LIFELONG ROBOTIC REINFORCEMENT LEARNING BY RETAINING EXPERIENCES

Annie Xie and Chelsea Finn
Stanford University
anniexie@stanford.edu

ABSTRACT

Multi-task learning ideally allows embodied agents such as robots to acquire a diverse repertoire of useful skills. However, many multi-task reinforcement learning efforts assume the agent can collect data from *all* tasks at *all* times, which can be unrealistic for physical agents that can only attend to one task at a time. Motivated by the practical constraints of physical learning systems, this work studies lifelong learning as a more natural multi-task learning setup. We present an approach that effectively leverages data collected from previous tasks to cumulatively and efficiently grow the robot's skill-set. In a series of simulated robotic manipulation experiments, our approach requires less than half the samples than learning each task from scratch, while avoiding the impractical round-robin data collection scheme. On a Franka Emika Panda robot arm, our approach incrementally solves ten challenging tasks, including bottle capping and block insertion.

1 INTRODUCTION

General-purpose embodied agents ideally should be capable of learning and performing a multitude of tasks. Multi-task learning paves a promising path towards such agents by capitalizing on structure shared between tasks to learn them more efficiently (Rusu et al., 2016a; Parisotto et al., 2016; Teh et al., 2017; Hausman et al., 2018; Riedmiller et al., 2018; Yu et al., 2020). However, the standard multi-task reinforcement learning paradigm, which requires data collection from each task in round-robin fashion, can often be unrealistic for embodied agents such as physical robots, which can only attend to one physical workspace at a time. Beyond the impracticality of the framework, it also limits the flexibility in the way tasks are assigned. A robot may encounter new tasks that it needs to learn over the course of its lifetime. For example, the demands of factory robots can vary based on the product being built; the capabilities of service robots may need to grow to accommodate a larger client base. Instead, a more natural multi-task learning setup for physical agents is that of lifelong learning, wherein tasks are learned *in sequence*.

Two core challenges of lifelong learning are to enable forward transfer, i.e. reusing knowledge from previous tasks to more efficiently learn new tasks, while mitigating negative backward transfer or catastrophic forgetting (McCloskey & Cohen, 1989; Kirkpatrick et al., 2017). Prior works have shown that the latter can be addressed by maintaining and replaying prior experiences (Isele & Cosgun, 2018; Rolnick et al., 2019; Chaudhry et al., 2019; Buzzega et al., 2020; Balaji et al., 2020). Since real robotic systems are more often bottlenecked by the the time it takes to collect data than by hard drive memory, we adopt this replay approach, assuming complete memory of past experiences, and focus on former challenge of enabling forward transfer across a sequence of tasks.

Prior work that address the forward transfer aspect of the lifelong learning problem primarily focus on the transfer of weights (Caruana, 1997; Fernández & Veloso, 2006; Rusu et al., 2016b; Schwarz et al., 2018; Julian et al., 2020), e.g.,

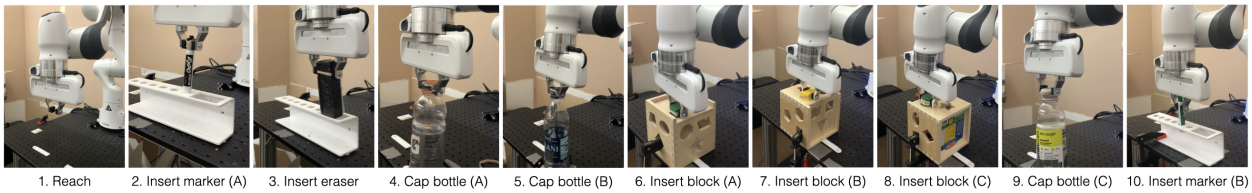


Figure 1: A Franka Emika robot arm learns a sequence of manipulation tasks, including block insertion and bottle capping, by retaining experience from previous tasks. Our algorithm can learn a sequence of tasks with different setups and objectives with fewer samples compared to when each task is learned from scratch.

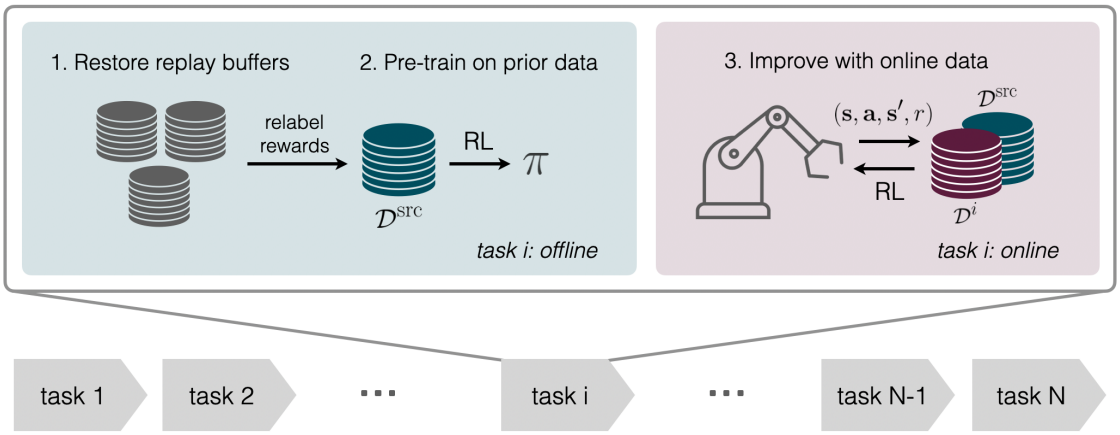


Figure 2: In our framework, we perform two steps during each new task. First, we pre-train on the experience from earlier tasks (left). To align the data to the same objective, we use the underlying reward function of the upcoming task to relabel this experience before pre-training. Second, we learn online in the robot's physical environment and gather new data to continuously improve the robot's policy until the task is solved (right).

by fine-tuning the previously trained neural network weights that represent the value function or policy. However, in retaining only the neural network weights, these methods may discard important information from the past experiences that are relevant to future tasks. Thus, our work aims to additionally transfer the previously collected low-level experience, in the form of $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ tuples saved in the replay buffer. Rather than leveraging the past experiences with standard experience replay, we propose to prioritize samples based on their utility for new tasks.

Since each task can significantly differ in their transition dynamics and reward functions, naively training on the previously collected samples suffers from sample selection bias (Cortes et al., 2008). Hence, when drawing upon samples from previous tasks, we should correct this bias by selectively identifying the most relevant samples for the new task. We measure the similarity between the past samples and the current task's transition dynamics to determine which samples to transfer in the online fine-tuning phase. Consequently, previously-collected samples that are relevant to the target task are used to simulate sampling in the real environment, both accelerating and stabilizing fine-tuning. This method of experience transfer, as we show in our experiments, complements and can be combined with other mediums of transfer such as of the previously trained weights of policy networks.

The core contribution of this work is a framework for efficient lifelong reinforcement learning that is suitable for physical robots (depicted in Fig. 2). In particular, our algorithm performs two distinct stages at the arrival of each new task: (1) it pre-trains a policy on the prior experience stored in its replay buffer, and (2) it improves the policy with *selective* data replay. Notably, both stages leverage previously collected data, and we find in an ablation study that each component serves an important role in our framework. Our simulated experiments consider two lifelong RL problems: a sequence of key-insertion tasks and a sequence of valve-turning tasks. On both problems, our approach considerably improves upon existing methods for sequential transfer, including Progressive Nets (Rusu et al., 2016b) and DARC (Eysenbach et al., 2021). Finally, we evaluate the efficacy of our approach on a sequence of ten challenging tasks with varying objectives and physical setups with a Franka Emika robot arm (see Fig. 1). It requires only 100K time-steps to learn ten tasks and achieves a 2x improvement over learning each task from scratch.

2 Related Work

Reinforcement learning has allowed robotic agents to autonomously learn an impressive array of skills (Kober et al., 2013; Levine et al., 2016), from locomotion (Kohl & Stone, 2004; Tedrake, 2004; Haarnoja et al., 2018a; Lee et al., 2020) to the manipulation of diverse objects (Gu et al., 2017; Kalashnikov et al., 2018; Lee et al., 2019; Andrychowicz et al., 2020). However, robotic RL setups are often only designed with the goal of solving an individual task in mind. As a result, the lifetime of these learning agents begins and ends with a single task, and each new task is to be learned from scratch. With the risks associated with physical interactions in the real world, this is not a practical approach for a robot to learn a diverse set of skills. Therefore, in the design of our algorithm, we emphasize data efficiency when solving a series of tasks.

In principle, if the agent leverages knowledge accrued from previously-solved tasks, then it should learn new ones more efficiently than from scratch (Taylor & Stone, 2009; Lazaric, 2012; Wang et al., 2017; Finn et al., 2017; Nagabandi et al., 2019; Zhao et al., 2020; Arndt et al., 2020; Song et al., 2020). In RL agents, this knowledge can be transferred through representations (Rusu et al., 2016b; Devin et al., 2017), learned models (Finn & Levine, 2017), network

weights (Fernández & Veloso, 2006; Rusu et al., 2016a), or experiences (Kaelbling, 1993; Taylor et al., 2008; Lazaric et al., 2008; Andrychowicz et al., 2017; Tirinzoni et al., 2018; Tao et al., 2020). Our work focuses on the transfer of weights and experiences, which we view as general and complementary forms of knowledge. Multi-task learning also aims to transfer knowledge across tasks but achieves this by learning the set of tasks together (Parisotto et al., 2016; Teh et al., 2017; Hausman et al., 2018; Yang et al., 2020; Yu et al., 2020). This framework has allowed robots to learn a range of goal-based tasks (Kaelbling, 1993; Andrychowicz et al., 2017) and other tasks that only differ in their reward functions, but may be less practical when each task has a unique physical setup. Unlike these prior works, we study the *sequential* transfer learning problem where data can only be collected from the current task rather than in a round-robin fashion, and study how to efficiently solve a sequence of tasks on a real robot manipulator.

A common form of sequential transfer is to reuse the weights of a policy network by fine-tuning them to the new task (Fernández & Veloso, 2006; Julian et al., 2020) or distilling the learned behavior (Levine et al., 2016; Rusu et al., 2016a; Parisotto et al., 2016; Schmitt et al., 2018). While learned policies and value functions readily offer prior task information in a compact form, they become less useful as the optimal policies between tasks are less similar. On the other hand, the raw experience accumulated from earlier tasks cannot be immediately used to generate behavior—we have to expend computational resources to optimize a policy with this data for example. Additionally, not all experiences may be relevant to the target task, which can be addressed by prioritizing samples by their relevance (Taylor et al., 2008; Lazaric et al., 2008; Tirinzoni et al., 2018; Tao et al., 2020; Eysenbach et al., 2021). Despite these challenges, individual experience samples also represent the most complete as well as unprocessed form of knowledge from a task, and hence can be used in flexible ways. For example, they can be used to optimize auxiliary objectives (Isele & Cosgun, 2018), combat catastrophic forgetting (Rolnick et al., 2019), and accelerate learning of new tasks (Andrychowicz et al., 2017; Yin & Pan, 2017; Tao et al., 2020). Nonetheless, none of these prior methods study lifelong learning of a sequence of physical robotic manipulation tasks. Our approach combines the strengths of weight and experience transfer to improve the data efficiency of sequential robotic task learning, and significantly outperforms prior methods in our experiments.

In many continual learning setups, the algorithm has a fixed storage budget, and must select existing samples to discard from its replay buffer. Prior work has proposed selection strategies such as reservoir sampling (Rolnick et al., 2019; Chaudhry et al., 2019; Buzzega et al., 2020; Balaji et al., 2020), which uniformly samples a datapoint to discard, and maximizing measures like surprise (Isele & Cosgun, 2018; Sun et al., 2021), diversity (Aljundi et al., 2019; Bang et al., 2021), or coverage of the state space (Isele & Cosgun, 2018). Our work sidesteps this problem by assuming the ability to retain all previous samples, since many robot learning settings are bottlenecked more by the speed of physical data collection than by hard disk size. With modern compute and hard drives, it is often practical to accumulate a significant amount of data and indeed modern machine learning systems have been most successful when trained on broad datasets that are much larger than those typically used for training robots (Krizhevsky et al., 2012).

3 Preliminaries

A task is defined by a Markov decision process with a continuous state space $\mathcal{S}$ and action space $\mathcal{A}$. The next state $\mathbf{s}' \in \mathcal{S}$ is determined by (unknown) dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and at each time step, the environment returns a scalar reward $r(\mathbf{s}, \mathbf{a})$. The goal of standard RL is to acquire a policy $\pi(\mathbf{a}|\mathbf{s})$ that maximizes the expected sum of rewards $\mathcal{J}(\pi) \coloneqq \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$, where ρ_π is the trajectory distribution induced by π. Next, we introduce an RL algorithm that solves the single-task setting with off-policy experience (Sec. 3.1). To reuse experience from prior tasks, we measure the relevance of individual samples for a new task with importance weights (Sec. 3.2).

3.1 Soft Actor-Critic

The soft actor-critic (Haarnoja et al., 2018b) (SAC) algorithm optimizes the maximum-entropy RL objective (Ziebart et al., 2008; Toussaint, 2009), using off-policy data to learn in a sample-efficient manner. The algorithm stores a replay buffer $\mathcal{D}$ of all collected transitions and rewards, i.e., $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$. With this data, a Q-function (or critic) Q_θ is trained to minimize the Bellman error $\mathcal{L}_Q = \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}',r)\sim\mathcal{D}} \left[(Q_\theta(\mathbf{s}, \mathbf{a}) - (r + V(\mathbf{s}')))^2\right]$, where $V(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[Q_\theta(\mathbf{s}, \mathbf{a}) - \alpha \log \pi_\phi(\mathbf{a}|\mathbf{s})]$ and α is a temperature parameter. The policy (or actor) π_ϕ is trained to minimize the KL divergence

$$\mathcal{L}_\pi = \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}} \left[D_{\text{KL}} \left(\pi_\phi(\mathbf{a}|\mathbf{s}) \middle\| \frac{\exp(Q_\theta(\mathbf{s}, \mathbf{a}))}{Z_\theta(\mathbf{s})} \right) \right]$$

where Z_θ is the partition function that normalizes the distribution $\exp(Q_\theta(\mathbf{s}, \mathbf{a}))$. In the overall algorithm, the agent alternates between collecting and storing data, and updating the actor and critic with this stored data. We build upon SAC as it is an effective RL algorithm that allows us to leverage previously collected data.

3.2 IMPORTANCE WEIGHTING

Domain adaptation methods typically leverage importance weighting to correct the bias of samples from the source domain (Zadrozny, 2004; Baktashmotlagh et al., 2014). When the tasks have different dynamics but same reward, prior work has defined the importance weights for each transition sample as the likelihood ratio $w(\mathbf{s}, \mathbf{a}, \mathbf{s}') := p^{\text{tgt}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})/p^{\text{src}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, where p^{src} are the transition probabilities in the source task and p^{tgt} are those in the target task. These weights can be estimated with learned probabilistic models (Tirinzoni et al., 2018) or with classifiers in a likelihood-free manner (Bickel et al., 2007; Eysenbach et al., 2021). Eysenbach et al. (Eysenbach et al., 2021) use the estimated weights to relabel the rewards, i.e. $\tilde{r}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \log \hat{w}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ in their method DARC, so that transitions that are likely under the target domain are weighed higher and vice versa.

While our method will build upon DARC, DARC on its own is ill-suited for the lifelong learning setting for two reasons. First, this definition of the importance weight assumes that the state-action distributions are the same in the source and target domain datasets, i.e., $p^{\text{src}}(\mathbf{s}, \mathbf{a}) = p^{\text{tgt}}(\mathbf{s}, \mathbf{a})$. DARC uses the same policy in the two domains, hence this approximately holds. However, we aim to learn a separate policy for each new task, making $p^{\text{src}}(\mathbf{s}, \mathbf{a})/p^{\text{tgt}}(\mathbf{s}, \mathbf{a})$ non-neglible. Second, the DARC setting places stronger limitations on the agent's access to the target domain, training the policy on *all* of the source-domain data is imperative. However, our setting allows the agent to improve in the target domain collecting new data as necessary. Due to the inevitable estimation error in $\hat{w}$, we find that training on all the source domain data, even if re-weighted, can be counterproductive.

4 LIFELONG REINFORCEMENT LEARNING VIA EXPERIENCE TRANSFER

The goal of our framework is to learn a series of robotic tasks in a practical and data-efficient manner. To this end, we describe the sequential multi-task learning problem (Sec. 4.1), in which the agent must learn policies for each task in sequence. We then devise an algorithm that leverages the prior experiences collected by the robot to improve sample efficiency when learning each additional task.

4.1 SEQUENTIAL MULTI-TASK LEARNING PROBLEM

We are interested in solving a sequence of tasks $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^N$. Formally, each task $\mathcal{T}^i$ is defined by a MDP, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $p^i(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and reward function $r^i(\mathbf{s}, \mathbf{a})$. Our work focuses on transfer between tasks on the same robotic platform, and thus assumes the state-action space is shared between tasks. However, because each task potentially presents a different physical environment setup and objective, the dynamics and reward functions may differ. Each task has a success criterion that determines if the current policy has successfully solved the given task. This criterion is designed by the user and verified either manually by the user or automatically. Crucially, the robot is only given the next task when it has satisfied the success criterion of the current one or has spent the maximum number of attempts.

Comparison to continual reinforcement learning. Unlike some continual RL formulations and approaches (Rusu et al., 2016b; Schwarz et al., 2018), we assume in this setting that offline data from previous tasks can be stored and retrieved for future use (Isele & Cosgun, 2018; Rolnick et al., 2019; Yin & Pan, 2017; Tao et al., 2020). This work also focuses on maximizing the forward transfer performance and *not* tackling catastrophic forgetting, a different challenge of continual learning. Also, motivated by the physical cost of switching tasks, we allow the robot to only collect new data in the task that it is currently solving.

Access to reward functions. Lastly, the environment is typically unknown to the agent. However, in robotics applications, the reward is often specified by the user, either directly provided as a closed-form function of states and actions, or indirectly as demonstrations of the task or examples of successful states. In the latter cases, a reward function can be recovered by inverse RL (Ziebart et al., 2008; Wulfmeier et al., 2015; Finn et al., 2016; Fu et al., 2018a) or learning a classifier that distinguishes between successful and unsuccessful states (Kalashnikov et al., 2018; Fu et al., 2018b; Xie et al., 2018; Singh et al., 2019). Hence, we assume that the agent can access the full reward function for each task $r^i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and not just the immediate reward feedback. As we describe next, this access allows us to *relabel* old experience with the new reward function so they are consistent with the new task.

4.2 ALGORITHM OVERVIEW

Our algorithm, depicted in Fig. 2 and summarized in Alg. 1, begins by learning the first task from scratch with standard SAC. For each following new task, we restore the replay buffer(s) from the previous task(s) and pre-train on this data

Algorithm 1 Lifelong RL by Retaining Experiences

$\mathcal{D}^1, \theta^1, \phi^1 \leftarrow \text{SAC}(\mathcal{T}^1)$
for $i = 2, \ldots, N$ **do**
  $\mathcal{D}^{\text{src}}, \theta^i, \phi^i \leftarrow \text{PRETRAIN}(\mathcal{D}^{1:i-1}, r^i)$
  $\mathcal{D}^i, \theta^i, \phi^i \leftarrow \text{IMPROVE}(\mathcal{D}^{\text{src}}, \theta^i, \phi^i)$

Algorithm 2 PRETRAIN

Input: Replay buffers $\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^{i-1}$
Input: Reward function r^i
Optional: Task parameters θ^{i-1}, ϕ^{i-1}
Initialize parameters θ^i, ϕ^i
Aggregate and relabel buffers (Eqn. 1)
for each iteration **do**
  Sample batch from $\mathcal{D}^{\text{src}}$
  Soft actor-critic updates:
  $\diamond\ \theta^i \leftarrow \theta^i - \alpha \nabla_{\theta^i} \mathcal{L}_Q$
  $\diamond\ \phi^i \leftarrow \phi^i - \alpha \nabla_{\phi^i} \mathcal{L}_\pi$
Return: $\mathcal{D}^{\text{src}}, \theta^i, \phi^i$

Algorithm 3 IMPROVE

Input: Relabeled source buffer $\mathcal{D}^{\text{src}}$
Input: Pre-trained parameters θ^i, ϕ^i
Initialize replay buffer $\mathcal{D}^i$
Initialize classifier ψ
for each iteration **do**
  for each environment step **do**
    Sample action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$
    Step in environment $\mathbf{s}' \sim p^i(\mathbf{s}'|\mathbf{s}, \mathbf{a})$
    Update buffer $\mathcal{D}^i \leftarrow \mathcal{D}^i \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)\}$
  for each update step **do**
    Sample batch from $\mathcal{D}^{\text{src}} \cup \mathcal{D}^i$
    Soft actor-critic updates:
    $\diamond\ \theta^i \leftarrow \theta^i - \alpha \nabla_{\theta^i} \mathcal{L}_Q$
    $\diamond\ \phi^i \leftarrow \phi^i - \alpha \nabla_{\phi^i} \mathcal{L}_\pi$
    Classifier update:
    $\diamond\ \psi \leftarrow \psi - \alpha \nabla_\psi \mathcal{L}_\psi$
  Filter source buffer $\mathcal{D}^{\text{src}}$ (Eqn. 2)
Return: $\mathcal{D}^i$, θ^i, ϕ^i

(Sec. 4.3). We then improve the agent's policy on the task with online experience (Sec. 4.4). Note that both of these stages reuses the previously collected, offline data in order to maximize the sample efficiency of our algorithm.

4.3 PRE-TRAINING ON PRIOR EXPERIENCE

At the arrival of a new task $\mathcal{T}^i$, we can either (1) randomly initialize the actor and critic weights or (2) restore them from the previous task. In our evaluation of both variants (Sec. 5), we find that depending on the similarity of the tasks, transferring the weights can result in better or worse performance compared to random initialization. Irrespective of the choice of initialization, we next restore the replay buffers $\mathcal{D}^{1:i-1}$ from the previous tasks. Then, with the reward function r^i for task $\mathcal{T}^i$, we can relabel the rewards of the aggregated dataset, making them consistent with task $\mathcal{T}^i$

$$\mathcal{D}^{\text{src}} := \bigcup_{j=1}^{i-1} \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r^i(\mathbf{s}, \mathbf{a})) \mid (\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \in \mathcal{D}^j\}. \tag{1}$$

We subsequently sample batches of this modified dataset $\mathcal{D}^{\text{src}}$ with which we apply offline updates to the policy and critic as a form of pre-training. Despite the potential discrepancies in the dynamics between tasks, we expect the pre-trained parameters to produce better trajectories than a random policy or even the policy from the previous task, and thus accelerate learning in the new task. The pre-training subroutine is summarized in Alg. 2.

4.4 IMPROVING WITH ONLINE EXPERIENCE

To improve the pre-trained policy from Sec. 4.3, our algorithm next collects online interactions in the robot's physical learning environment for task $\mathcal{T}^i$. Since our goal is to minimize the amount of online experience the robot needs to collect in its environment, we aim to also use the relabeled experience $\mathcal{D}^{\text{src}}$ during this online phase. However, because of the differing dynamics between tasks, these samples may vary in utility for accomplishing the current task. Addressing the dynamics gap is significantly more challenging, because unlike the reward function which is commonly specified by the user, we rarely have access to a model of the environment. One approach is to learn to model the dynamics $p^i(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ and relabel the transitions by querying the model. However, increasingly complex dynamics demand more expressive models and thus more environment samples to fit them. It is often easier and more efficient to instead estimate the importance weights, i.e., the likelihood ratio of samples under the target task versus the source tasks.

Likelihood-free importance weights. Let $\mathcal{D}^{\text{tgt}}$ represent the samples from the target task i and $\mathcal{D}^{\text{src}}$ samples from all previous tasks. Our algorithm trains a separate policy for each new task, leading to unequal state-action distributions $p^{\text{tgt}}(\mathbf{s}, \mathbf{a}) \neq p^{\text{src}}(\mathbf{s}, \mathbf{a})$. Hence, we define the importance weights as $w(\mathbf{s}, \mathbf{a}, \mathbf{s}') = p^{\text{tgt}}(\mathbf{s}, \mathbf{a}, \mathbf{s}')/p^{\text{src}}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. In contrast to the importance weights in DARC (Eysenbach et al., 2021) and in off-policy RL, the w we define here account and correct for shifts in both the transition dynamics due to the different tasks, and the marginal state-action distribution due to the different policies. To estimate w for samples $(\mathbf{s}, \mathbf{a}, \mathbf{s}') \in \mathcal{D}^{\text{src}}$, we express the ratio with Bayes' rule as

$$w(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \frac{p(\text{target}|\mathbf{s}, \mathbf{a}, \mathbf{s}')}{p(\text{source}|\mathbf{s}, \mathbf{a}, \mathbf{s}')} \cdot \frac{p(\text{source})}{p(\text{target})}.$$

This expression is a more useful form since we can estimate the first term with a classifier $c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ that outputs the probability that a $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ tuple is from the target task, trained with the cross-entropy loss:

$$\mathcal{L}_\psi = -\mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}^i}\left[\log c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')\right] - \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}^{\text{src}}}\left[\log(1 - c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}'))\right].$$

The second term can be estimated with the ratio of the replay buffers' size, $|\mathcal{D}^{\text{src}}|/|\mathcal{D}^i|$. Intuitively, the $\mathcal{D}^{\text{src}}$ samples weigh less as we collect more samples in the target task.

Which samples should we transfer? One way to use these weights is to re-weigh the examples in the RL objective. However, the weights can be numerically unstable and may require clipping to lie in a more reasonable range. We instead filter out samples that are unlikely in the target task, according to the classifier c_ψ. Concretely, we apply a threshold to the first term of the importance weight:

$$\tilde{w}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \mathbb{1}\left(\frac{c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')}{1 - c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')} \geq \gamma\right)$$

Thresholding allows us to control how conservatively samples are transferred with γ. We incorporate the second term by sampling training batches according to the ratio $|\mathcal{D}^{\text{src}}|/|\mathcal{D}^i|$. As $|\mathcal{D}^{\text{src}}|$ remains fixed and $|\mathcal{D}^i|$ grows, we in turn upsample from the target task buffer $\mathcal{D}^i$ and downsample from the source buffer $\mathcal{D}^{\text{src}}$. Our overall filtering rule is:

$$\mathcal{D}^{\text{src}} \leftarrow \left\{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \;\middle|\; (\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \in \mathcal{D}^{\text{src}}, \frac{c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')}{1 - c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')} \geq \gamma\right\} \tag{2}$$

We train the classifier c_ψ online and re-filter the source data $\mathcal{D}^{\text{src}}$ at regular intervals. The complete online improvement phase is outlined in Alg. 3.

5 Experiments

Our experiments seek to answer the following questions: **(1)** How does our approach compare to existing methods for sequential transfer? **(2)** How important is the pre-training phase and how important is filtering prior data in the online fine-tuning phase? **(3)** Can our approach sequentially learn to solve a series of manipulation tasks on a robot? The videos accompanying our experiments are on our project webpage: https://sites.google.com/view/retaining-experience-anon/.

5.1 Experimental Setup

Comparisons. To answer question **(1)**, we carefully study our method as well as several comparisons in a simulated robot environment. In particular, we compare to two methods that leverage previously trained weights:

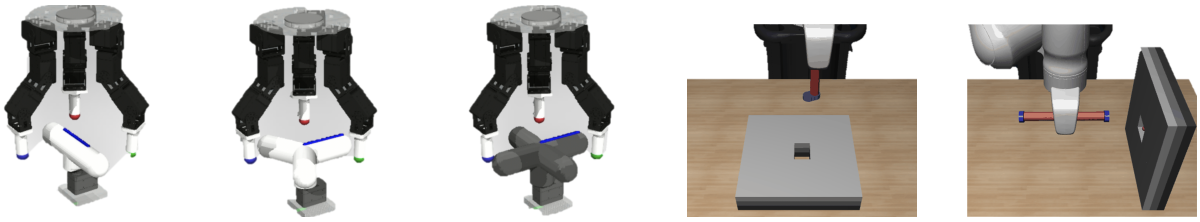
- **Progressive Nets** (Rusu et al., 2016b). Continual learning framework that transfers previously learned features.
- **Fine-tuning** (Julian et al., 2020). Ablation of our method that restores the weights from the previous task.

We also compare to methods that reuse previously collected data. When evaluating these methods, we also apply the reward relabeling scheme defined in Eqn. 1.

- **DARC** (Eysenbach et al., 2021). Domain adaptation method that defines the importance weights $w^{\text{DARC}}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = p^{\text{tgt}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})/p^{\text{src}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, which are estimated with two classifiers $c(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ and $c(\mathbf{s}, \mathbf{a})$, and adds $\Delta r = \log w^{\text{DARC}}$ to the rewards of prior data. DARC is designed to only learn from data collected in the source domain. To use both old and new data, we only relabel the experiences from previously solved tasks, and clip Δr from above at 0. We find these modifications improve the performance of DARC.
- **Off-policy IW**. Vanilla off-policy RL that reweighs samples with the importance weights $w^{\text{OP}} = \pi^i(\mathbf{a}|\mathbf{s})/\pi^{i-1}(\mathbf{a}|\mathbf{s})$.

Finally, we evaluate our approach and its ablations:

- **Ours**. The weights of this variant are randomly initialized before the pre-training phase of each task, and only the replay buffers are transferred.
- **Ours (warm-start)**. In addition to the replay buffers, we also restore the trained weights of the policy and critic from the previous task.
- **Ours (with DARC weights)**. An ablation of **Ours** that does not use the filtering scheme introduced in Section 4.4 and instead, like DARC, relabels the rewards as $r + \log w^{\text{DARC}}$.



(a) (b)

Figure 3: (a) D'Claw tasks with different valve shapes (image taken from (Yang et al., 2022)). (b) Examples of vertical and horizontal key-insertion tasks.

Simulated environments. We evaluate each method on two simulated robotic environments, depicted in Fig. 3. In each, the algorithm must learn to solve a sequence of tasks, with varying dynamics and reward functions.

- **ROBEL D'Claw** (Yang et al., 2022). This benchmark includes a set of 10 valve-rotation tasks, each with a different valve shape. The tasks also differ in the position feedback gain, friction coefficients, Cartesian position of the valve, target orientation of the valve, and desired direction of rotation, i.e., clockwise versus counterclockwise. We design *random* task sequences of 10 tasks each, which have randomly sampled parameters defining each task. For visualization, we additionally design a *hard* task sequence that alternates between rotating the valve clockwise and counterclockwise, in addition to randomizing the other task parameters.

- **Key Insertion**. We construct a family of key-insertion tasks within Robosuite (Zhu et al., 2020), using the MuJoCo physics engine (Todorov et al., 2012). The objective of each task is to insert the key into a box, while different tasks have varying placements of the box, key sizes, and initial orientations of the key with respect to the box. For more task heterogeneity, we rotate the box in a subset of the tasks, and the key has to be horizontally inserted, rather than vertically. We again design *random* task sequences of 16 tasks, each with randomly sampled task parameters. Similar to the D'Claw domain, we design a *hard* task sequence that alternates between vertical and horizontal insertion tasks, in addition to randomizing the other parameters.

A full description of each environment is provided in App. B.

Evaluation metrics. In our evaluation of each method, we track the **average performance** across the agent's entire lifetime and the **final performance** on each task of the sequence after T_{task} time-steps. In our simulations, the performance is measured in terms of task success, and the agent trains for $T_{\text{task}} = 50\text{k}$ steps in each task.

5.2 Evaluating Forward Transfer

Comparative evaluation. In Table 1, we summarize the performance of all methods in terms of task success. In the D'Claw domain, our method outperforms all comparisons in terms of average and final task success, matching the performance of learning each task from scratch with 100K time-steps. On average, the final performance of our method on each task is 82%. Critically, **Ours (warm-start)**, which additionally transfers the policy and critic network weights, is less successful, indicating that this initialization harms performance. We hypothesize that the transfer of weights is less useful due to the heterogeneity in the tasks, specifically since tasks can vary in which direction the valve needs to be rotated. Nonetheless, we see that **Ours (warm-start)** improves upon naive fine-tuning, hence demonstrating the benefits of our data transfer scheme. **Progressive Nets** notably leverages the previously trained weights the most constructively, as it achieves the highest average and final task success of all weight transfer methods.

The data transfer methods, **DARC** and **Off-Policy IW**, perform worse than if the agent learned each task from scratch with the same number of online samples. However, augmenting DARC with our pre-training scheme significantly improves its performance. In the Key Insertion tasks, both of the data transfer methods perform competitively with our approach, which suggests that the exact differences in the importance weights are less important here. However, **Ours** is the only method that achieves both high average and high final task performance.

In Fig. 4, we visualize the lifetime performance of each method on the *hard* task sequence of both simulated domains. Of the baselines, we select **Scratch**, **Progressive Nets**, and **DARC** to visualize. The latter two are chosen as the representative weight transfer and data transfer approaches. We evaluate these methods on the two sequences and plot the average over 5 trials each with a different random seed. On the D'Claw sequence, our approach has a final performance of 87%, with a slight advantage over **Progressive Nets**, which has a success rate of 80%. In the Key Insertion domain, however, our approach on average achieves a final success rate of 87% while **Progressive Nets** and **DARC** perform comparably to learning from scratch.

Method	ROBEL D'Claw		Key Insertion	
	Average Perf	Final Perf	Average Perf	Final Perf
Scratch (50k)	0.28 ± 0.02	0.59 ± 0.03	0.26 ± 0.02	0.52 ± 0.04
Scratch (100k)	0.47 ± 0.02	0.85 ± 0.03	0.45 ± 0.03	0.69 ± 0.04
Prog. Nets	0.41 ± 0.02	0.74 ± 0.02	0.50 ± 0.04	0.75 ± 0.03
Fine-tuning	0.27 ± 0.02	0.50 ± 0.04	0.49 ± 0.03	0.63 ± 0.03
DARC	0.23 ± 0.01	0.56 ± 0.03	0.60 ± 0.02	$\mathbf{0.89 \pm 0.02}$
Off-policy IW	0.33 ± 0.04	0.59 ± 0.04	$\mathbf{0.69 \pm 0.03}$	$\mathbf{0.82 \pm 0.03}$
Ours	$\mathbf{0.49 \pm 0.02}$	$\mathbf{0.82 \pm 0.02}$	$\mathbf{0.69 \pm 0.03}$	$\mathbf{0.87 \pm 0.03}$
Ours (warm-start)	0.32 ± 0.05	0.61 ± 0.07	0.50 ± 0.03	0.62 ± 0.04
Ours (w. DARC weights)	0.41 ± 0.03	$\mathbf{0.77 \pm 0.03}$	$\mathbf{0.66 \pm 0.01}$	0.76 ± 0.03

Table 1: The average and final task success of each method on the ROBEL D'Claw and Key Insertion domains. In each simulated domain, we evaluate on 5 *random* task sequences. We report the mean and standard error in each entry. Scratch (100k) refers to learning from scratch for 100k environment steps, *twice* the amount of other methods.

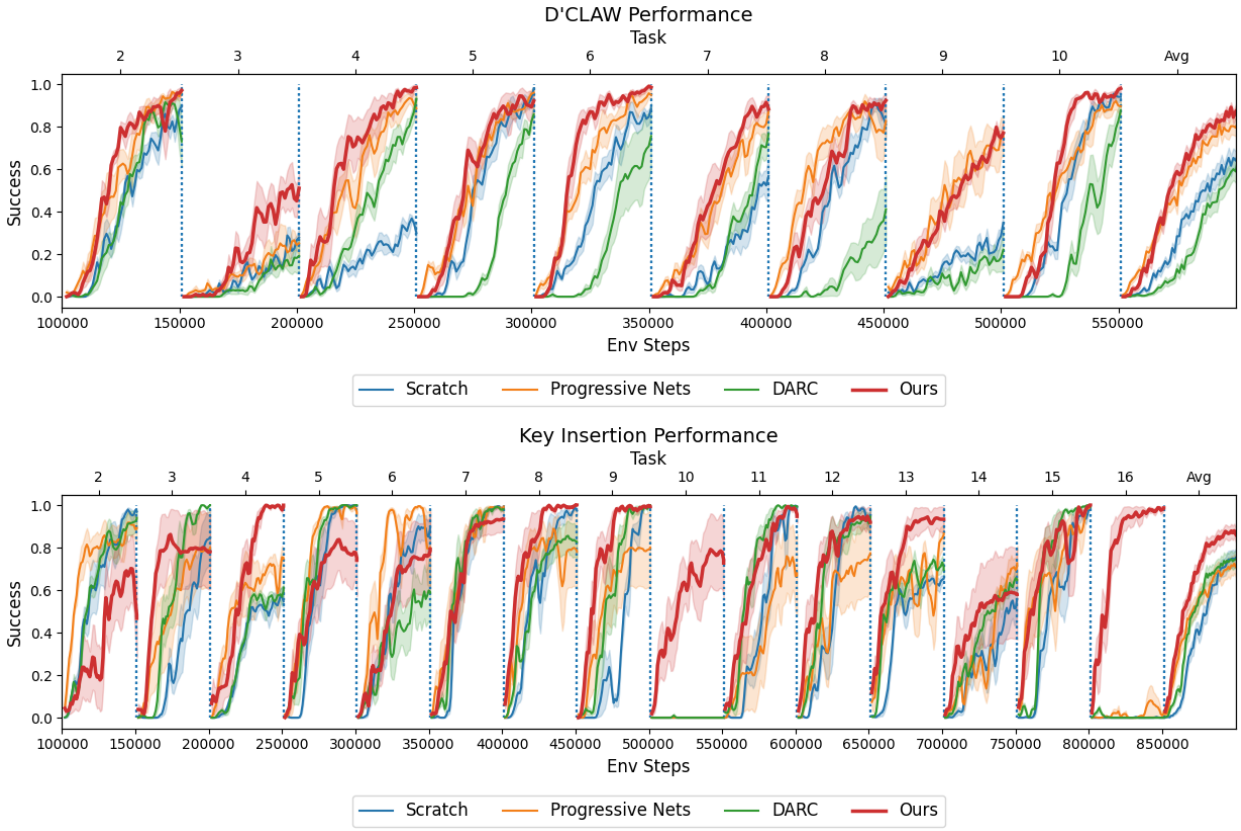
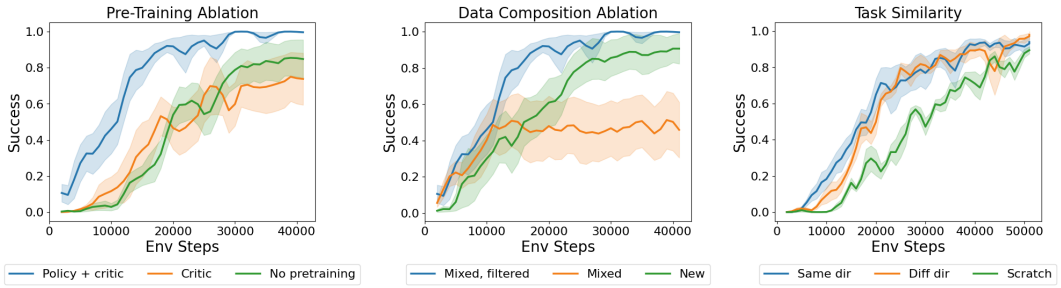


Figure 4: Lifetime performance of our method and select baselines on the *hard* task sequence of each domain. Shaded regions depict the standard error across 5 random seeds, and the solid lines depict their average. The final curve averages the learning curves across all of the tasks in the sequence. Learning curves for different tasks are separated by blue dotted vertical lines.

Improved efficiency. A desirable property in sequential learning is compounding learning, that is, improved data efficiency as more tasks are seen. We evaluate this property on a held-out horizontal insertion task after training on a varying number of tasks, and summarize the results in terms of the final task success in Table 2. Generally, the task success trends upwards as we increase the number of prior tasks we train on, suggesting that the data efficiency of our algorithm improves as we provide more tasks. We hypothesize that the performance improves with more tasks because of the heterogeneity in our task sequence, i.e., having both horizontal (H) and vertical (V) insertion tasks. In particular, when $N = 1$ and $N = 3$, all the prior tasks are V tasks. When $N = 5$, the first H task is introduced, leading to better transfer on the target task, which is also an H task. The introduction of the second H task in the $N = 7$ case leads to another improvement, albeit a smaller one.

# of Total Tasks	1	3	5	7
# of Horizontal Tasks	0	0	1	2
Ours	0.48	0.54	0.81	0.84

Table 2: Average task success after learning a varying number of tasks. The performance trends upwards with more tasks.



(a) (b) (c)

Figure 5: (a-b) Ablations of the pre-training and online phases, evaluated on 5 different target tasks. (c) Effect of similarity of source and target tasks. Shaded regions indicate the standard error across the 5 trials; the solid lines represent their average.

5.3 Ablations: Investigating Advantages of Prior Experience

The results from Sec. 5.2 demonstrate that the previously collected data samples, when appropriately leveraged, are a powerful form of knowledge transfer. To answer question **(2)** and to verify our algorithm utilizes this prior experience more effectively than alternative design choices, we ablate the pre-training and online improvement stages of our framework. The ablations are evaluated on 5 random target tasks.

Pre-training. Our algorithm pre-trains both the policy and critic with the relabeled data from $\mathcal{D}^{\text{src}}$ as its first step of learning a new task. Alternatively, we can pre-train the critic only or not pre-train at all. In this comparison, we randomly initialize all weights prior to pre-training to isolate its effects. The results, averaged across 5 different target tasks for each method, are presented in Fig. 5a, and suggest that pre-training both the policy and critic weights leads to more efficient learning, compared to only pre-training the critic weights and no pre-training.

Online improvement. In the online improvement step, our algorithm trains on a mixture of filtered prior experience and new experience collected online. We compare this data composition to (1) uniformly mixing the prior and new experience, akin to standard experience replay, and (2) training on new experience only. Again, we relabel all of the previously collected data with Eqn. 1. For each variant, we first randomly initialize the policy and critic weights and perform the pre-training phase. Then, in the online fine-tuning phase, we evaluate each of the three data composition schemes. As shown in Fig. 5b, filtering out the prior samples that are unlikely under the current dynamics leads to improved data efficiency. In fact, the standard experience replay scheme finds a suboptimal policy as a result of training on unfiltered data from previous tasks.

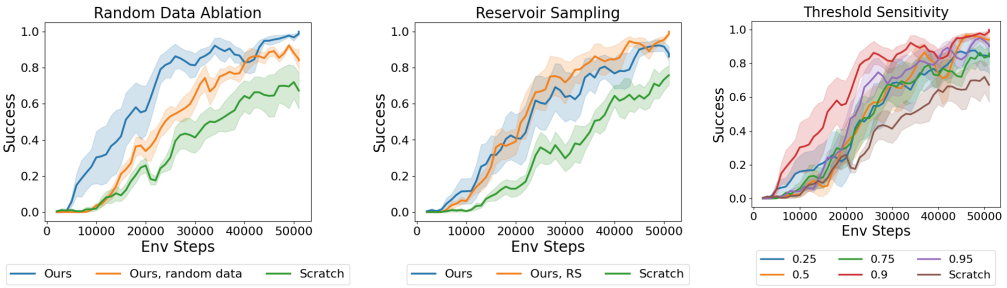
We next investigate the effects of the following factors on our method: the similarity of the source and target tasks, the quality of the prior data, the amount of prior data, and the thresholding value from Eqn. 2.

Task similarity. We design two transfer setups of different task similarity levels in the D'Claw environment, (1) Same Direction: the source and target tasks differ in the valve shape, *but* the objective of both is to rotate the valve in the same direction, and (2) Different Direction: the source and target tasks differ in the valve shape, *and* the objectives differ in their desired rotation direction. In Fig. 5c, we plot the learning curves for these two setups, along with learning from scratch. Despite the different objectives, the data collected in the two scenarios are similarly useful for transfer with our method. Since we relabel the rewards prior to transfer, the differences in the objectives are significantly mitigated.

Random data. In this experiment, we study the importance of the quality of the prior experience by evaluating our method with data collected with a *random* policy in the source task. We perform this experiment in the D'Claw environment on 5 different pairs of source-target tasks, and plot the learning curves in Fig. 6a. With a dataset collected in the source task with a random policy, our method performs slightly worse compared to when the data is collected with a learning policy. Nonetheless, transferring random data is still advantageous to learning completely from scratch. This suggests that our method is not particularly sensitive to the composition of the data.

Reservoir sampling. While the experiments in Sec. 5.2 study transfer performance while retaining *all* of the previously collected data, the storage and computation requirements of our algorithm also grow with the number of tasks learned and can become infeasible to execute. Therefore, we conduct an experiment with reservoir sampling in the final task of the 5 random task sequences in the D'Claw domain. Of the 500K samples collected in the first nine tasks, we only select 50K transitions to transfer to the target task. As depicted in Fig. 6b, the performance of our method does not degrade with the reduced buffer size, indicating that the 50K samples are sufficient for strong transfer.

Sensitivity of threshold. In Fig. 6c, we plot the transfer results with different threshold values: 0.25, 0.5, 0.75, 0.90, and 0.95, and find that the smaller thresholds show slightly worse transfer performance. Similarly, a large threshold like 0.95 also leads to a drop in performance. Overall however, our method is quite robust to different threshold values.



(a) (b) (c)

Figure 6: Studies of different factors on our method: (a) randomly collected source data, (b) quantity of source data, and (c) threshold value. Shaded regions indicate the standard error across the 5 trials; the solid lines represent their average.

Task	1	2	3	4	5	6	7	8	9	10
Scratch	1.11±0.05	2.65±0.24	1.62±0.02	1.27±0.09	0.83±0.15	2.81±0.23	1.90±0.03	2.11±0.28	1.30±0.23	1.99±0.19
Ours	–	**0.96±0.05**	**1.30±0.25**	**0.66±0.08**	**0.53±0.05**	**0.57±0.01**	**0.50±0.01**	**0.81±0.22**	**0.78±0.06**	**0.60±0.01**

Table 3: The final distance to goal (in centimeters) for each of the 10 tasks on the Franka robot. We evaluate the policies after 10K environment steps for 10 trials, and report the means and standard errors.

5.4 Learning in the Real World

To answer question **(3)**, we evaluate our approach with a Franka Emika Panda robot arm on a sequence of challenging tasks with varying physical setups.

Experimental setup. We evaluate our algorithm on a physical robot arm on a sequence of 10 object manipulation tasks, ranging from capping a bottle to inserting a block (see Fig. 1). The robot's state consists of its end-effector pose, and takes actions corresponding to 3D Cartesian end-effector displacements at 5 Hz. We evaluate **Ours**, as it outperformed the other variants in the simulations.

Results. We focus our evaluation on forward transfer and compare the learning efficiency of our algorithm to learning each task from scratch. After each epoch, we roll out the mean policy for 10 evaluation episodes, and plot the average distance to the goal position versus the number of training environment steps averaged across all tasks after the first one in Fig. 7. The individual learning curves for each task are included in App. C. Overall, our algorithm achieves an average distance of 0.75 centimeters to the goal within 10K environment steps of a new task compared to an average distance of 1.83 centimeters achieved by learning from scratch in the same number of steps. In Table 3, we report the average final performance by task for the two methods. Like our simulation results, compared to learning from scratch, our algorithm learns each task more efficiently by leveraging the retained experiences. Further, we qualitatively observe that one key benefit of retaining and relabeling the data is that it affords the agent fewer exploration steps in the regions of the state space that are irrelevant for the task at hand.

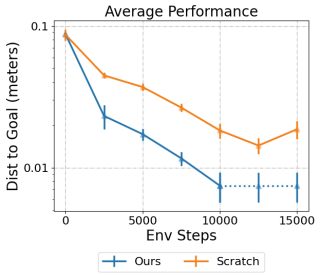


Figure 7: The distance to goal averaged across 9 tasks on the physical robot. The error bars represent 95% CI.

6 Discussion

Algorithms for lifelong learning should allow robots to solve new tasks in succession by accumulating and building upon prior experience. To this end, we presented a simple framework that allows robots to learn sequences of tasks in a data-efficient manner by retaining and reusing prior experience. Our approach partitions the learning of each task into two stages: first, it pre-trains on relabeled prior data; second, it improves with online experience. In our experiments, we demonstrated this transfer of experience as a powerful mechanism for lifelong learning on physical robots.

Nonetheless, limitations of this approach remain, which we hope to address in future work. First, our algorithm assumes that all the previously collected data can be stored, which may present scalability challenges as the number of tasks to learn grows. However, we expect that naive approaches to discard experiences, e.g. uniformly by task, may be effective towards mitigating this challenge. Finally, the algorithm is in principle applicable to any task that can be solved with reinforcement learning; but, our experiments consider only reaching, placing, and insertion style tasks. We hope to study a wider range of manipulation tasks, which may require the use of visual observations or force feedback. We can also expect transfer to become more difficult as the tasks become more dissimilar, an interesting challenge for future investigation.

ACKNOWLEDGMENTS

This research was supported by an NSF Graduate Research Fellowship, Google, ONR grant N00014-21-1-2685, and JPMorgan Chase & Co.. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

REFERENCES

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Neural Information Processing Systems (NeurIPS)*, 2017.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 2020.

Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. Meta reinforcement learning for sim-to-real domain adaptation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. Domain adaptation on the statistical manifold. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Yogesh Balaji, Mehrdad Farajtabar, Dong Yin, Alex Mott, and Ang Li. The effectiveness of memory replay in large scale continual learning. *arXiv preprint arXiv:2010.02418*, 2020.

Jihwan Bang, Heesu Kim, YoungJoon Yoo, Jung-Woo Ha, and Jonghyun Choi. Rainbow memory: Continual learning with a memory of diverse samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8218–8227, 2021.

Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning for differing training and test distributions. In *Proceedings of the 24th international conference on Machine learning*, pp. 81–88, 2007.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. Sample selection bias correction theory. In *International conference on algorithmic learning theory*, pp. 38–53. Springer, 2008.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

Benjamin Eysenbach, Swapnil Asawa, Shreyas Chaudhari, Ruslan Salakhutdinov, and Sergey Levine. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. *International Conference on Learning Representations (ICLR)*, 2021.

Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *AAMAS*, pp. 720–727, 2006.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793. IEEE, 2017.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. *International Conference on Machine Learning (ICML)*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2018a.

Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. *Neural Information Processing Systems (NeurIPS)*, 2018b.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *international conference on robotics and automation (ICRA)*, 2017.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *Robotics: Science and Systems (RSS)*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018b.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. *International Conference on Learning Representations (ICLR)*, 2018.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ryan Julian, Benjamin Swanson, Gaurav S Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Efficient adaptation for end-to-end vision-based robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

Leslie Pack Kaelbling. Learning to achieve goals. *IJCAI*, 1993.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Conference on Robot Learning (CoRL)*, 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation*, 2004.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.

Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pp. 143–173. Springer, 2012.

Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *international conference on Machine learning*, 2008.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 2016.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2019.

Emilio Parisotto, Lei Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2016.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. *International Conference on Machine Learning (ICML)*, 2018.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P Lillicrap, and Greg Wayne. Experience replay for continual learning. *Neural Information Processing Systems (NeurIPS)*, 2019.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *International Conference on Learning Representations (ICLR)*, 2016a.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016b.

Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, and Karen Simonyan. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *International Conference on Machine Learning (ICML)*, 2018.

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *Robotics: Science and Systems (RSS)*, 2019.

Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2020.

Shengyang Sun, Daniele Calandriello, Huiyi Hu, Ang Li, and Michalis Titsias. Information-theoretic online memory selection for continual learning. In *International Conference on Learning Representations*, 2021.

Yunzhe Tao, Sahika Genc, Tao Sun, and Sunil Mallya. Repaint: Knowledge transfer in deep actor-critic reinforcement learning. *arXiv preprint arXiv:2011.11827*, 2020.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, 2008.

Russell L Tedrake. *Applied optimal control for dynamically stable legged locomotion*. PhD thesis, Massachusetts Institute of Technology, 2004.

Yee Whye Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Neural Information Processing Systems (NeurIPS)*, 2017.

Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. Importance weighted transfer of samples in reinforcement learning. In *International Conference on Machine Learning*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Marc Toussaint. Robot trajectory optimization using approximate inference. *International Conference on Machine Learning (ICML)*, 2009.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *CogSci*, 2017.

Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv:1507.04888*, 2015.

Annie Xie, Avi Singh, Sergey Levine, and Chelsea Finn. Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning*, 2018.

Fan Yang, Chao Yang, Huaping Liu, and Fuchun Sun. Evaluations of the gap between supervised and reinforcement lifelong learning on robotic manipulation tasks. In *Conference on Robot Learning*, pp. 547–556. PMLR, 2022.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. *Neural Information Processing Systems (NeurIPS)*, 2020.

Haiyan Yin and Sinno Pan. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Neural Information Processing Systems (NeurIPS)*, 2020.

Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. *International Conference on Machine Learning (ICML)*, 2004.

Tony Z Zhao, Anusha Nagabandi, Kate Rakelly, Chelsea Finn, and Sergey Levine. Meld: Meta-reinforcement learning from images via latent state models. *Conference on Robot Learning (CoRL)*, 2020.

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

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. *AAAI*, 2008.

A Hyperparameter Details

Policy and critic networks. For all experiments, we implement our algorithm on top of the soft actor-critic (SAC) Haarnoja et al. (2018b) algorithm. The policy and critic are each MLPs with 2 fully-connected layers of size 256 and ReLU non-linearities.

Domain classifier networks. For all experiments, the domain classifier networks D_1, D_2 are each MLPs with 2 fully-connected layers of size 256 and ReLU non-linearities. Following Eysenbach et al. (2021), we inject Gaussian input noise with $\sigma = 1.0$ to combat overfitting at the beginning when there are few samples from the task currently being learned.

Learning rates. For our simulated experiments, we use the Adam optimizer and learning rate of $3\text{e}{-4}$ for the policy and critic updates, and a learning rate of $1\text{e}{-3}$ for the domain classifiers. For our robot experiments, we use a learning rate of $1\text{e}{-3}$ for the policy, critic, and domain classifier updates.

Pre-training phase. For all experiments, we pre-train the policy and critic of **Ours** and **Ours (warm-start)** with the relabeled data from the restored replay buffers for 10k iterations before online improvement.

Online improvement phase. In the online improvement phase of **Ours** and **Ours (warm-start)**, we use a threshold value of $\gamma = 0.9$ for all experiments. We re-filter the source dataset $\mathcal{D}^{\text{src}}$ after every 1000 iterations. At the beginning of the online phase, the batches used for the policy and critic updates are composed of 50% filtered prior data and 50% new online data. We increase this ratio ρ of new data to prior data according to:

$$\rho = \text{clip}(|\mathcal{D}^i| + 12500)/25000, 1.0)$$

where $|\text{b}^i|$ is the size of the replay buffer for the current task. In other words, the ratio is increased linearly and reaches 1.0 once 25k steps have been taken in the current task.

B Environment Setup

B.1 Simulated Experiments

In the experiments on the D'Claw benchmark (Yang et al., 2022), the objective is to rotate the valve to a target angle. Across tasks, we vary the valve shape, position feedback gain, friction coefficients, position of the value, target angle, desired direction of rotation. The reward function across all tasks is:

$$-0.5 \cdot |s_\theta - g_\theta| + \mathbb{1}(|s_\theta - g_\theta| < 0.05),$$

where s_θ and g_θ are the valve's current and target angles respectively.

In the simulated key insertion experiments, we use the Robosuite Zhu et al. (2020) simulation framework which employs the MuJoCo physics engine Todorov et al. (2012). The robot's state includes the robot's joint positions and velocities, its end-effector pose, and a binary indicator of whether the key is inside the robot's gripper. The action controls the deltas in the 3D-position and the z-axis rotation of the robot's end-effector. Between tasks, we vary the xy-position of the box g_{xy}, the relative orientation of the key to the hole g_θ, and the length of the key l. The reward function across all tasks is:

$$\mathbb{1}(\|\mathbf{s}_{xy} - g_{xy}\|_2 \leq 0.03) \cdot \mathbb{1}(\mathbf{s}_z - l \leq g_{z,u} + 0.005) - \tanh(\|10 \cdot (\mathbf{s}_{xy} - g_{xy}\|_2 + |\mathbf{s}_z - g_{z,l}|)) - \tanh(|\mathbf{s}_\theta - g_\theta|),$$

where $\mathbf{s}_{xy}$ is the xy-coordinates of the robot end-effector, $\mathbf{s}_\theta$ is the z-axis rotation of the end-effector, $g_{z,u}$ is the z-coordinate of the top of the box, and $g_{z,l}$ is the z-coordinate of the bottom. Note, in Fig. 3, that the box is composed of three "layers," each colored with a different shade of gray. We measure task success on a scale of $\{0, 1, 2, 3\}$ based on which layer the key head reaches at the final time-step of the episode, and report all results in terms of the normalized scores by dividing by 3.

B.2 Robot Experiments

In our robot experiments, we design a series of 10 tasks with varying setups and objectives. We describe these tasks below.

Task 1: Reaching. The robot is to reach a fixed goal position $g_{\text{reach}} = [0.466, 0.028, 0.153]^T$ without any obstacles. The reward function for this task is

$$r^1(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{reach}}\|_2),$$

where $\mathbf{s}_{xyz}$ are the 3D Cartesian coordinates of the robot end-effector.

Task 2: Marker insertion (A). The objective is to align the marker to the corresponding hole of a marker rack. We specify this objective through a goal position $g_{\text{marker-A}} = [0.443, 0.014, 0.152]^T$ for the end-effector. The reward function for this task is

$$r^2(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{marker-A}}\|_2).$$

Task 3: Eraser insertion. The objective is to align the eraser to the corresponding hole of the same rack from the previous task. We specify this objective through a goal position $g_{\text{eraser}} = [0.448, 0.067, 0.152]^T$ for the end-effector. The reward function for this task is

$$r^3(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{eraser}}\|_2),$$

Task 4: Bottle capping (A). The objective is to align the cap to a Gatorade bottle. We specify this objective through a goal position $g_{\text{bottle-A}} = [0.469, 0.053, 0.189]^T$ for the end-effector. Different from the previous reward functions, we additionally specify a waypoint $w^1 = [0.469, 0.053, 0.230]^T$, as the bottle is significantly taller than the other objects from previous tasks. The reward function for this task is

$$\begin{aligned} r^4(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2}(&\mathbb{1}(\|\mathbf{s}_{xy} - w^1_{xy}\|_2 < 0.03) \cdot \mathbb{1}(|\mathbf{s}_z - w^1_z| \leq 0.005) \\ &+ (1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{bottle-A}}\|_2))), \end{aligned}$$

where $\mathbf{s}_{xy}$ are the xy-coordinates of the end-effector, w^1_{xy} are the xy-coordinates of the waypoint, $\mathbf{s}_z$ is the z-coordinate of the effector, and w^1_z is the z-coordinate of the waypoint.

Task 5: Bottle capping (B). The objective is to align the cap to a plastic water bottle. We specify this objective through a goal position $g_{\text{bottle-B}} = [0.469, -0.014, 0.210]$ for the end-effector. Similar to the previous bottle task, because this bottle is also relatively tall, we additionally specify a waypoint $w^2 = [0.469, -0.014, 0.240]$. The reward function for this task is:

$$\begin{aligned} r^5(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2}(&\mathbb{1}(\|\mathbf{s}_{xy} - w^2_{xy}\|_2 < 0.03) \cdot \mathbb{1}(|\mathbf{s}_z - w^2_z| \leq 0.005) \\ &+ (1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{bottle-B}}\|_2))), \end{aligned}$$

where w^2_{xy} are the xy-coordinates of the waypoint and w^2_z is the z-coordinate of the waypoint.

Task 6: Block insertion (A). The objective is to align the square block to the corresponding hole of the toy cube. We specify this objective through a goal position $g_{\text{block-A}} = [0.472, -0.002, 0.125]^T$ for the end-effector. The reward function for this task is

$$r^6(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{block-A}}\|_2),$$

Task 7: Block insertion (B). The objective is to align the parallelogram-shaped block to the corresponding hole of the toy cube. We specify this objective through a goal position $g_{\text{block-B}} = [0.465, -0.015, 0.140]^T$ for the end-effector. The reward function for this task is

$$r^7(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{block-B}}\|_2),$$

Task 8: Block insertion (C). The objective is to align the octagon-shaped block to the corresponding hole of the toy cube. We specify this objective through a goal position $g_{\text{block-C}} = [0.472, -0.060, 0.140]^T$ for the end-effector. The reward function for this task is

$$r^8(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{block-C}}\|_2),$$

Task 9: Bottle capping (C). The objective is to align the cap to the Vitamin water bottle. We specify this objective through a goal position $g_{\text{bottle-C}} = [0.460, -0.032, 0.185]^T$ for the end-effector. The reward function for this task is

$$r^9(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{bottle-C}}\|_2),$$

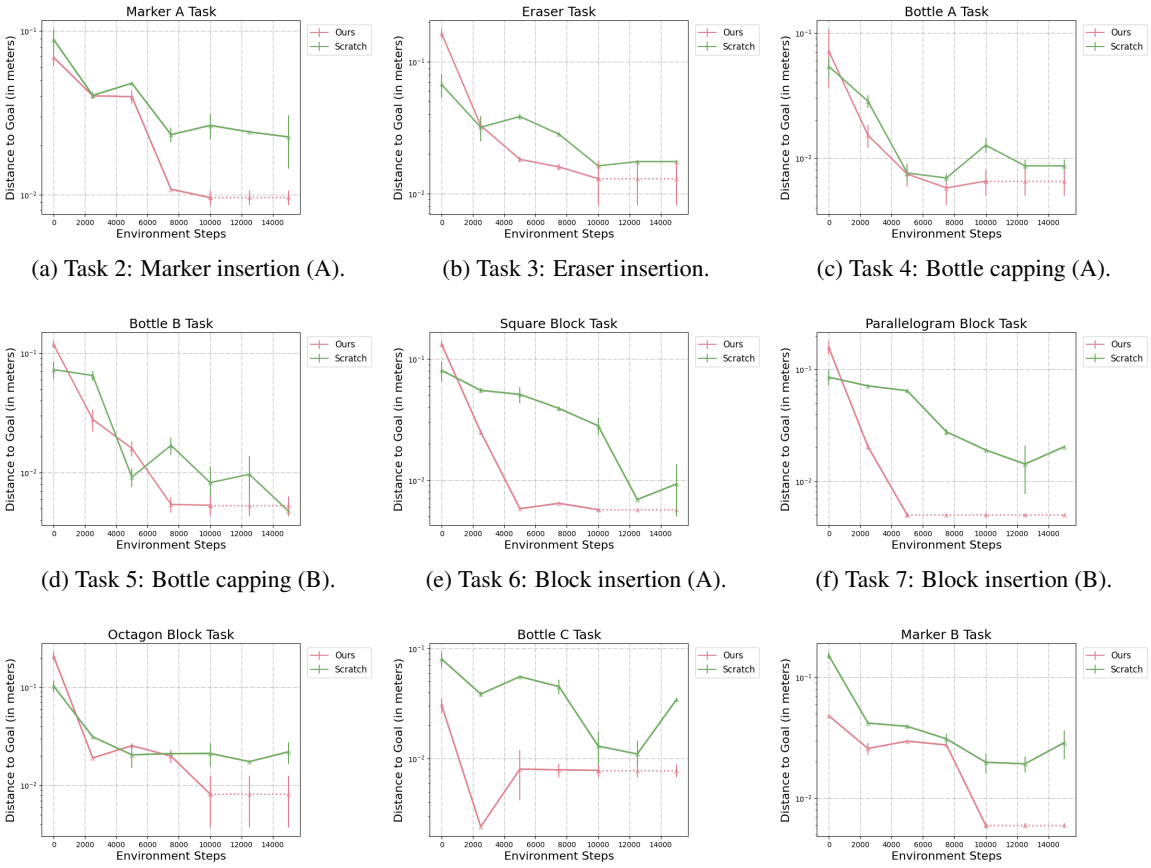
Task 10: Marker insertion (B). The objective is to align the cap to the Vitamin water bottle. We specify this objective through a goal position $g_{\text{marker-B}} = [0.444, -0.020, 0.118]^T$ for the end-effector. The reward function for this task is

$$r^9(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{marker-B}}\|_2),$$

All of the goal positions lie within a bounded region roughly of size 2cm $\times$ 4cm $\times$ 4cm.

C Additional Plots

In Fig. 8, we provide the individual learning curves for each of the 10 tasks from our robot experiments. For each data-point, we average the distance to the goal position (in meters) at the final time-step across 10 evaluation episodes. Our method attains a lower average distance after 10K time-steps than learning from scratch for all 5 tasks following the initial reaching task.



(a) Task 2: Marker insertion (A). (b) Task 3: Eraser insertion. (c) Task 4: Bottle capping (A).

(d) Task 5: Bottle capping (B). (e) Task 6: Block insertion (A). (f) Task 7: Block insertion (B).

(g) Task 8: Block insertion (C). (h) Task 9: Bottle capping (C). (i) Task 10: Marker insertion (B).

Figure 8: Individual learning curves for each task.

D Power Analysis

Deep RL experiments often exhibit high variance across different random seeds. To address this concern, we performed bootstrap analysis on the main results in Table 4 to test their statistical significance (Henderson et al., 2018; Colas et al., 2018). Based on the analysis, **DARC** and **Ours** achieve similar final performance in the Key Insertion tasks but **Ours** has higher average performance. However, there is ambiguity about the ordering between **Off-policy IW** and **Ours** in the Key Insertion tasks. Hence, we run another 5 seeds for both methods. After running the bootstrap test with the additional trials, we find that there is still not enough evidence to show an order relation between the two methods. Hence, we conclude the two methods achieve similar performance in this environment. We report the results of the bootstrap confidence interval test, with 10000 bootstrap iterations, at significance level 0.05.

E Wall-Clock Time

In Table 5, we report the wall-clock time of each algorithm, in minutes, on the final task of the sequence in each simulated domain.

Method	ROBEL D'Claw		Key Insertion	
	CI for Average Perf	CI for Final Perf	CI for Average Perf	CI for Final Perf
Scratch (50k)	$[0.173, 0.268]$	$[0.150, 0.308]$	$[0.114, 0.294]$	$[0.311, 0.514]$
Prog. Nets	$[0.037, 0.132]$	$[0.053, 0.154]$	$[0.093, 0.286]$	$[0.017, 0.227]$
Fine-tuning	$[0.175, 0.291]$	$[0.227, 0.425]$	$[0.116, 0.296]$	$[0.150, 0.331]$
DARC	$[0.218, 0.307]$	$[0.180, 0.340]$	$[0.023, 0.168]$	–
Off-policy IW	$[0.072, 0.258]$	$[0.130, 0.323]$	–	–
Ours (warm-start)	$[0.074, 0.274]$	$[0.062, 0.366]$	$[0.099, 0.288]$	$[0.150, 0.347]$
Ours (w. DARC weights)	$[0.018, 0.162]$	–	–	$[0.029, 0.193]$

Table 4: The estimated confidence intervals, with bootstrap test level 0.05, for $\mu_1 - \mu_2$, where μ_1 is the average final return achieved by Ours and μ_2 is that of the comparisons. When there is not enough evidence to show an order relation between μ_1 and μ_2, we denote this result by –.

Method	ROBEL D'Claw	Key Insertion
Scratch	33	66
Prog. Nets	83	185
Fine-tuning	33	66
DARC	33	60
Off-policy IW	33	62
Ours	41	93
Ours (warm-start)	42	94
Ours (w. DARC weights)	40	71

Table 5: Wall-clock time of each algorithm (in minutes) on the final task of the sequence in each simulated domain.