DRS: LEARNING REUSABLE DENSE REWARDS FOR MULTI-STAGE TASKS

Tongzhou Mu, Minghua Liu, Hao Su
UC San Diego
{t3mu,mil070,haosu}@ucsd.edu

ABSTRACT

The success of many RL techniques heavily relies on human-engineered dense rewards, which typically demands substantial domain expertise and extensive trial and error. In our work, we propose **DrS** (**D**ense **r**eward learning from **S**tages), a novel approach for learning *reusable* dense rewards for multi-stage tasks in a data-driven manner. By leveraging the stage structures of the task, DrS learns a high-quality dense reward from sparse rewards and demonstrations if given. The learned rewards can be *reused* in unseen tasks, thus reducing the human effort for reward engineering. Extensive experiments on three physical robot manipulation task families with 1000+ task variants demonstrate that our learned rewards can be reused in unseen tasks, resulting in improved performance and sample efficiency of RL algorithms. The learned rewards even achieve comparable performance to human-engineered rewards on some tasks. See our project page for videos.

1 INTRODUCTION

The success of many reinforcement learning (RL) techniques heavily relies on dense reward functions (Hwangbo et al., 2019; Peng et al., 2018), which are often tricky to design by humans due to heavy domain expertise requirements and tedious trials and errors. In contrast, sparse rewards, such as a binary task completion signal, are significantly easier to obtain (often directly from the environment). For instance, in pick-and-place tasks, the sparse reward could simply be defined as the object being placed at the goal location. Nonetheless, sparse rewards also introduce challenges (e.g., exploration) for RL algorithms (Pathak et al., 2017; Burda et al., 2018; Ecoffet et al., 2019). Therefore, a crucial question arises: *can we learn dense reward functions in a data-driven manner?*

Ideally, the learned reward will be **reused** to efficiently solve new tasks that share similar success conditions with the task used to learn the reward. For example, in pick-and-place tasks, different objects may need to be manipulated with varying dynamics, action spaces, and even robot morphologies. For clarity, we refer to each variant as a *task* and the set of all possible pick-and-place tasks as a *task family*. Importantly, the reward function, which captures approaching, grasping, and moving the object toward the goal position, can potentially be transferred within this task family. This observation motivates us to explore the concept of *reusable rewards*, which can be learned as a function from some tasks and reused in unseen tasks. While existing literature in RL primarily focuses on the reusability (generalizability) of policies, we argue that rewards can pose greater flexibility for reuse across tasks. For example, it is nearly impossible to directly transfer a policy operating a two-finger gripper for pick-and-place to a three-finger gripper due to action space misalignment, but a reward inducing the approach-grasp-move workflow may apply for both types of grippers.

However, many existing works on reward learning do not emphasize reward reuse for new tasks. The field of learning a reward function from demonstrations is known as inverse RL in the literature (Ng et al., 2000; Abbeel & Ng, 2004; Ziebart et al., 2008). More recently, adversarial imitation learning (AIL) approaches have been proposed (Ho & Ermon, 2016; Kostrikov et al., 2018; Fu et al., 2017; Ghasemipour et al., 2020) and gained popularity. Following the paradigm of GANs (Goodfellow et al., 2020), AIL approaches employ a policy network to generate trajectories and train a discriminator to distinguish between agent trajectories from demonstration ones. By using the discriminator score as rewards, (Ho & Ermon, 2016) shows that a policy can be trained to imitate the demonstrations. Unfortunately, such rewards are *not reusable* across tasks – at convergence, the discriminator outputs

$\frac{1}{2}$ for both the agent trajectories and the demonstrations, as discussed in (Goodfellow et al., 2020; Fu et al., 2017), making it unable to learn useful information for solving new tasks.

In contrast to AIL, we propose a novel approach for learning *reusable rewards*. Our approach involves incorporating sparse rewards as a supervision signal in lieu of the original signal used for classifying demonstration and agent trajectories. Specifically, we train a discriminator to classify *success trajectories* and *failure trajectories* based on the binary sparse reward. Please refer to Fig. 2 (a)(b) for an illustrative depiction. Our formulation assigns higher rewards to transitions in success trajectories and lower rewards to transitions within failure trajectories, which is consistent throughout the entire training process. As a result, the reward will be reusable once the training is completed. Expert demonstrations can be included as success trajectories in our approach, though they are not mandatory. We only require the availability of a sparse reward, which is a relatively weak requirement as it is often an inherent component of the task definition.

Our approach can be extended to leverage the inherent structure of *multi-stage tasks* and derive stronger dense rewards. Many tasks naturally exhibit multi-stage structures, and it is relatively easy to assign a binary indicator on whether the agent has entered a stage. For example, in the "Open Cabinet Door" task depicted in Fig. 1, there are three stages: 1) approach the door handle, 2) grasp the handle and pull the door, and 3) release the handle and keeping it steady. If the agent is grasping the handle of the door but the door has not been opened enough, then we can simply use a corresponding binary indicator asserting that the agent is in the 2nd stage. [1] By utilizing these stage indicators, we can learn a dense reward for each stage and combine them into a more structured reward. Since the horizon for each stage is shorter than that of the entire task, learning a high-quality dense reward becomes more feasible. Furthermore, this approach provides flexibility in incorporating extra information beyond the final success signal. We dub our approach as **DrS** (**D**ense **r**eward learning from **S**tages).

Our approach exhibits strong performance on challenging tasks. To assess the reusability of the rewards learned by our approach, we employ the ManiSkill benchmark (Mu et al., 2021; Gu et al., 2023), which offers a large number of task variants within each task family.We evaluate our approach on three task families: Pick-and-Place, Open Cabinet Door, and Turn Faucet, including 1000+ task variants. Each task variant involves manipulating a different object and requires precise low-level physical control, thereby highlighting the need for a good dense reward. Our results demonstrate that the learned rewards can be reused across tasks, leading to improved performance and sample efficiency of RL algorithms compared to using sparse rewards. In certain tasks, the learned rewards even achieve performance comparable to those attained by human-engineered reward functions.

Moreover, our approach **drastically reduces the human effort** needed for reward engineering. For instance, while the human-engineered reward for "Open Cabinet Door" involves *over 100 lines of code, 10 candidate terms, and tons of "magic" parameters*, our approach only requires *two boolean functions* as stage indicators: if the robot has grasped the handle and if the door is open enough. See appendix B for a detailed example illustrating how our method reduces the required human effort.

Our contributions can be summarized as follows:

- We propose **DrS** (**D**ense **r**eward learning from **S**tages), a novel approach for learning reusable dense rewards for multi-stage tasks, effectively reducing human efforts in reward engineering.
- Extensive experiments on 1,000+ task variants from three task families showcase the effectiveness of our approach in generating high-quality and reusable dense rewards.

2 Related Works

Learning Reward from Demo (Offline) Designing rewards is challenging due to domain knowledge requirements, so approaches to learning rewards from data have gained attention. Some methods adopt classification-based rewards, i.e., training a reward by classifying goals (Smith et al., 2019; Kalashnikov et al., 2021; Du et al., 2023) or demonstration trajectories (Zolna et al., 2020). Other methods (Zakka et al., 2022; Aytar et al., 2018) use the distance to goal as a reward function, where the distance is usually computed in a learned embedding space, but these methods usually require that the goal never changes in a task. These rewards are only trained on offline datasets, hence they can *easily be exploited* by an RL agent, i.e., an RL can enter a state that is not in the dataset and get a wrong reward signal, as studied in (Vecerik et al., 2019; Xu & Denil, 2021).

[1] Stage indicators are only required during RL training, but **not required** when deploying policy to real world.

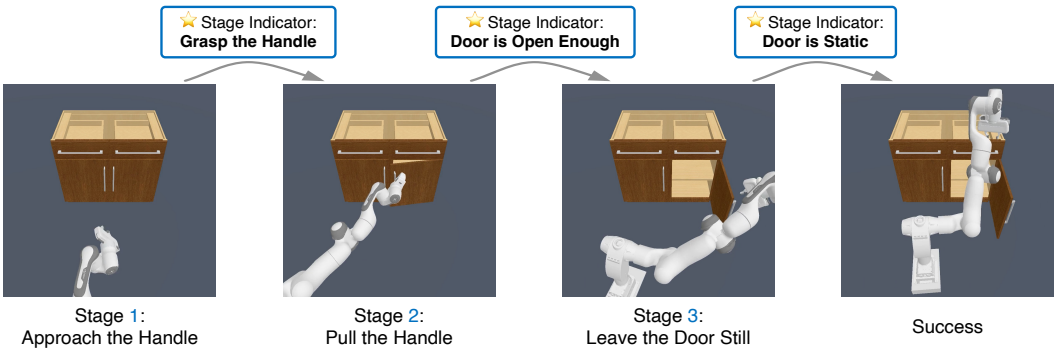


Figure 1: An illustration of stage indicators in an OpenCabinetDoor task, which can be naturally divided into three stages plus a success state. A stage indicator is a binary function representing whether the current state is in a certain stage, and it can be simply defined by some boolean functions.

Learning Reward from Demo (Online) The above issue can be addressed by allowing agents to verify the reward in the environment, and inverse reinforcement learning (IRL) is the prominent paradigm. IRL aims to recover a reward function given expert demonstrations. Traditional IRL methods (Ng et al., 2000; Abbeel & Ng, 2004; Ziebart et al., 2008; Ratliff et al., 2006) often require multiple iterations of Markov Decision Process solvers (Puterman, 2014), resulting in poor sample efficiency. In recent years, adversarial imitation learning (AIL) approaches are proposed (Ho & Ermon, 2016; Kostrikov et al., 2018; Fu et al., 2017; Ghasemipour et al., 2020; Finn et al., 2016). They operate similarly to generative adversarial networks (GANs) (Goodfellow et al., 2020), in which a generator (the policy) is trained to maximize the confusion of a discriminator, and the discriminator (serves the role of rewards) is trained to classify the agent trajectories and demonstrations. However, such rewards are *not reusable* as we discussed in the introduction - classifying agent trajectories and demonstrations is impossible at convergence. In contrast, our approach gets rid of this issue by classifying the success/failure trajectories instead of expert/agent trajectories.

Learning Reward from Human Feedback Recent studies (Christiano et al., 2017; Ibarz et al., 2018; Jain et al., 2013) infer the reward through human preference queries on trajectories or explicitly asking for trajectory rankings (Brown et al., 2019). Another line of works (Fu et al., 2018; Singh et al., 2019) involves humans specifying desired outcomes or goals to learn rewards. However, in these methods, the rewards only distinguish goal from non-goal states, offering relatively weak incentives to agents at the beginning of an episode, especially in long-horizon tasks. In contrast, our approach classifies all the states in the trajectories, providing strong guidance throughout the entire episode.

Reward Shaping Reward shaping methods aim to densify sparse rewards. Earlier works (Ng et al., 1999) study the forms of shaped rewards that induce the same optimal policy as the ground-truth reward. Recently, some works (Trott et al., 2019; Wu et al., 2021) have shaped the rewards as the distance to the goal, similar to some offline reward learning methods mentioned above. Another idea (Memarian et al., 2021) involves shaping delayed reward by ranking trajectories based on a fine-grained preference oracle. In contrast to these reward shaping approaches, our method leverages demonstrations, which are available in many real-world problems (Sun et al., 2020; Dasari et al., 2019). This not only boosts the reward learning process but also reduces the additional domain knowledge required by these methods.

Task Decomposition The decomposition of tasks into stages/sub-tasks has been explored in various domains. Hierarchical RL approaches (Frans et al., 2018; Nachum et al., 2018; Levy et al., 2018) break down policies into sub-policies to solve specific sub-tasks. Skill chaining methods (Lee et al., 2021; Gu et al., 2022; Lee et al., 2019) focus on solving long-horizon tasks by combining multiple short-horizon policies or skills. Recently, language models have also been utilized to break the whole task into sub-tasks Ahn et al. (2022). In contrast to these approaches that utilize stage structures in policy space, our work explores an orthogonal direction by designing rewards with stage structures.

3 Problem Setup

In this work, we adopt the Markov Decision Process (MDP) $\mathcal{M} := \langle S, A, T, R, \gamma \rangle$ as the theoretical framework, where R is a reward function that defines the goal or purpose of a task. Specifically, we focus on tasks with *sparse rewards*. In this context, "sparse reward" denotes a binary reward function that gives a value of 1 upon successful task completion and 0 otherwise:

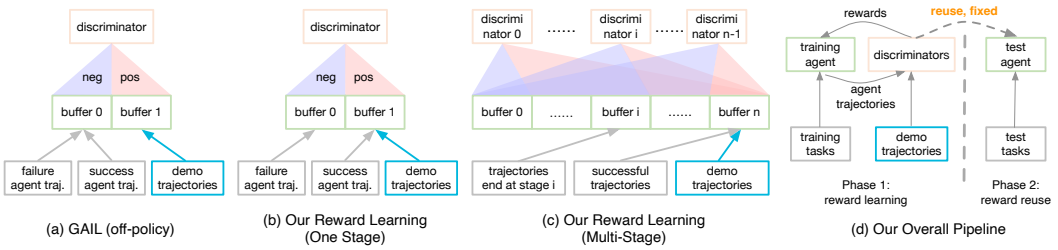


Figure 2: a) GAIL's discriminator aims to distinguish agent trajectories from demonstrations. b) In single-stage tasks, the discriminator in our approach aims to distinguish success trajectories from failure ones. c) In multi-stage tasks, our approach train a separate discriminator for each stage. The discriminator for stage k aims to distinguish trajectories that reach beyond stage k from those that only reach up to stage k. d) Overall, our approach has 2 phases: **reward learning** and **reward reuse**.

$$R_{sparse}(s) = \begin{cases} 1 & \text{task is completed by reaching one of the success states } s \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Our objective is to learn a dense reward function from a set of training tasks, with the intention of reusing it for unseen test tasks. Specifically, we aim to successfully train RL agents from scratch on the test tasks using the learned rewards. The desired outcome is to enhance the efficiency of RL training, surpassing the performance achieved by sparse rewards.

We assume that both the training and test tasks are in the same *task family*. A task family refers to a set of task variants that share the same success criteria, but may differ in terms of assets, initial states, transition functions, and other factors. For instance, the task family of object grasping includes tasks such as "Alice robot grasps an apple" and "Bob robot grasps a pen." The key point is that tasks within the same task family share a common underlying sparse reward.

Additionally, we posit that the task can be segmented into multiple stages, and the agent has access to several *stage indicators* obtained from the environment. A stage indicator is a binary function that indicates whether the current state corresponds to a specific stage of the task. An example of stage indicators is in Fig. 1. This assumption is quite general as many long-term tasks have multi-stage structures, and determining the current stage of the task is not hard in many cases. By utilizing these stage indicators, it becomes possible to construct a reward that is slightly denser than the binary sparse reward, which we refer to as a *semi-sparse* reward, and it serves as a strong baseline:

$$R_{semi-sparse}(s) = k, \text{ when state } s \text{ is at stage } k \tag{2}$$

We aim to design an approach that learns a dense reward based on the stage indicators. When expert demonstration trajectories are available, they can also be incorporated to boost the learning process.

Note that the stage indicators are only required during RL training, but *not required when deploying the policy to the real world*. Training RL agents directly in the real world is often impractical due to cost and safety issues. Instead, a more common practice is to train the agent in simulators and then transfer/deploy it to the real world. While obtaining the stage indicators in simulators is fairly easy, it is also possible to obtain them in the real world by various techniques (robot proprioception, tactile sensors Lin et al. (2022); Melnik et al. (2021), visual detection/tracking Kalashnikov et al. (2018; 2021), large vision-language models Du et al. (2023), etc.).

4 DrS: Dense reward learning from Stages

Dense rewards are often tricky to design by humans (see an example in appendix B), so we aim to learn a reusable dense reward function from stage indicators in multi-stage tasks and demonstrations when available. Overall, our approach has two phases, as shown in Fig. 2 (d):

- **Reward Learning Phase**: learn the dense reward function using training tasks.
- **Reward Reuse Phase**: reuse the learned dense reward to train new RL agents in test tasks.

Since the reward reuse phase is just a regular RL training process, we only discuss the reward learning phase in this section. We first explain how our approach learns a dense reward in one-stage tasks (Sec. 4.1). Then, we extend this approach to multi-stage tasks (Sec. 4.2).

4.1 Reward Learning on One-Stage Tasks

In line with previous work (Vecerik et al., 2019; Fu et al., 2018), we employ a classification-based dense reward. We train a classifier to distinguish between good and bad trajectories, utilizing the learned classifier as dense reward. Essentially, states resembling those in good trajectories receive higher rewards, while states resembling bad trajectories receive lower rewards. While previous Adversarial Imitation Learning (AIL) methods (Ho & Ermon, 2016; Kostrikov et al., 2018) used discriminators as classifiers/rewards to distinguish between agent and demonstration trajectories, these discriminators cannot be directly *reused* as rewards to train new RL agents. As the policy improves, the agent trajectories (negative data) and the demonstrations (positive data) can become nearly identical. Therefore, at convergence, the discriminator output for both agent trajectories and demonstrations tends to approach $\frac{1}{2}$, as observed in GANs (Goodfellow et al., 2020) (also noted by (Fu et al., 2017; Xu & Denil, 2021)). This makes it unable to learn useful info for solving new tasks.

Our approach introduces a simple modification to existing AIL methods to ensure that the discriminator continues to learn meaningful information even at convergence. The key issue previously mentioned arises from the diminishing gap between agent and demonstration trajectories over time, making it challenging to differentiate between positive and negative data. To address this, we propose training the discriminator to distinguish between success and failure trajectories instead of agent and demonstration trajectories. By defining success and failure trajectories based on the sparse reward signal from the environment, the gap between them remains intact and does not shrink. Consequently, the discriminator effectively emulates the sparse reward signal, providing dense reward signals to the RL agent. Intuitively, a state that is closer to the success states in terms of task progress (rather than Euclidean distance) receives a higher reward, as it is more likely to occur in success trajectories. Fig. 2(a) and (b) illustrate the distinction between our approach and traditional AIL methods.

To ensure that the training data consistently includes both success and failure trajectories, we use replay buffers to store historical experiences, and train the discriminator in an off-policy manner. While the original GAIL is on-policy, recent AIL methods (Kostrikov et al., 2018; Orsini et al., 2021) have adopted off-policy training for better sample efficiency. Note that although our approach shares similarities with AIL methods, it is not adversarial in nature. In particular, our policy does not aim to deceive the discriminator, and the discriminator does not seek to penalize the agent's trajectories.

4.2 Reward Learning on Multi-Stage Tasks

In multi-stage tasks, it is desirable for the reward of a state in stage $k+1$ to be strictly higher than that of stage k to incentivize the agent to progress towards later stages. The semi-sparse reward (Eq. 2) aligns with this intuition, but it is still a bit too sparse. If each stage of the task is viewed as an individual task, the semi-sparse reward acts as a sparse reward for each stage. In the case of a one-stage task, a discriminator can be employed to provide a dense reward. Similarly, for multi-stage tasks, a separate discriminator can be trained for each stage to serve as a dense reward for that particular stage. By training stage-specific discriminators, we can effectively address the sparse reward issue and guide the agent's progress through the different stages of the task. Fig. 3 gives an intuitive illustration of our learned reward, which fills the gaps in semi-sparse rewards, resulting in a smooth reward curve.

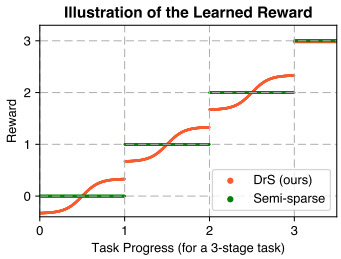


Figure 3: An illustration of our learned reward, which fills the gaps in semi-sparse rewards, resulting in a smooth reward curve.

To train the discriminators for different stages, we need to establish the positive and negative data for each discriminator. In one-stage tasks, positive data comprises success trajectories and negative data encompasses failure trajectories. In multi-stage tasks, we adopt a similar approach with a slight modification. Specifically, we assign a stage index to each trajectory, which is determined as the highest stage index among all states within the trajectory:

$$\text{StageIndex}(\tau : (s_0, s_1, ...)) = \max_i \text{ StageIndex}(s_i), \tag{3}$$

where τ is a trajectory and s_i are the states in τ. For the discriminator associated with stage k, positive data consists of trajectories that progress beyond stage k (StageIndex $> k$), and negative data consists of trajectories that reach up to stage k (StageIndex $\leq k$).

Once the positive and negative data for each discriminator have been established, the next step is to combine these discriminators to create a reward function. While the semi-sparse reward (Eq. 2) lacks incentives for the agent at stage k until it reaches stage $k+1$, we can fill in the gaps in the semi-sparse reward by the stage-specific discriminators. We define our learned reward function for a multi-stage task as follows:

$$R(s') = k + \alpha \cdot \tanh(\text{Discriminator}_k(s')) \tag{4}$$

where k is the stage index of s' and α is a hyperparameter. Basically, the formula incorporates a dense reward term into the semi-sparse reward. The $\tanh$ function is used to bound the output of the discriminators. As the range of the $\tanh$ function is (-1, 1), any $\alpha < \frac{1}{2}$ ensures that the reward of a state in stage $k+1$ is always higher than that of stage k. In practice, we use $\alpha = \frac{1}{3}$ and it works well.

4.3 Implementation

From the implementation perspective, our approach is similar to GAIL, but with a different training process for discriminators. While the original GAIL is combined with TRPO (Schulman et al., 2015), (Orsini et al., 2021) found that using state-of-the-art off-policy RL algorithms (like SAC (Haarnoja et al., 2018) or TD3 (Fujimoto et al., 2018)) can greatly improve the sample efficiency of GAIL. Therefore, we also combine our approach with SAC, and the full algorithm is summarized in Algo. 1.

Algorithm 1 DrS (**D**ense **r**eward learning from **S**tages)

Require: Task MDP $\mathcal{M}$, Number of stages in task N, Demonstration dataset $\mathcal{D} := \{\tau^0, \tau^1, ...\}$ (optional)
Initialize policy π, critic Q, replay buffer $\mathcal{B}_R$
Initialize discriminators $f_0, f_1, ..., f_{N-1}$, stage buffers $\mathcal{B}_0, \mathcal{B}_2, .., \mathcal{B}_N$
Fill demo $\mathcal{D}$ into $\mathcal{B}_\mathcal{N}$: $\mathcal{B}_\mathcal{N} \leftarrow \mathcal{B}_\mathcal{N} \cup \mathcal{D}$
for each iteration **do**
  Collect trajectories $\{\tau_\pi^0, \tau_\pi^1, ...\}$ by executing π in $\mathcal{M}$
  Add trajectories to replay buffer: $\mathcal{B}_R \leftarrow \mathcal{B}_R \cup \{\tau_\pi^0, \tau_\pi^1, ...\}$
  for each trajectory τ_π^i in $\{\tau_\pi^0, \tau_\pi^1, ...\}$ **do**
    $j = \text{StageIndex}(\tau_\pi^i)$ according to Eq. 3
    $\mathcal{B}_j \leftarrow \mathcal{B}_j \cup \{\tau_\pi^i\}$
  for each gradient step for discriminators **do**
    for each discriminator f_k **do**
      Sample negative data from $\bigcup_{i=0}^{k} \mathcal{B}_i$
      Sample positive data from $\bigcup_{i=k+1}^{N} \mathcal{B}_i$
      Update f_k using BCE loss
  for each gradient step for the policy π **do**
    Sample from $\mathcal{B}_R$
    Compute rewards according to Eq. 4
    Update π and Q by SAC (Haarnoja et al., 2018)

In addition to the regular replay buffer used in SAC, our approach maintains N different stage buffers to store trajectories corresponding to different stages(defined by Eq. 3). Each trajectory is assigned to only one stage buffer based on its stage index. During the training of the discriminators, we sample data from the union of multiple buffers. Note that our approach uses the next state s' as the input to the reward, which aligns with common practices in human reward engineering (Gu et al., 2023; Zhu et al., 2020). However, our approach is also compatible with alternative forms of input, such as (s, a) or (s, a, s').

5 Experiments

5.1 Setup and Task Descriptions

We evaluated our approach on three challenging physical manipulation task families from the ManiSkill (Mu et al., 2021; Gu et al., 2023): Pick-and-Place, Turn Faucet, and Open Cabinet Door. Each task family includes a set of different objects to be manipulated. To assess the reusability of the learned rewards, we divided the objects within each task family into non-overlapping training and test sets, as depicted in Fig.4. During the reward learning phase, we learned the rewards by training an agent for each task family to manipulate all training objects. In the subsequent reward reuse phase, the learned reward rewards are reused to train an agent to manipulate all test objects for each task family. And we compare with other baseline rewards in this **reward reuse** phase. It is important to note that our learned rewards are agnostic to the specific RL algorithm employed. However, we utilized the Soft Actor-Critic (SAC) algorithm to evaluate the quality of the different rewards.

To assess the *reusability* of the learned rewards, it is crucial to have a diverse set of tasks that exhibit similar structures and goals but possess variations in other aspects. However, most existing benchmarks lack an adequate number of task variations within the same task family. As a result, we primarily conducted our evaluation on the ManiSkill benchmark, which offers a range of object

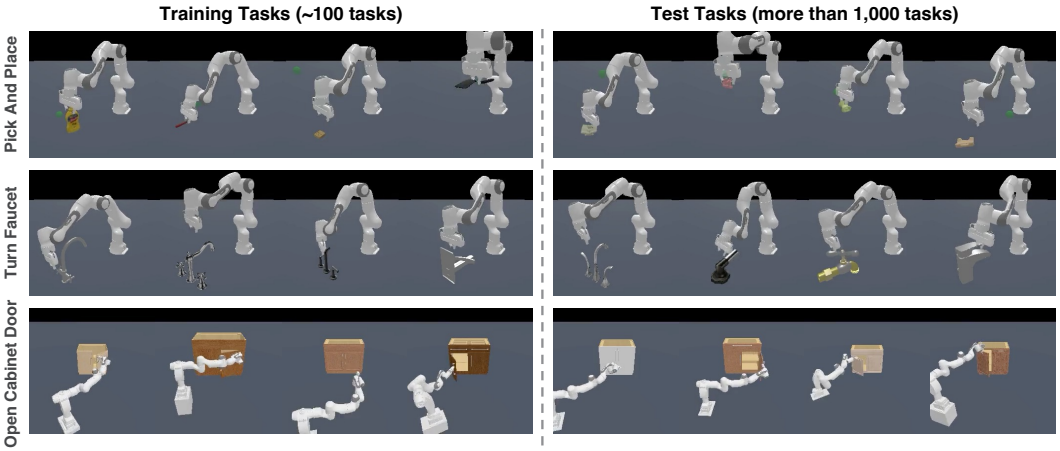


Figure 4: We evaluated our approach DrS on more than 1,000 task variants from three task families in ManiSkill (Gu et al., 2023). Each task variant is associated with a different object. All tasks require low-level physical control. The objects in training tasks and test tasks are non-overlapped.

variations within each task family. This allowed us to thoroughly evaluate our learned rewards in a realistic and comprehensive manner.

Pick-and-Place: A robot arm is tasked with picking up an object and relocating it to a random goal position in mid-air. The task is completed if the object is in close proximity to the goal position, and both the robot arm and the object remain stationary. The stage indicators include: (a) the gripper grasps the object, (b) the object is close the goal position, and (c) both the robot and the object are stationary. We learn rewards on 74 YCB objects and reuse rewards on 1,600 EGAD objects.

Turn Faucet: A robot arm is tasked to turn on a faucet by rotating its handle. The task is completed if the handle reaches a target angle. The stage indicators include: (a) the target handle starts moving, (b) the handle reaches a target angle. We learn rewards on 10 faucets and reuse rewards on 50 faucets.

Open Cabinet Door: A single-arm mobile robot is required to open a designated target door on a cabinet. The task is completed if the target door is opened to a sufficient degree and remains stationary. The stage indicators include: (a) the robot grasps the door handle, (b) the door is open enough, and (c) the door is stationary. We learn rewards on 4 cabinet doors and reuse rewards on 6 cabinet doors. Note that we remove all single-door cabinets in this task family, as they can be solved by kicking the side of the door and this behavior can be readily learned by sparse rewards.

We employed low-level physical control for all task families. Please refer to the appendix A for a detailed description of the object sets, action space, state space, and demonstration trajectories.

5.2 Baselines

Human-Engineered The original human-written dense rewards in the benchmark, which require a significant amount of domain knowledge, thus can be considered as an *upper bound* of performance.

Semi-Sparse The rewards constructed based on the stage indicators, as discussed in Eq. 2. The agent receives a reward of k when it is in stage k. This baseline extends the binary sparse reward.

VICE-RAQ (Singh et al., 2019) An improved version of VICE (Fu et al., 2018). It learns a classifier, where the positive samples are successful states annotated by querying humans, and the negative samples are all other states collected by the agent. Since our experiments do not involve human feedback, we let VICE-RAQ query the oracle success condition infinitely for a fair comparison.

ORIL (Zolna et al., 2020) A representative offline reward learning method, where the agent does not interact with the environments but purely learns from the demonstrations. It learns a classifier (reward) to distinguish between the states from success trajectories and random trajectories.

5.3 Comparison with Baseline Rewards

We trained RL agents using various rewards and assessed the reward quality based on both the sample efficiency and final performance of the agents. The experimental results, depicted in Fig. 5, demonstrate that our learned reward surpasses semi-sparse rewards and all other reward learning

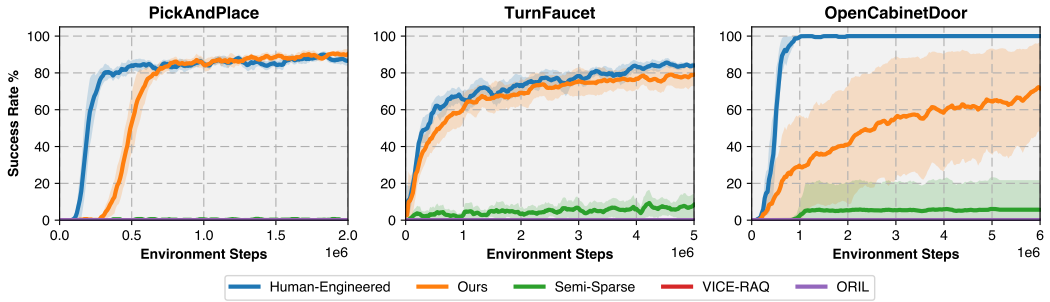


Figure 5: Evaluation results of **reusing learned rewards**. All curves use SAC to train, but with different rewards. VICE-RAQ and ORIL get no success. 5 random seeds, the shaded region is std.

methods across all three task families. This outcome suggests that our approach successfully acquires high-quality rewards that significantly enhance RL training. Remarkably, our learned rewards even achieve performance comparable to human-engineered rewards in Pick-and-Place and Turn Faucet.

Semi-sparse rewards yielded limited success within the allocated training budget, suggesting that RL agents face exploration challenges when confronted with sparse reward signals. VICE-RAQ failed in all tasks. Notably, it actually failed during the reward learning phase on the training tasks, rendering the learned rewards inadequate for supporting RL training on the test tasks. This failure aligns with observations made by (Wu et al., 2021). We hypothesize that by only classifying the success states from other states, it cannot provide sufficient guidance during the early stages of training, where most states are distant from the success states and receive low rewards. Unsurprisingly, ORIL does not get any success on all tasks either. Without interacting with the environments to gather more data, the learned reward functions easily tend to overfit the provided dataset. When using such rewards in RL, the flaws in the learned rewards are easily exploited by the RL agents.

5.4 Ablation Study

We examined various design choices within our approach on the Pick-and-Place task family.

5.4.1 Robustness to Stage Configurations

Though many tasks present a natural structure of stages, there are still different ways to divide a task into stages. To assess the robustness of our approach in handling different task structures, we experiment with different numbers of stages and different ways to define stage indicators.

Number of Stages The Pick-and-Place task family originally consisted of three stages: (a) approach the object, (b) move the object to the goal, and (c) make everything stationary. We explored two ways of reducing the number of stages to two, namely merging stages (a) and (b) or merging stages (b) and (c), as well as the 1-stage case. Our results, presented in Fig. 6, indicate that the learned rewards with 2 stages can still effectively train RL agents in test tasks, albeit with lower sample efficiency than those with 3 stages. Specifically, the reward that preserves stage (c) "make everything stationary" performs slightly better than the reward that preserves stage (a) "approach the object". This suggests that it may be more challenging for a robot to learn to stop abruptly without a dedicated stage. However, when reducing the number of stages to 1, the learned reward failed to train RL agents in test tasks, demonstrating the benefit of using more stages in our approach.

Definition of Stages The stage indicator "object is placed" is initially defined as if the distance between the object and the goal is less than 2.5 cm. We create two variants of it, where the distance thresholds are 5cm and 10cm, respectively. The results, as depicted in Fig. 7, demonstrate that changing the distance threshold within a reasonable range does not significantly affect the efficiency of RL training. Note that the task success condition is unchanged, and our rewards consistently encourage the agents to reach the success state as it yields the highest reward according to Eq. 4. The stage definitions solely affect the efficiency of RL training during the reward reuse phase.

Overall, the above results highlight the robustness of our approach to different stage configurations, indicating that it is not heavily reliant on intricate stage designs. This robustness contributes to a significant reduction in the burden of human reward engineering.

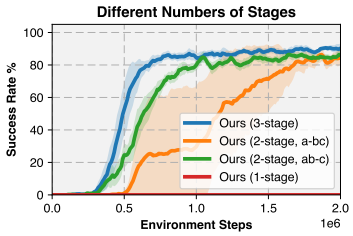


Figure 6: Ablation study on the number of stages, see here.

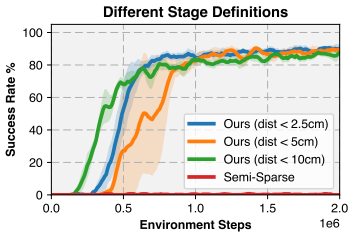


Figure 7: Ablation study on the stage definitions, see here.

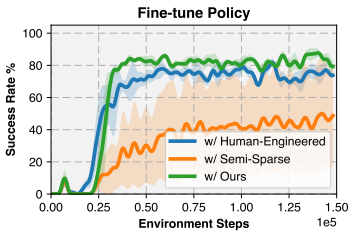


Figure 8: Fine-tune the policy from reward learning, see here.

5.4.2 Fine-tuning Policy

In our previous experiments, we assessed the quality of the learned reward by reusing it in training RL agents from scratch since it is the most common and natural way to use a reward. However, our approach also produces a policy as a byproduct in the reward learning phase. This policy can also be fine-tuned using various rewards in new tasks, providing an alternative to training RL agents from scratch. We compare the fine-tuning of the byproduct policy using human-engineered rewards, semi-sparse rewards, and our learned rewards.

As shown in Fig. 8, all policies improve rapidly at the beginning due to the good initialization of the policies. However, fine-tuning with our learned reward yields the best performance (even slightly better than the human-engineered reward), indicating the advantages of utilizing our learned dense reward even with a good initialization. Furthermore, the significant variance observed when fine-tuning the policy with semi-sparse rewards highlights the limitations of sparse reward signals in effectively training RL agents, even with a very good initialization.

5.4.3 Additional Ablation Studies

Additional ablation studies are provided in appendix E, with key conclusions summarized as follows:

- DrS is compatible with various modalities of reward input, including point cloud data. E.1
- Reward learned by GAIL, even with stage indicators, is not reusable. E.2
- The way of combining the dense rewards from each stage matters. E.3

6 Conclusion and Limitations

To make RL a more widely applicable tool, we have developed a data-driven approach for learning dense reward functions that can be reused in new tasks from sparse rewards. We have evaluated the effectiveness of our approach on robotic manipulation tasks, which have high-dimensional action spaces and require dense rewards. Our results indicate that the learned dense rewards are effective in transferring across tasks with significant variation in object geometry. By simplifying the reward design process, our approach paves the way for scaling up RL in diverse scenarios.

We would like to discuss two main limitations when using the multi-stage version of our approach.

Firstly, though our experiments show the substantial benefits of knowing the multi-stage structure of tasks (at training time, not needed at policy deployment time), we did not specifically investigate how this knowledge can be acquired. Much future work on be done here, by leveraging large language models such as ChatGPT (OpenAI, 2023) (by our testing, they suggest stages highly aligned to the ones we adopt by intuition for all tasks in this work) or employing information-theoretic approaches. Further discussions regarding this point can be found in appendix F.

Secondly, the reliance on stage indicators adds a level of inconvenience when directly training RL agents in the real world. While it is infrequent to directly train RL agents in the real world due to cost and safety issues, when necessary, stage information can still be obtained using existing techniques, similar to (Kalashnikov et al., 2018; 2021). For example, the "object is grasped" indicator can be acquired by tactile sensors (Lin et al., 2022; Melnik et al., 2021), and the "object is placed" indicator can be obtained by forward kinematics, visual detection/tracking techniques (Kalashnikov et al., 2018; 2021), or even large vision-language models (Du et al., 2023).

REFERENCES

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1, 2004.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando De Freitas. Playing hard exploration games by watching youtube. *Advances in neural information processing systems*, 31, 2018.

Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pp. 783–792. PMLR, 2019.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*, 2019.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*, 2023.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pp. 49–58. PMLR, 2016.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. *Advances in neural information processing systems*, 31, 2018.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pp. 1259–1277. PMLR, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill mobile manipulation for object rearrangement. *arXiv preprint arXiv:2209.02778*, 2022.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiaing Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023.

Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. *arXiv preprint arXiv:2307.14535*, 2023.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. *Advances in neural information processing systems*, 26, 2013.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pp. 651–673. PMLR, 2018.

Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *arXiv preprint arXiv:1809.02925*, 2018.

Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2019.

Youngwoon Lee, Joseph J Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. *arXiv preprint arXiv:2111.07999*, 2021.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2018.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Yijiong Lin, John Lloyd, Alex Church, and Nathan F Lepora. Tactile gym 2.0: Sim-to-real deep reinforcement learning for comparing low-cost high-resolution robot touch. *IEEE Robotics and Automation Letters*, 7(4):10754–10761, 2022.

Andrew Melnik, Luca Lach, Matthias Plappert, Timo Korthals, Robert Haschke, and Helge Ritter. Using tactile sensing to improve the sample efficiency and performance of deep deterministic policy gradients for simulated in-hand manipulation tasks. *Frontiers in Robotics and AI*, 8:538773, 2021.

Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Scott Niekum, and Ufuk Topcu. Self-supervised online reward shaping in sparse-reward environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2369–2375. IEEE, 2021.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 31:3303–3313, 2018.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287, 1999.

Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

OpenAI. Gpt-4 technical report, 2023.

Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning? *Advances in Neural Information Processing Systems*, 34: 14656–14668, 2021.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 729–736, 2006.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

Lucy Xiaoyang Shi, Archit Sharma, Tony Z Zhao, and Chelsea Finn. Waypoint-based imitation learning for robotic manipulation. *arXiv preprint arXiv:2307.14326*, 2023.

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine. Avid: Learning multi-stage tasks via pixel-level translation of human videos. *arXiv preprint arXiv:1912.04443*, 2019.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2446–2454, 2020.

Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *2019 international conference on robotics and automation (ICRA)*, pp. 754–760. IEEE, 2019.

Zheng Wu, Wenzhao Lian, Vaibhav Unhelkar, Masayoshi Tomizuka, and Stefan Schaal. Learning dense rewards for contact-rich manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6214–6221. IEEE, 2021.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Automated dense reward function generation for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023.

Danfei Xu and Misha Denil. Positive-unlabeled reward learning. In *Conference on Robot Learning*, pp. 205–219. PMLR, 2021.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, pp. 537–546. PMLR, 2022.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

Konrad Zolna, Alexander Novikov, Ksenia Konyushkova, Caglar Gulcehre, Ziyu Wang, Yusuf Aytar, Misha Denil, Nando de Freitas, and Scott Reed. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.

A TASK DESCRIPTIONS

For all tasks, we use consistent setups for state spaces, action spaces, and demonstrations. The state spaces adhere to a standardized template that includes proprioceptive robot state information, such as joint angles and velocities of the robot arm, and, if applicable, the mobile base. Additionally, task-specific goal information is included within the state. Please refer to the ManiSkill paper (Gu et al., 2023) for more details. Below, we present the key details pertaining to the tasks used in this paper.

A.1 PICK-AND-PLACE

- Stage Indicators:
  - Object is grasped: Both of the robot fingers contact the object, and the impulse (force) at the contact points is non-zero.
  - Object is placed: The distance between the object and the goal position is less than 2.5 cm. This is given by the success signal of the original task, not designed by us.
  - Robot and object are stationary: The joint velocities of all robot joints are less than 0.2 rad/s. The object velocity is less than 3 cm/s. *This is given by the success signal of the original task, not designed by us.*
- Object Set: The objects in training tasks are from the YCB dataset, including 74 objects. And the objects in test tasks are from the EGAD dataset, including around 1600 objects.
- Action Space: Delta position of the end-effector and the joint positions of the gripper.
- Representation of the target object: The object to be picked is represented by its pose.
- Demonstrations: We use 200 demonstration trajectories in total for this task family (around 2.7 trajectories per task). The demonstrations are from a trained RL agent.

A.2 TURN FAUCET

- Stage Indicators:
  - Handle is moving: The joint velocity of the target joint is greater than 0.01 rad/s.
  - Handle reached the target angle: The joint angle is greater than 90% of the limit. *This is given by the success signal of the original task, not designed by us.*
- Object Set: The objects in training and test tasks are both from the PartNet-Mobility dataset. The training tasks include 10 faucets, and the test tasks include 50 faucets.
- Action Space: Delta pose of the end-effector and joint positions of the gripper.
- Representation of the target object: The faucet handle to be turned is represented by its center of mass and joint angle.
- Demonstrations: We use 100 demonstration trajectories in total for this task family (around 10 trajectories per task). The demonstrations are from a trained RL agent.

A.3 OPEN CABINET DOOR

- Stage Indicators:
  - Handle is grasped: Both of the robot fingers contact the handle, and the impulse (force) at the contact points is non-zero.
  - Door is open enough: The joint angle is greater than 90% of the limit. This is given by the success signal of the original task, not designed by us.
  - Door is stationary: The velocity of the door is less than 0.1 m/s, and the angular velocity is less than 1 rad/s. *This is given by the success signal of the original task, not designed by us.*
- Object Set: The objects in training and test tasks are both from the PartNet-Mobility dataset. The training tasks include 4 cabinet doors, and the test tasks include 6 cabinet doors. We remove all single-door cabinets in this task family, as they can be solved by kicking the side of the door and this behavior can be readily learned by sparse rewards.

- Action Space: Joint velocities of the robot arm joints and mobile robot base, and joint positions of the gripper.
- Representation of the target object: The door to be opened is represented by the center of mass of its handle and its joint angle.
- Demonstrations: We use 200 demonstration trajectories in total for this task family (around 50 trajectories per task). The demonstrations are from a trained RL agent.

B Comparison of Human Effort: Stage Indicators vs. Human-Engineered Rewards

This section explains why designing stage indicators is much easier than designing a full dense reward.

The key challenges in reward engineering lies in **designing reward candidate terms** and **tuning associated hyperparameters**. To illustrate, let us use the "Open Cabinet Door" task familly as an example. The code of human engineered reward is in Listing 1, and the code of our stage indicators is in Listing 2.

The human-engineered reward involves the following reward candidate terms:

- Distance between the robot gripper quaternion, and a set of manually designed grasp quaternions
- Distance between robot hand and door handle
- Signed-distance between tool center point (center of two fingertips) and door handle
- Robot joint velocity
- Door handle velocity
- Door handle angular velocity
- Door joint velocity
- Door joint position
- Multiple boolean functions to determine task stages

Each reward candidate term needs 1$\sim$4 hyperparameters (e.g., normalization function, clip upper bound, clip lower bound, scaling coefficient). In total, this reward function involves **more than 20 hyperparameters** to tune. *The major effort of reward engineering is thus spent iterating over these candidate terms and tuning the hyperparameters by trail and error.* This process is laborious but critical for the success of human-engineered rewards. According to the authors of ManiSkill, they spend **over one month** crafting the dense reward for the "Open Cabinet Door" tasks.

In contrast, our stage indicators for "Open Cabinet Door" tasks only requires to design **two boolean functions**: whether the robot has grasped the handle and whether the door is open enough. The third stage indicator is given by the tasks success signal so we do not need to design it. This trims the number of hyperparameters down from 20+ to just 1 (the first boolean function requires one hyperparameter, and the second boolean function is directly taken from the task's success condition so no hyperparamters), and reduces the lines of code from 100+ to 7 (with a utility function to check grasping, which is from the original codebase).

Therefore, our approach significantly reduces the human effort required for reward engineering.

```
    def _compute_grasp_poses(self, mesh: trimesh.Trimesh, pose: sapien.
    Pose):
        # NOTE(jigu): only for axis-aligned horizontal and vertical cases
        mesh2: trimesh.Trimesh = mesh.copy()
        # Assume the cabinet is axis-aligned canonically
        mesh2.apply_transform(pose.to_transformation_matrix())

        extents = mesh2.extents
        if extents[1] > extents[2]:  # horizontal handle
            closing = np.array([0, 0, 1])
        else:  # vertical handle
```

```
            closing = np.array([0, 1, 0])

        # Only rotation of grasp poses are used. Thus, center is dummy.
        approaching = [1, 0, 0]
        grasp_poses = [
            self.agent.build_grasp_pose(approaching, closing, [0, 0, 0]),
            self.agent.build_grasp_pose(approaching, -closing, [0, 0, 0]),
        ]

        pose_inv = pose.inv()
        grasp_poses = [pose_inv * x for x in grasp_poses]

        return grasp_poses

    def _compute_handles_grasp_poses(self):
        self.target_handles_grasp_poses = []
        for i in range(len(self.target_handles)):
            link = self.target_links[i]
            mesh = self.target_handles_mesh[i]
            grasp_poses = self._compute_grasp_poses(mesh, link.pose)
            self.target_handles_grasp_poses.append(grasp_poses)

    def compute_dense_reward(self, *args, info: dict, **kwargs):
        reward = 0.0

        # ------------------------------------------------------ #
        # The end-effector should be close to the target pose
        # ------------------------------------------------------ #
        handle_pose = self.target_link.pose
        ee_pose = self.agent.hand.pose

        # Position
        ee_coords = self.agent.get_ee_coords_sample()  # [2, 10, 3]
        handle_pcd = transform_points(
            handle_pose.to_transformation_matrix(), self.target_handle_pcd
        )
        # trimesh.PointCloud(handle_pcd).show()
        disp_ee_to_handle = sdist.cdist(ee_coords.reshape(-1, 3), handle_pcd)
        dist_ee_to_handle = disp_ee_to_handle.reshape(2, -1).min(-1)  # [2]
        reward_ee_to_handle = -dist_ee_to_handle.mean() * 2
        reward += reward_ee_to_handle

        # Encourage grasping the handle
        ee_center_at_world = ee_coords.mean(0)  # [10, 3]
        ee_center_at_handle = transform_points(
            handle_pose.inv().to_transformation_matrix(), ee_center_at_world
        )
        # self.ee_center_at_handle = ee_center_at_handle
        dist_ee_center_to_handle = self.target_handle_sdf.signed_distance(
            ee_center_at_handle
        )
        # print("SDF", dist_ee_center_to_handle)
        dist_ee_center_to_handle = dist_ee_center_to_handle.max()
        reward_ee_center_to_handle = (
            clip_and_normalize(dist_ee_center_to_handle, -0.01, 4e-3) - 1
        )
        reward += reward_ee_center_to_handle

```

```
        # pointer = trimesh.creation.icosphere(radius=0.02, color=(1, 0, 0))
        # trimesh.Scene([self.target_handle_mesh, trimesh.PointCloud(ee_center_at_handle)]).show()

        # Rotation
        target_grasp_poses = self.target_handles_grasp_poses[self.target_link_idx]
        target_grasp_poses = [handle_pose * x for x in target_grasp_poses]
        angles_ee_to_grasp_poses = [
            angle_distance(ee_pose, x) for x in target_grasp_poses
        ]
        ee_rot_reward = -min(angles_ee_to_grasp_poses) / np.pi * 3
        reward += ee_rot_reward

        # -------------------------------------------------- #
        # Stage reward
        # -------------------------------------------------- #
        coeff_qvel = 1.5  # joint velocity
        coeff_qpos = 0.5  # joint position distance
        stage_reward = -5 - (coeff_qvel + coeff_qpos)
        # Legacy version also abstract coeff_qvel + coeff_qpos.

        link_qpos = info["link_qpos"]
        link_qvel = self.link_qvel
        link_vel_norm = info["link_vel_norm"]
        link_ang_vel_norm = info["link_ang_vel_norm"]

        ee_close_to_handle = (
            dist_ee_to_handle.max() <= 0.01 and dist_ee_center_to_handle > 0
        )
        if ee_close_to_handle:
            stage_reward += 0.5

            # Distance between current and target joint positions
            # TODO(jigu): the lower bound 0 is problematic? should we use lower bound of joint limits?
            reward_qpos = (
                clip_and_normalize(link_qpos, 0, self.target_qpos) * coeff_qpos
            )
            reward += reward_qpos

            if not info["open_enough"]:
                # Encourage positive joint velocity to increase joint position
                reward_qvel = clip_and_normalize(link_qvel, -0.1, 0.5) * coeff_qvel
                reward += reward_qvel
            else:
                # Add coeff_qvel for smooth transition of stagess
                stage_reward += 2 + coeff_qvel
                reward_static = -(link_vel_norm + link_ang_vel_norm * 0.5)
                reward += reward_static

                # Legacy version uses static from info, which is incompatible with MPC.
                # if info["cabinet_static"]:
                if link_vel_norm <= 0.1 and link_ang_vel_norm <= 1:
                    stage_reward += 1

        # Update info
```

```
        info.update(ee_close_to_handle=ee_close_to_handle, stage_reward=stage_reward)

        reward += stage_reward
        return reward
```

Listing 1: Human-engineered rewards for Open Cabinet Door tasks. The code is from the ManiSkill2 github repo (commit id: `493be36`).

```
def compute_stage_indicators(self, info: dict):
    stage_indicators = [
        self.agent.check_grasp(self.target_link), # this utility function is given by the original ManiSkill2 codebase. It requires one hyperparameter 'max_angle' but we just use the default value
        self.link_qpos >= self.target_qpos,
        info['success'], # we use the task success singal as the 3rd stage indicator, and it requires the door to be stationary
    ]
    return stage_indicators
```

Listing 2: Our stage indiactors for Open Cabinet Door tasks which is way more easier to design than the human-engineered rewards.

C Comparison with Text2Reward

Text2Reward (Xie et al., 2023) is a concurrent work with our paper. We offer a comparison in this section to help readers understand the differences between our paper and Text2Reward.

While both (Xie et al., 2023) and our paper share the common goal of generating rewards for new tasks, they employ fundamentally distinct setups and methodologies. In short, the primary distinction lies in the fact that **our approach learns rewards from training tasks and success signals (or stage indicators), while (Xie et al., 2023) generates rewards based on exemplar reward codes and the knowledge embedded in Large Language Models (LLMs)**.

To elaborate, the following disparities exist in respective setups and assumptions:

- Both (Xie et al., 2023) and our methods need to interact with environments. However, we emphasize more on evaluating the learned rewards on unseen test tasks.
- (Xie et al., 2023) assumes access to a pool of instruction-reward code pairs, while our method requires training on relevant training tasks instead.
- (Xie et al., 2023) assumes access to the source code of the tasks, allowing them to provide LLMs with a Pythonic environment abstraction and various utility functions. In contrast, our method solely relies on success signals (or stage indicators) and does not require the code of the tasks.

D Implementation Details

D.1 Reward Learning Phase

D.1.1 Network Architectures

- Actor Network: 4-layer MLP, hidden units (256, 256, 256)
- Critic Networks: 4-layer MLP, hidden units (256, 256, 256)
- Discriminator Networks (Reward): 2-layer MLP, hidden units (32)

D.1.2 Hyperparameters

We use SAC (Haarnoja et al., 2018) as the backbone RL algorithm in the reward learning phase of DrS. The related hyperparameters are listed in Table 1.

D.2 Reward Reuse Phase

D.2.1 Network Architectures

- Actor Network: 4-layer MLP, hidden units (256, 256, 256)
- Critic Networks: 4-layer MLP, hidden units (256, 256, 256)

D.2.2 Hyperparameters

During the reward reuse phase, we use different rewards to train agents by SAC (Haarnoja et al., 2018). The related hyperparameters are listed in Table 2.

Name	Value
max iteration	6M
replay buffer size	6M
update-to-data (UTD) ratio	0.5
optimizer	Adam
actor learning rate	3e-4
critic learning rate	3e-4
discriminator learning rate	3e-4
gradient clip	10.0
target smoothing coefficient	0.005
discount factor	0.8
training frequency	64 steps
target network update frequency	1 step
discriminator update frequency	1 step
batch size	1024
entropy regularization coefficient	0.2

Table 1: The hyperparameters used in reward learning phase of DrS.

Name	Value
max iteration	6M
replay buffer size	6M
update-to-data (UTD) ratio	0.5
optimizer	Adam
actor learning rate	3e-4
critic learning rate	3e-4
gradient clip	10.0
target smoothing coefficient	0.005
discount factor	0.8
training frequency	64 steps
target network update frequency	1 step
batch size	1024
entropy regularization coefficient	0.2

Table 2: The hyperparameters used in the reward reuse phase of DrS.

E Additional Ablation Study

In this section, we present more ablation studies that are not included in the main paper due to the space limit. These experiments are conducted on the Pick-and-Place task family.

E.1 Modality of the Inputs to the Rewards

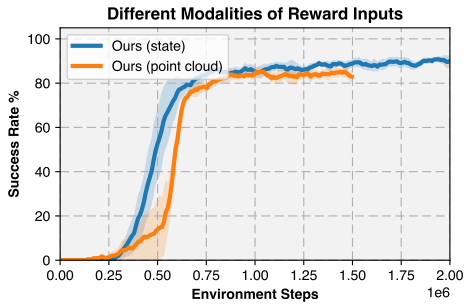


Figure 9: An experiment about using our approach with point cloud inputs, and the point clouds are processed by a PointNet.

Our approach is able to accommodate various input modalities for the reward functions, including both low-dimensional state vectors and high-dimensional visual inputs. To demonstrate this compatibility,

we conducted an additional experiment using point cloud inputs. In this experiment, the reward function (discriminator) not only considers the low-dimensional state but also takes a point cloud as input, with the point cloud being processed by a PointNet. The results of this experiment are depicted in Fig. 9.

We can see that the reward with point cloud input performs similarly to the one with state input, which shows that our approach is perfectly compatible with high-dimensional visual inputs. However, the techniques about visual inputs are a bit orthogonal to our focus (reward learning), and learning with visual inputs takes significantly more time, so we still keep most of our experiments on state inputs.

The results reveal that the reward function utilizing point cloud input performs comparably to the one utilizing state input, demonstrating the seamless integration of our approach with high-dimensional visual inputs. However, it is worth noting that the techniques about visual inputs, while compatible with our framework, are a little bit orthogonal to our focus (reward learning). Moreover, learning with visual inputs typically takes a significantly longer training time. Consequently, the majority of our experiments primarily use state inputs, allowing us to concentrate on the core aspects of reward learning.

E.2 Discriminator Modification and Stage Indicators

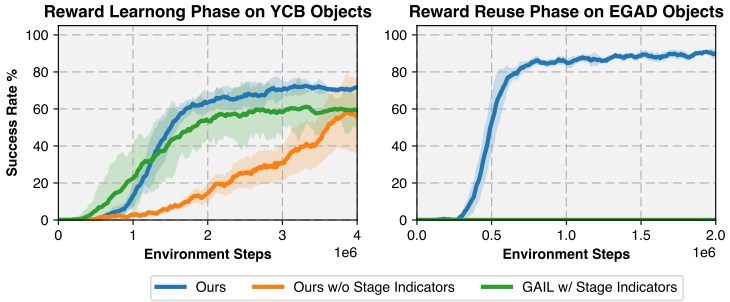


Figure 10: An ablation study was conducted to examine the impact of discriminator modification and stage indicators. Both the reward learning phase and reward reuse phase are shown. The learned rewards from the ablated baselines failed to successfully train new agents in the test tasks.

In contrast to GAIL (Ho & Ermon, 2016), our approach incorporates two critical modifications in the training of discriminators to facilitate the learning of reusable dense rewards. These modifications entail: (a) replacing the agent-demonstration discriminator with the success-failure discriminator, and (b) employing stage indicators by utilizing a separate discriminator for each stage. To ascertain the significance of these modifications, we devised two ablation baselines:

- **GAIL w/ Stage Indicators**: This baseline serves as an equivalent representation of our method without the incorporation of the success-failure discriminator. In GAIL, the discriminator solely distinguishes between agent and expert trajectories, making it incapable of learning separate rewards for each stage. To incorporate the stage indicators within the GAIL framework, we first train the original GAIL on the training tasks. During the reward reuse phase, we linearly combine the GAIL reward with the semi-sparse reward, thus leveraging the stage information. Through experimentation, we explored different weightings to strike an optimal balance between these two reward components.
- **Ours w/o Stage Indicators**: In this baseline, we exclude the stage indicators and solely rely on the task completion signal to train the discriminator. This approach is equivalent to the one-stage reward learning discussed in Sec. 4.1.

Fig. 10 illustrates the comparison between the two ablation baselines and our method during both the reward learning phase and reward reuse phase. While both "GAIL w/ Stage Indicators" and "Ours w/o Stage Indicators" demonstrate similar success rates as our method at the conclusion of the reward learning phase, it is crucial to emphasize that the learned rewards from both ablation baselines **fail** to

be reused to the test tasks. In contrast, our method achieves the acquisition of high-quality reward functions capable of effectively training new RL agents in the test tasks. This outcome substantiates the indispensability of the two proposed components in facilitating the acquisition of reusable dense rewards.

E.3 Reward Formulation

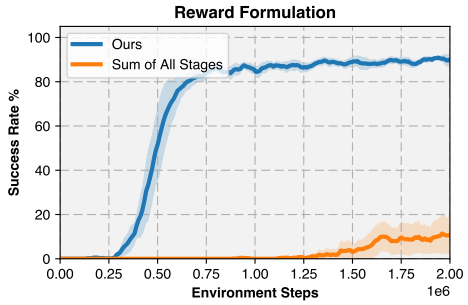


Figure 11: Ablation study of reward formulation. Comparison is done by reusing the learned rewards to train new agents on the test tasks.

In our approach, we leverage the stage indicators and define the reward function as the sum of the semi-sparse reward and the discriminator's bounded prediction for the current stage, as expressed in Eq. 4. This formulation ensures that the reward strictly increases across stages. To evaluate the effectiveness of this formulation, we compare it with a straightforward variant, denoted as $\sum_k \tanh(\text{Discriminator}_k(s'))$, which sums up the discriminator predictions for all stages. As depicted in Fig. 11, the simple variant exhibits significantly poorer performance, underscoring the importance of focusing on the dense reward specific to the current stage.

F Automatically Generating Stage Indicators

This section discusses a few promising solutions to automatically generate stage indicators, drawing inspiration from some recent publications. Though this topic is *a little bit beyond the scope of our paper*, we believe this is a valuable discussion for the readers.

F.1 Employ LLMs for Code Generation of Stage Indicators

Beyond task decomposition, LLMs demonstrate the capability to directly write code (Liang et al., 2023; Singh et al., 2023; Yu et al., 2023; Ha et al., 2023) for robotic tasks. A recent study (Ha et al., 2023) exemplified how LLMs, when prompted with the appropriate APIs, can generate success conditions (code snippets) for each subtask. Given the swift advancements in the domain of large models, it is entirely feasible to generate both stage structures and stage indicators using them.

F.2 Infer Stages via Keyframe Discovery

The boundaries between stages can be viewed as keyframes in the trajectories. A recent approach introduced by (Shi et al., 2023) suggests the automated extraction of such keyframes from trajectories, leveraging reconstruction errors. Given these keyframes, one intuitive solution is to develop a keyframe classifier that can act as a stage indicator. However, this requires a certain degree of consistency across keyframes, and we believe it is an interesting direction to explore.

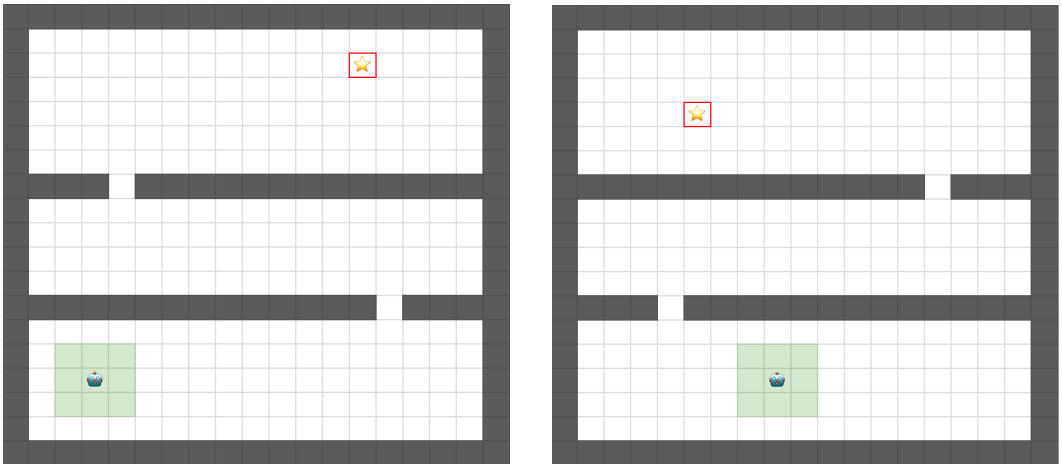
G Additional Experiments on Other Domains

G.1 Navigation

G.1.1 Introduction

In this section, we in incorporated experiments on navigation tasks, which were conducted during the initial stages of our project. We do not include these results in the main paper, as we found these simple navigation tasks to be less interesting compared to the robot manipulation tasks.

G.1.2 Setup



(a) Training Task (b) Test Task

Figure 12: Visualization of the training and test tasks in the navigation domain. The agent begins at a random location in the bottom room, and the goal is randomly positioned in the top room.

Task Description We have developed a 2D navigation task conceptually similar to MiniGrid, as visually represented in Fig. 12. The maps are 17x17, where the agent is randomly placed in the bottom room and needs to navigate to the star, randomly located in the top room.

Observation Observations provided to the agent include its xy coordinates, the xy coordinates of the goal, and a 3x3 patch around itself.

Action The agent has a choice of 5 actions: moving up, down, left, right, or remaining stationary.

Training and Test Set The reward is learned on the map shown in 12a and then reused on the map in 12b. The difference between these two maps lies in the positions of two gates.

G.1.3 Results

Our method is also effective in learning reusable rewards for navigation tasks. Given the relative simplicity of this specific navigation task, our approach's one-stage version suffices, eliminating the need for additional stage information. The results for this experiment are shown in Fig. 13.

The results clearly demonstrate that the learned reward from our approach successfully guides the RL agent to complete the task perfectly. In contrast, RL agents with sparse rewards show poor performance. Note that the map used in the test task differs from the training one, so directly transferring policy would not work. We also visualize the learned reward in Fig. 14. See the caption for a detailed analysis.

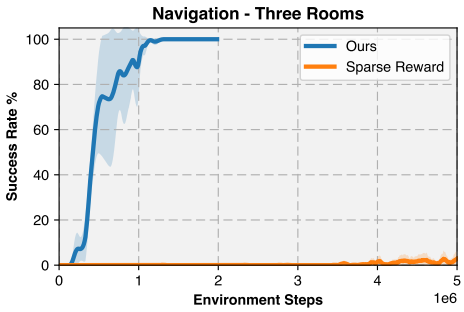


Figure 13: Evaluation results of reusing learned rewards in the navigation task. All curves use DQN to train, but with different rewards. 3 random seeds, the shaded region is std.

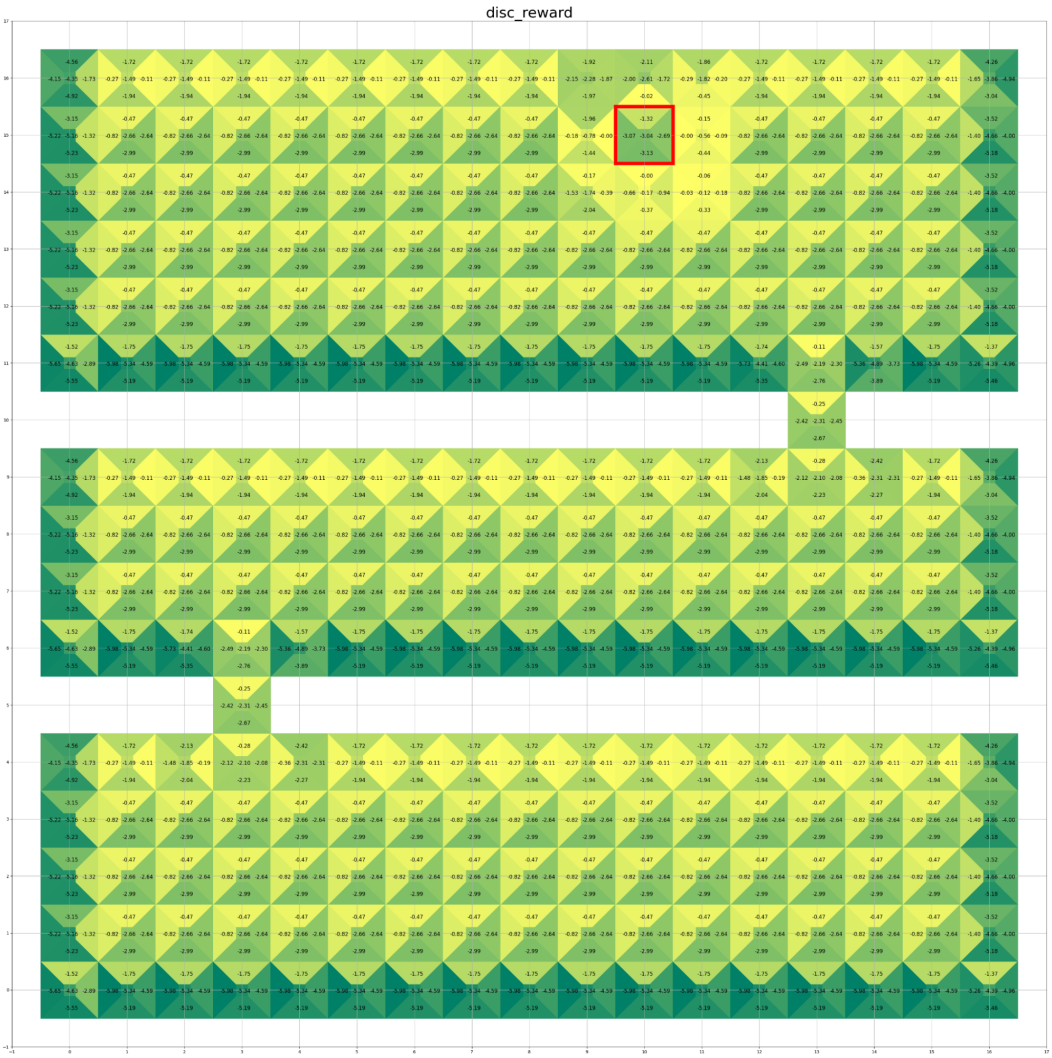


Figure 14: Visualization of Learned Reward: Each cell displays five values corresponding to the rewards for five different actions. A lighter color means a higher reward value. The red box shows the location of a randomly chosen goal. Note that the inputs to both the learned reward function and the agent include only the local 3x3 area around the agent, excluding any information about the gate positions. Overall, the learned reward encourages upward movement, aligning with the placement of the goal in the top room. When the gates are not within the agent's local 3x3 patch, the rewards for moving left or right are nearly equivalent, which is reasonable since the gate's position cannot be determined without direct observation. However, when the gates are visible within the local patch, the learned reward directs the agent to go through these gates. This behavior of the learned reward aligns well with the task's objectives.

G.2 LOCOMOTION

G.2.1 INTRODUCTION

While it can be tricky to divide locomotion tasks into stages, our method (specifically, the one-stage version) is capable of effectively handling such tasks, if they have a short horizon. In this section, we demonstrate that our approach can learn reusable rewards for Half Cheetah, a representative locomotion task in MuJoCo.

For tasks that are long-horizon and hard to specify stages, such as the Ant Maze, crafting rewards is very challenging even for experienced human experts. Therefore, we leave these tasks for future work.

G.2.2 SETUP

Task Description Our experiment uses `HalfCheetah-v3` from Gymnasium. The `HalfCheetah` task has a predefined reward threshold of 4800, as specified in their code, which is used to gauge task completion according to their documentation. Thus, we define the sparse reward (success signal) for this task as achieving an accumulative dense reward greater than 4800.

Training and Test Set In the reward learning phase, we use the standard `HalfCheetah-v3` task. In the reward reuse phase, we modify the task by increasing the damping of the front leg joints (thigh, shin, and foot joints) by 1.5 times. This increased damping makes it more challenging for the cheetah to achieve high speeds.

G.2.3 RESULTS

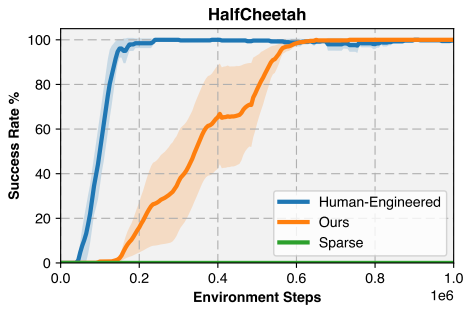


Figure 15: Evaluation results of reusing learned rewards in the `HalfCheetah-v3` task. All curves use SAC to train, but with different rewards. 3 random seeds, the shaded region is std.

Our method has successfully demonstrated its capability to learn reusable rewards in the Half Cheetah task. The results are illustrated in Fig. 15. Notably, the performance achieved using the learned reward is comparable to that of the human-engineered reward, while the sparse reward proved ineffective in training an RL agent. Given that many locomotion tasks emphasize low-level control and are typically of a shorter horizon, our approach's one-stage version proves to be highly effective. Additionally, this version does not require any stage information, further underscoring its efficiency and adaptability in handling such tasks.

H Discussion on the Desired Properties of Dense Rewards

H.1 Overview

Our paper primarily focuses on learning a dense reward, so one important question we want to discuss is: *What kind of dense reward do we aspire to learn?*

It is somewhat challenging to strictly distinguish dense rewards from sparse rewards, due to the lack of strict definitions of dense rewards in the existing literature (to the best of our knowledge). However, this does not preclude a meaningful discussion about the desired properties of dense rewards. Unlike sparse rewards, which typically only provide reward signals when the task is solved, dense rewards offer more frequent and immediate feedback regarding the agent's actions.

We posit that the fundamental property of an *effective* dense reward is its capacity to enhance the sample efficiency of RL algorithms. The rationale behind this property is straightforward: a well-structured dense reward should **reduce the need for extensive exploration during RL training**. By providing direct guidance and immediate feedback, the agent can quickly discover optimal actions, thereby accelerating the learning process.

In line with this philosophy, an **ideal dense reward should allow the derivation of optimal policies with minimal effort**. By analyzing a simple tabular case, we find that **our learned reward exhibits this great property**. To be more specific, in the example below, **we can obtain the optimal policy by greedily following the path of maximum reward at each step**.

H.2 Analysis on a Simple Tabular Case

Under certain assumptions, we can obtain the optimal policy by greedily following the path of maximum reward at each step, i.e.,

$$\pi^*(s) = \arg\max_a R^\dagger(s, a)$$

, where π^* is the optimal policy and $R^\dagger$ is the learned reward.

H.2.1 Setup and Assumptions

In this analysis, we consider a MDP with the following assumptions:

- Deterministic transitions: $s' = P(s, a)$
- Discrete and finite state/action space: $S = \{s_0, s_1, ...\}$, $A = \{a_0, a_1, ...\}$
- Given sparse reward: $R(s, a, s') = 1$ if $s' = s_{goal}$, otherwise 0
- Discount factor: $\gamma < 1$

Other assumptions about our approach:

- Only one stage, so the one-stage version of our approach is applied.
- The buffers for success trajectories and failure trajectories are large enough, but not infinite.
- After training for a sufficiently long time, policy converges to the optimal policy π^*. (This is a strong assumption, but it is possible in theory.)

H.2.2 Notations

- Learned reward: $R^\dagger(s, a) = \tanh(\text{Discriminator}(s, a))$, so $R^\dagger(s, a) \in (-1, 1)$
- Buffer for success trajectories $\mathcal{B}^+$, buffer for failure trajectories $\mathcal{B}^-$
- Optimal policy: $\pi^*(a|s)$, which represents the probability of choosing action a at state s. Here we overload the notation π^* to capture the potential multi-modal output of the policy.

H.2.3 CONNECTION BETWEEN OPTIMAL POLICY AND LEARNED REWARD

Here, we want to demonstrate that *the learned reward of an optimal action is always higher than that of any non-optimal action in each state*. If this holds, it then becomes feasible to straightforwardly identify the optimal action at each state by adopting a greedy strategy that selects the action yielding the highest reward.

When $\gamma < 1$, π^* will go to s_{goal} by the shortest paths, so $\pi^*(a|s) = 1/k_s$ or 0, where k_s is the number of optimal actions at s.

$\forall s$, there are two kinds of actions a^+ and a^-

1. $\pi^*(a^+|s) > 0$, which means a^+ is one of the optimal actions. Then (s, a^+) must be in $\mathcal{B}^+$, possibly be in $\mathcal{B}^-$. Therefore, $R^\dagger(s, a^+) > -1 + \epsilon$, when the discriminator converges. This is because the buffers are finite-size, (s, a^+) will be sampled into positive training data of the discriminator with a probability larger than 0.
2. $\pi^*(a^-|s) = 0$, which means a^- is NOT one of the optimal actions. Then (s, a^-) will only be in $\mathcal{B}^-$, and will NOT be in $\mathcal{B}^+$ Therefore, $R^\dagger(s, a^-) \to -1$, when the discriminator converges. This is because (s, a^-) will only show in the negative training data of the discriminator.

Therefore, we have $R^\dagger(s, a^+) > R^\dagger(s, a^-)$ for all states s. By employing a greedy strategy that selects $\arg\max_a R^\dagger(s, a)$, we can reach the goal states in the same way as how the optimal policy π^* reaches the goal.

H.3 FURTHER DISCUSSIONS

This subsection is dedicated to addressing additional questions the readers may raise after reading the above analysis.

H.3.1 DOES THE ABOVE CONCLUSION GENERALIZE TO MORE COMPLICATED CASES?

Although our analysis highlights a desirable property of the learned reward in a simple tabular case, this finding should not be hastily generalized to more complex cases, such as the robotic manipulation tasks used in our paper. This caution is due to two primary reasons:

1. In environments where the state and action spaces are continuous, the ability of the neural network to interpolate plays a significant role in shaping the final learned reward.
2. Practically, achieving convergence for both the policy and the discriminator can be a very time-consuming process.

H.3.2 THE NECESSITY OF LEARNED REWARD DESPITE ITS SIMILARITY TO POLICY

The learned reward might appear redundant at first glance, as it seems to convey the same information as the learned policy. This observation raises a potential question: why is there a need for a learned reward if we already have a learned policy? Couldn't we just utilize the learned policy directly?

The answer lies in the distinct advantages that the learned reward offers, particularly when adapting to new tasks. When the environment dynamics change, a new policy can be effectively retrained using the learned reward in conjunction with the new environmental dynamics. Directly transferring the policy, or fine-tuning it with a sparse reward, can be less efficient in certain situations. For a practical illustration of this concept, refer to Fig. 14 and Sec, G.1.3. These sections provide a compelling example where the transfer of rewards demonstrates success, in contrast to the less effective transfer of policies.