments are continuous state and action, we learn a separate actor policy, similarly done in [Vien *et al.*, 2013].

A.2 Training details

In all of our experiments, we use the hyperparameters in Table 3 for the reward models used in all benchmarks. For the agent update phase of SDP, an additional hyperparameter is associated with the number of environment interactions made before the standard preference/scalar feedback learning loop begins. However, for simplicity, we kept the same value as the existing feedback frequency hyperparameter, as feedback frequency also dictates the number of environment interactions made between feedback sessions

Furthermore, we use most of the existing reward model hyperparameters used in PEBBLE, however, we adjusted the following four hyperparameters: feedback frequency, amount of feedback per session, trajectory segment size (only for Meta-world), and activation function for the final NN layer. We adjusted the first two hyperparameters because PEBBLE originally used a significantly larger feedback budget, therefore we wanted the feedback schedule to better reflect a smaller feedback budget. We used a different trajectory segment size for Meta-world because we wanted to keep the segment sizes the same across both the DMControl and Meta-world environments. Moreover, we found that the output activation function could significantly affect learning, therefore we tested all benchmarks using both Tanh (original activation used) and Leaky-ReLU and chose the reward model that achieved better final performance. For the RUNE and SURF baselines, we use any hyperparameters associated with their specific algorithm according to the original paper (see Table 2). For a fair comparison with SDP, we provide all HiTL baselines (e.g., PEBBLE, R-PEBBLE, Deep TAMER, RUNE, and SURF) with the sub-optimal data set to be used in both the reward model and by the RL agent.

Furthermore, to select trajectory segments for the teacher to provide feedback on, we use uniform sampling in the DMControl tasks and disagreement sampling in the Meta-world tasks. Disagreement sampling is a popular active learning approach in which trajectories with higher uncertainty (based on an ensemble of neural networks) are more likely to be sampled [Christiano *et al.*, 2017]. As for the SAC hyperparameters, we use the values found in Table 1.

For the experiments in which we leveraged sub-optimal data from a different task (i.e., Walker-stand, Quadruped-walk, Drawer-open), we gathered 50,000 transitions from partially trained policies. We note that for these experiments, we purposely did not use transitions gathered from a random policy. In these experiments, the prior and target tasks were environments in which the simulated robot was identical. The only difference is the environmental reward. Therefore, the

random policy for both environments would be the same. To truly demonstrate transfer, we wanted to ensure we obtained low-quality transitions of the prior task that were different from the target task.

Each partially trained policy achieved a final score of approximately 15-20% of that achieved by a fully trained policy. More specifically, we used the following procedure to train the SAC policies. First, for Walker-stand, we trained a SAC policy for 5,000 time steps, and the average final performance was approximately 194. Second, for Quadruped-walk, we trained a SAC policy for 100,000 timesteps, and the average final performance was approximately 184. Lastly, for Drawer-open, we trained a SAC policy for 50,000 time steps, and the average final success rate was approximately 14%.

Hyperparameter	Value
Initial temperature	.1
Discount	.99
Batch size	1024 (DMControl) 512 (Metaworld)
Critic, Alpha, Actor: β_1, β_2	$(.9, .999)$
Critic τ	0.005
Optimizer	Adam
Critic target update frequency	2
Learnable temperature	True
Actor and Critic: # of hidden layers	2 (DMControl) 3 (Meta-world)
Actor and Critic: # of hidden units per layer	1024 (DM Control) 256 (Meta-world)
Actor and Critic learning rate	0.00005 (Cheetah-run pref fb exp) 0.0001 (Cheetah-run scalar fb exp) 0.0005 (Walker-walk) 0.0001 (Quadruped) 0.0003 (Meta-world)
Alpha learning rate	0.0001
Actor update frequency	1
Number of training steps	1 million for all experiments except Walker-walk (Pref and scalar feedback exps.), Door-lock Window-open

Table 1: Hyperparameters for SAC agent used in all experiments (both preference and scalar feedback variants).

Hyperparameter	Value
SURF: Threshold μ	.99
SURF: Lambda μ	1
SURF: μ	4
SURF: Inverse label ratio	10
SURF: Data augmentation window	5
SURF: Crop Range	5
RUNE: Beta schedule	Linear decay
RUNE: Beta init	0.05
RUNE: Beta decay	0.00001

Table 2: Hyperparameters for SURF and RUNE. All are taken from the original authors' implementations.

Hyperparameter	Value
Segment size	50
# of random pre-training steps (i.e., sub-optimal data transitions)	50000
# of unsupervised exploration steps	9000
Include unsup steps as sub-opt data for SDP	All experiments except Walker-walk (Pref and scalar feedback exps.), Door-open and Quadruped-walk (Pref learning exps.)
Learning rate	0.0003
Batch size	128
Ensemble size	3
Frequency of feedback	20000 (DMControl) 10000 (Window-open, Door-unlock) 5000 (Hammer, Drawer-open, Door-open, Door-lock) 2000 (Door-open)
Feedback budget for ablations	100 (Cheetah-run, Walker-walk) 50 (Hammer, Drawer-open, Door-open, Door-lock)
# of feedback queries per session	20 (All DMControl tasks, Window-open, Door-unlock) 10 (Cheetah-run scalar fb experiment)
Sampling scheme	Uniform (DMControl) Disagreement Sampling (Meta-world)
# of training updates	50
# of NN layers	4 (including output layer)
Intermediate NN activations	Leaky relu
Final output activation	Leaky ReLU (PEBBLE and Deep TAMER) Leaky ReLU (SDP all except Tanh for Hammer, Drawer-open, Walker-walk (preference feedback exp).)
Hidden units	128
Loss	Cross entropy (Preference learning experiments) MSE (Scalar feedback experiments)
Optimizer	Adam

Table 3: Hyperparameters for the reward model used in all experiments (both preference and scalar feedback variants).

B Additional analysis

For simplicity, in all additional experiments in this section, we only compare SDP + PEBBLE with PEBBLE (or R-PEBBLE).

Figure 5 emphasizes how the agent update phase does result in new transitions, therefore the teacher provides feedback to transitions that are different from the original sub-optimal transitions used for pre-training. Figure 6 shows the true reward values for sub-optimal data gathered through a random policy in the DMControl suite. This emphasizes that the true reward value is close to the value we pseudo-label the sub-optimal transitions with (i.e., zero), therefore SDP should not yield a large incorrect reward bias.

Figure 7 demonstrates that the reward model pre-training phase does not produce reward model weights of zero, emphasizing why we do not experience the same performance degradation that occurs if we use a zero-weight initialization.

Figures 8a and 8b show additional phase ablations in the preference learning experiments done on the Cheetah-run and Door-open tasks. This highlights the importance of using both phases in SDP. Furthermore, Figure 8c shows another example of SDP when pre-trained with fake transitions in Cheetah-run. This emphasizes that SDP achieves its best performance when pre-trained with real sub-optimal transitions an RL agent experiences.

Figures 9a and 9b show additional ablations over the amount of prior data used in SDP and Figure 9c ablates the number of feedback queries. Overall, we observed similar performance gains as described in section 5.3.

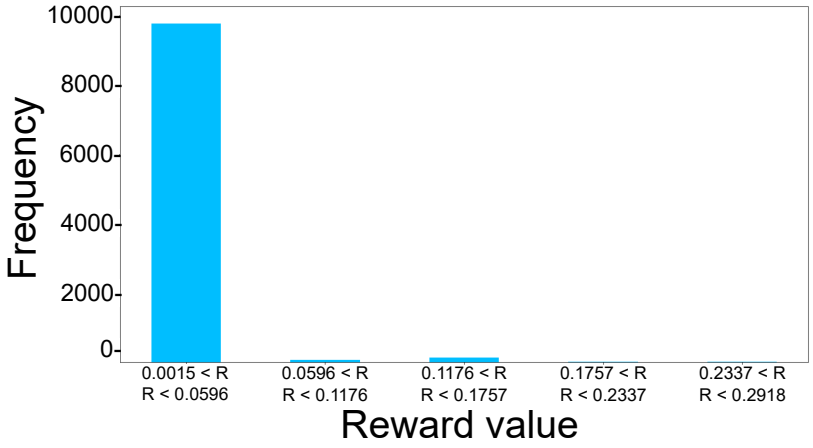
SDP is combined with three preference learning algorithms, PEBBLE, RUNE, and SURF. A core feature of these algorithms is that every time the reward model is updated, all transitions inside the RL agent's replay buffer are updated using the latest reward model. Figure 10a shows the effects of not relabeling the sub-optimal data (i.e., the reward labels) with the latest learned reward model. This means the reward labels remain frozen at zero throughout the entire training process. This ablation was to demonstrate that if the sub-optimal data in the agent's replay buffer is not updated with the latest reward model, then performance will suffer. This is likely the case because the incorrect reward bias from the pseudo-labeling process persists, whereas when we relabel the transitions with the updated reward model, the incorrect reward bias may reduce over time.

Moreover, we show that the effectiveness of SDP relies on the use of sub-optimal data transitions. If we use high-quality data transitions (i.e., transitions that came from a fully trained RL agent policy), SDP will fail (see Figure 10b). This unsurprising result confirms that pseudo-labeling high-reward transitions with zero can significantly hurt the reward model and the agent's performance

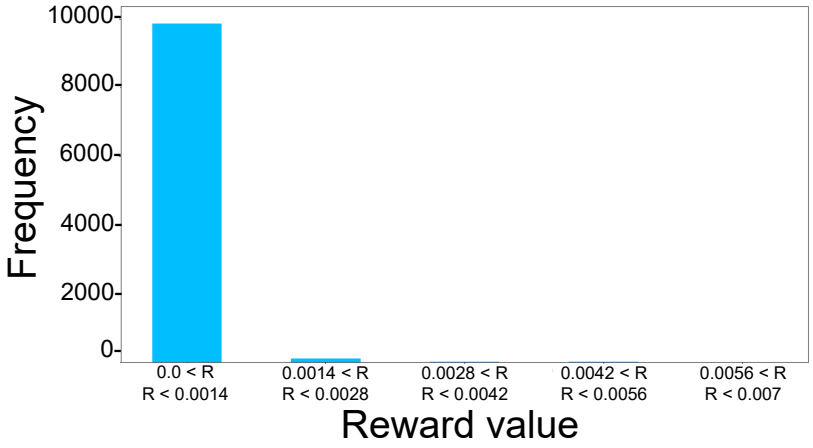
In our earlier experiments, we assumed our teacher always provided accurate feedback. However, human teachers can make mistakes when giving feedback. Therefore, we performed experiments with noisy teachers to more closely resemble human teachers. For the scalar-feedback experiment, to create the noisy teacher, we first add a noise value $X \sim \mathcal{N}(0,\, 1)$ to the numerical ratings, and then round them to the closest integer. In the preference feedback experiments, we change the preference ordering with 10% probability. Unsurprisingly, the presence of noisy teachers causes degraded performance for both SDP and PEBBLE (or R-PEBBLE) (see dotted curves in Figure 11). However, our results demonstrate that SDP can maintain performance gains over PEBBLE and R-PEBBLE despite the presence of a noisy teacher.

Tables 4 and 5 provide a summary of the mean final performance and the mean area under the curve for all environments and benchmarks.

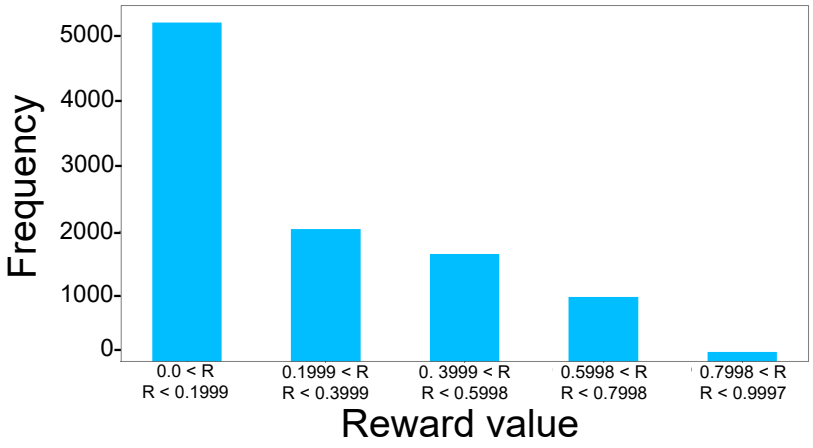
Figure 5: Door open, preference learning exp. These plots show how the agent's policy has changed from the reward model pre-training phase (purple histograms) to the agent update phase (blue histograms), thereby resulting in a different distribution for the state features.



(a) Walker-walk.

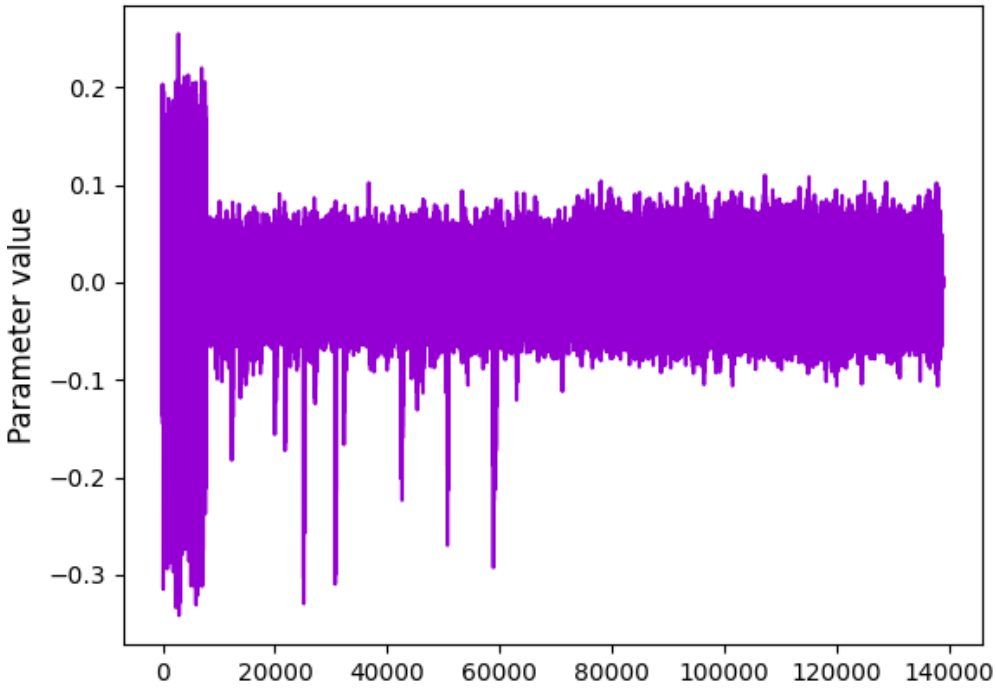


(b) Cheetah-run.

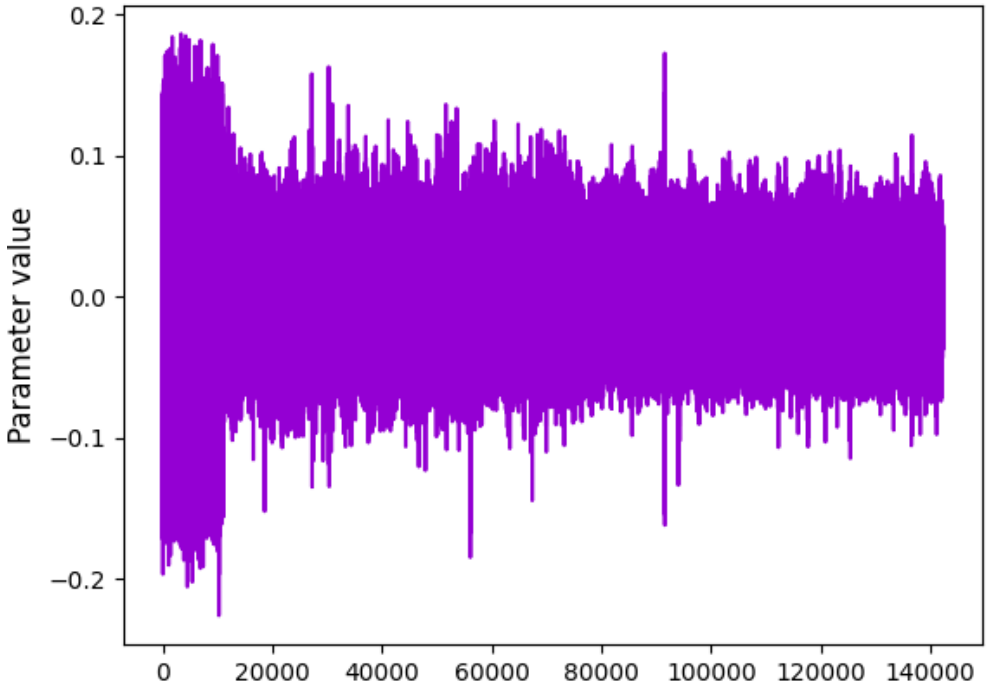


(c) Quadruped-walk.

Figure 6: Distribution of true reward values for transitions obtained with a random policy

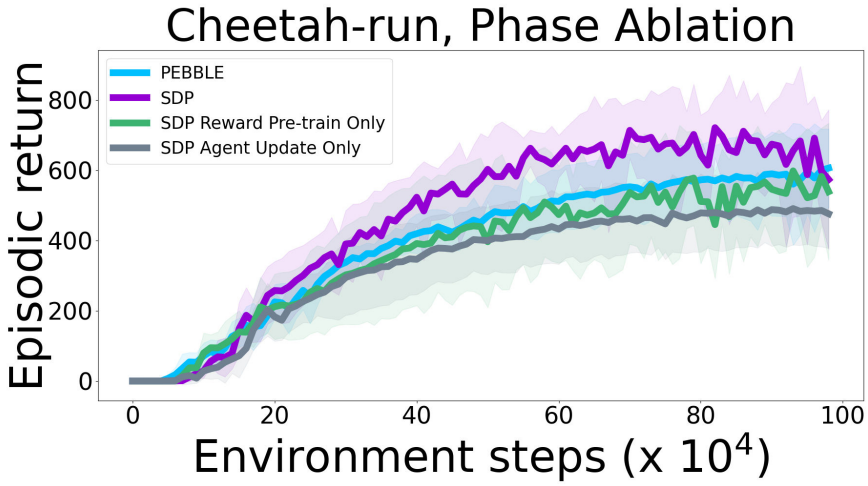


(a) Scalar feedback experiment in Walker-walk.

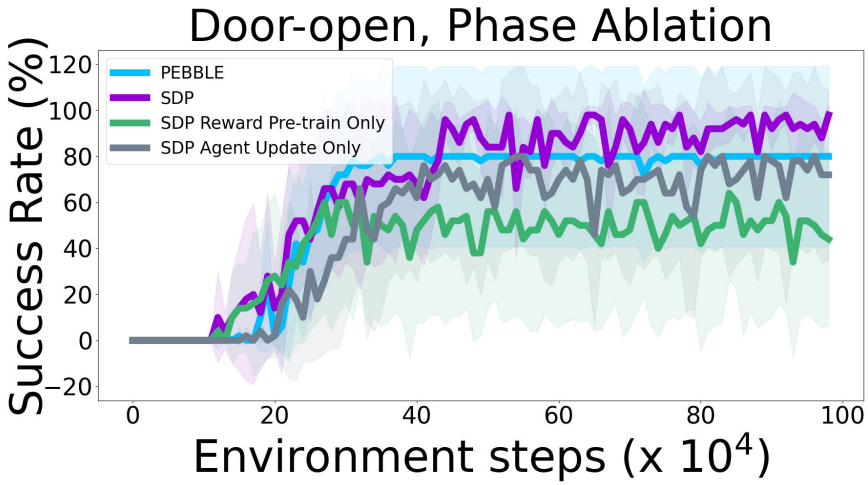


(b) Preference feedback experiment in Door-open

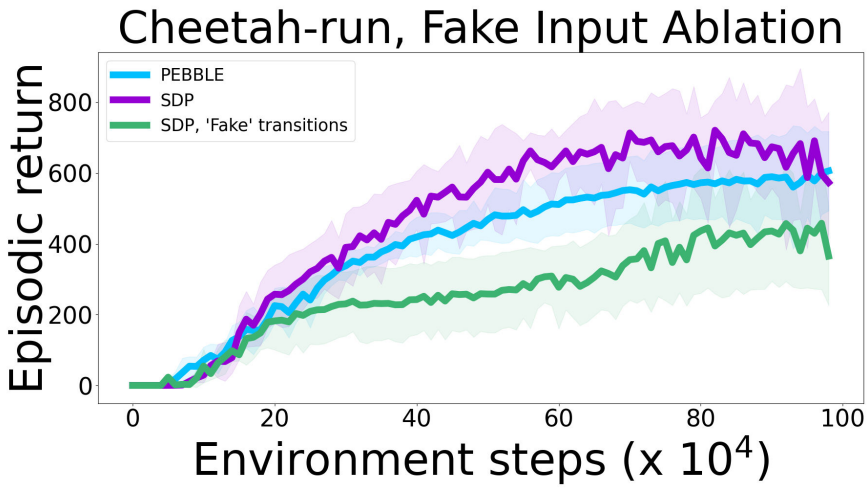
Figure 7: These figures show the reward model weights of SDP after the reward model pre-training phase. This demonstrates that the reward model pre-train phase of SDP does not result in zero neural network weights.



(a) SDP phase ablation: Cheetah-run, preference feedback

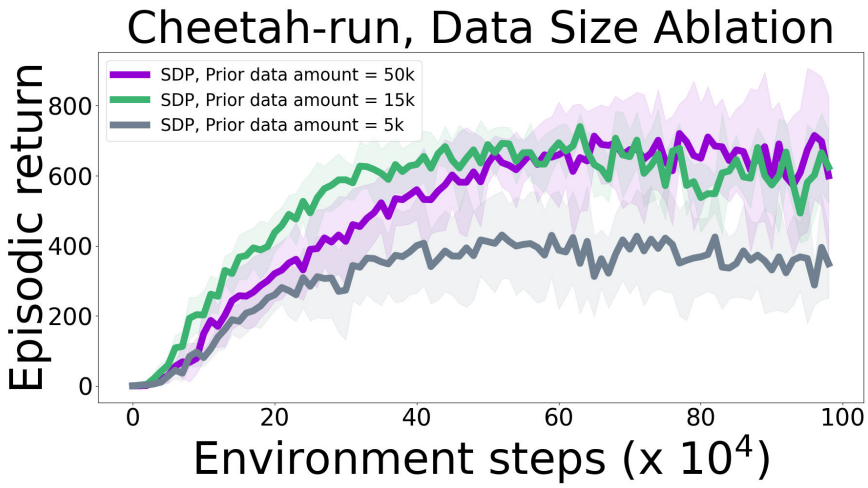


(b) SDP phase ablation: Door-open, preference feedback

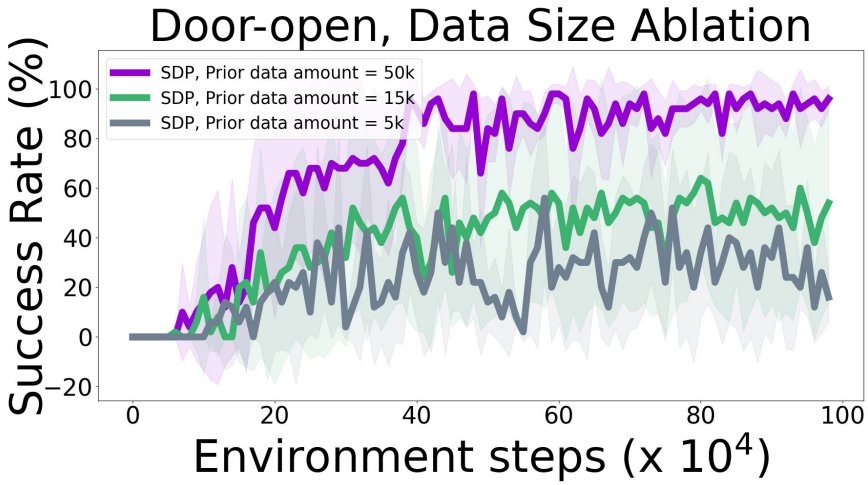


(c) Fake input ablation: Cheetah-run, preference feedback

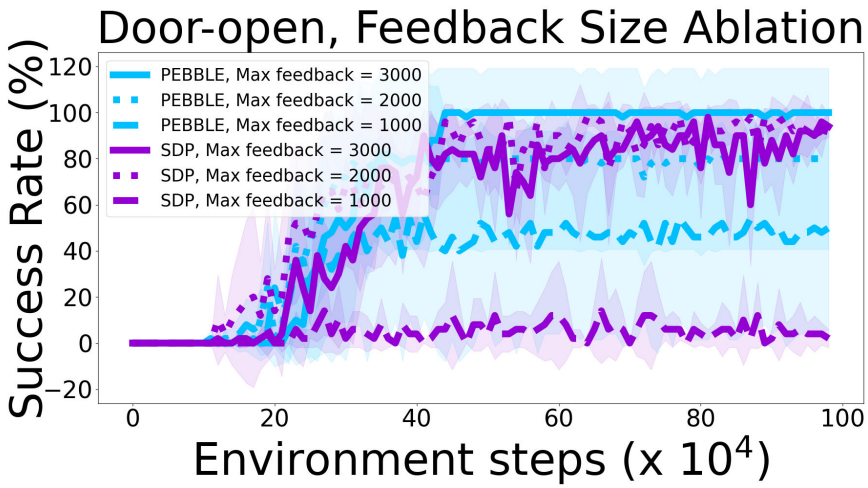
Figure 8: SDP phase and fake input studies: Figures 8a and 8b demonstrate the importance of using both components of SDP, reward pre-training (green) and agent update (grey). Figure 8c compares SDP pre-trained with real sub-optimal transitions to SDP pre-trained with fake transitions.



(a) Amount of prior data study: Cheetah-run, preference feedback

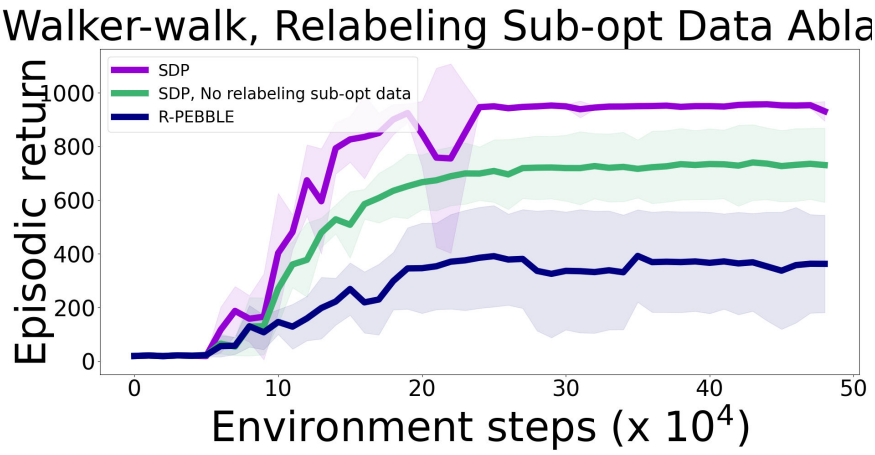


(b) Amount of prior data study: Door-open, preference feedback

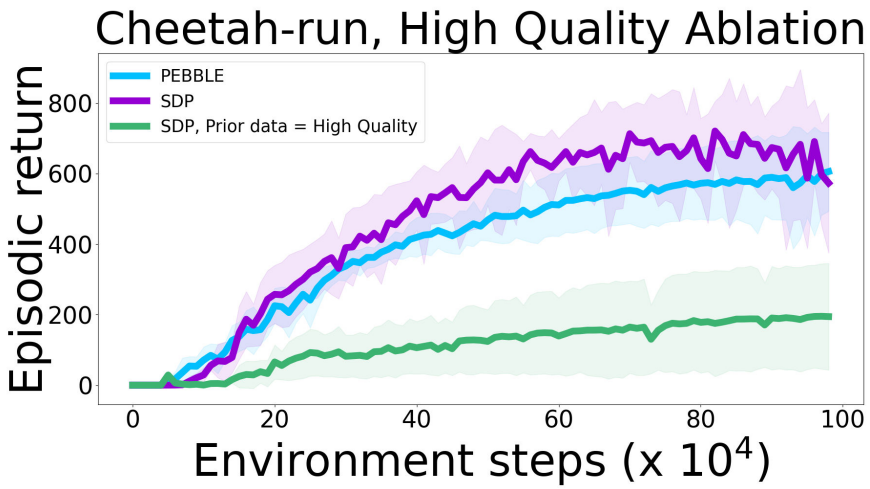


(c) Amount of feedback study: Door-open, preference feedback

Figure 9: Amount of pre-training data and feedback studies: Figures 9a and 9b compare the number of sub-optimal data used in SDP. Figure 9c compares the number of feedback queries and their effect on SDP and PEBBLE.



(a) Relabeling sub-opt data study: Walker-walk, scalar feedback



(b) Using high-quality data in SDP study: Cheetah-run, preference feedback

Figure 10: Figure 10a shows the results of not relabeling sub-optimal data transitions with the updated reward model. Figure 10b shows the results of pseudo-labeling high-quality data transitions with zeros in SDP.

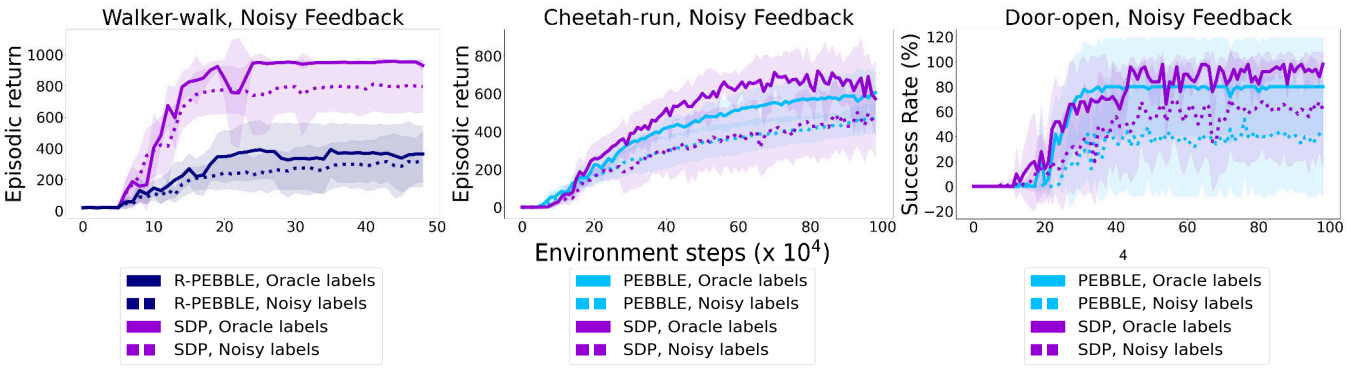


Figure 11: Imperfect feedback study: These figures compare the effect of noisy teachers on SDP and PEBBLE (or R-PEBBLE).

Task - Feedback	Metric	R-PEBBLE	SDP + R-PEBBLE	Deep TAMER	SAC
Walker-walk - 100	Final Perf.	362.99 ± 181.22*	931.31 ± 36.59	33.10 ± 11.10*	961.46 ± 6.21
	AUC	13147.1 ± 6102.31*	34981.65 ± 1559.24	1823.42 ± 578.48*	41973.52 ± 593.89
Cheetah-run - 60	Final Perf.	522.10 ± 238.87*	862.72 ± 49.13	30.71 ± 16.26*	794.24 ± 43.93
	AUC	33557.58 ± 15185.47*	55144.89 ± 4266.94	2459.23 ± 1226.34*	53996.18 ± 2814.56
Quadruped-walk - 200	Final Perf.	543.92 ± 193.21*	777.38 ± 156.74	73.74 ± 48.38*	912.98 ± 33.84
	AUC	37076.52 ± 11968.48*	58370.85 ± 10759.0	7365.68 ± 3556.27*	58110.69 ± 12486.76

Table 4: This table shows the performance (mean +/- 95% confidence intervals) for all scalar feedback experiments.
* indicates that SDP + the base preference learning algorithm achieves a statistically greater score ($p < 0.05$) than the base preference learning algorithm alone (i.e., without SDP).

Task - Feedback	Metric	PEBBLE	SDP + PEBBLE	RUNE	SDP + RUNE	SURF	SDP + SURF	SAC
Walker-walk - 100	Final Perf.	168.38 ± 118.07*	487.24 ± 49.71	163.05 ± 72.50*	575.35 ± 121.51	211.39 ± 109.20*	420.19 ± 49.16	961.46 ± 6.21
	AUC	7224.14 ± 3184.56*	17939.94 ± 1412.96	6232.93 ± 2724.95*	20614.29 ± 3543.59	7334.85 ± 3132.28*	16288.09 ± 1588.21	41973.52 ± 593.89
Cheetah-run - 200	Final Perf.	579.35 ± 74.58*	735.31 ± 95.76	677.57 ± 82.51	622.58 ± 317.32	645.63 ± 90.88	648.27 ± 128.68	794.24 ± 43.93
	AUC	38966.57 ± 5818.25*	51342.85 ± 3772.22	45538.51 ± 5134.30	36646.1 ± 17981.84	40234.13 ± 3388.71	42323.53 ± 4395.56	53996.18 ± 2814.56
Quadruped-walk - 500	Final Perf.	373.12 ± 282.63*	741.91 ± 105.44	377.91 ± 261.50*	700.03 ± 111.22	599.04 ± 208.38	764.71 ± 156.54	912.98 ± 33.84
	AUC	20834.8 ± 11054.15*	42074.4 ± 12172.93	22780.57 ± 10608.41	32454.73 ± 4036.69	31076.33 ± 12776.06	44115.58 ± 9547.31	58110.69 ± 12486.76
Door-unlock - 500	Final Perf.	40.0 ± 37.70	70.0 ± 29.07	24.0 ± 37.50	28.0 ± 34.17	18.0 ± 35.28*	72.0 ± 20.93	100.0 ± 0.0
	AUC	2110.0 ± 1852.85*	4400.0 ± 1170.56	1196.0 ± 1476.11	1876.0 ± 2378.14	1498.0 ± 2617.88*	4650.0 ± 1711.70	8594.0 ± 135.74
Window open - 200	Final Perf.	28.0 ± 35.82	54.0 ± 33.72	10.0 ± 19.60	28.0 ± 21.82	8.0 ± 9.60*	70.0 ± 35.61	100.0 ± 0.0
	AUC	732.0 ± 1093.36	1656.0 ± 922.40	218.0 ± 341.87	1024.0 ± 847.59	202.0 ± 252.33*	2018.0 ± 1005.70	4432.0 ± 150.22
Drawer open - 1000	Final Perf.	0.0 ± 0.0	26.0 ± 32.56	0.0 ± 0.0*	64.0 ± 18.18	20.0 ± 39.20*	76.0 ± 14.67	100.0 ± 0.0
	AUC	348.0 ± 642.97	1622.0 ± 1891.32	42.0 ± 50.51*	3860.0 ± 1597.17	714.0 ± 1374.95*	4072.0 ± 1916.90	8068.0 ± 844.38
Door open - 2000	Final Perf.	80.0 ± 39.2	98.0 ± 3.92	80.0 ± 39.20	56.0 ± 45.38	100 ± 0.0	86.0 ± 22.85	100.0 ± 0.0
	AUC	5908.0 ± 2894.91	6510.0 ± 897.07	5570.0 ± 2809.87	3424.0 ± 2587.34	7282.0 ± 490.13	6366.0 ± 714.27	9020.0 ± 254.72
Door-lock - 500	Final Perf.	34.0 ± 28.81*	90.0 ± 8.77	58.0 ± 29.98	40.0 ± 37.19	76.0 ± 37.50	80.0 ± 12.40	96.0 ± 4.8
	AUC	788.0 ± 676.40*	2274.0 ± 261.83	1266.0 ± 662.60	918.0 ± 717.44	1846.0 ± 910.79	1932.0 ± 224.79	3718.0 ± 110.25
Hammer - 7500	Final Perf.	2.0 ± 3.92*	74.0 ± 17.09	16.0 ± 31.36*	68.0 ± 29.98	0.0 ± 0.0*	52.0 ± 34.73	100.0 ± 0.0
	AUC	462.0 ± 236.13*	2122.0 ± 1104.20	466.0 ± 288.1*	2496.0 ± 684.39	536.0 ± 244.93*	3066.0 ± 1085.67	4538.0 ±2550.30

Table 5: This table shows the performance (mean +/- 95% confidence intervals) for all preference learning experiments.
* indicates that SDP + the base preference learning algorithm achieves a statistically greater score ($p < 0.05$) than the base preference learning algorithm alone (i.e., without SDP).