Lifelong Robotic Reinforcement Learning by Retaining Experiences

Annie Xie and Chelsea Finn
Department of Computer Science
Stanford University
anniexie@stanford.edu

Abstract

Multi-task learning ideally allows robots to acquire a diverse repertoire of useful skills. However, many multi-task reinforcement learning efforts assume the robot can collect data from *all* tasks at *all* times. In reality, the tasks that the robot learns arrive sequentially, depending on the user and the robot's current environment. In this work, we study a practical sequential multi-task RL problem that is motivated by the practical constraints of physical robotic systems, and derive an approach that effectively leverages the data and policies learned for previous tasks to cumulatively 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 impractical round-robin data collection. On a Franka Emika Panda robot arm, our approach incrementally learns ten challenging tasks, including bottle capping and block insertion.

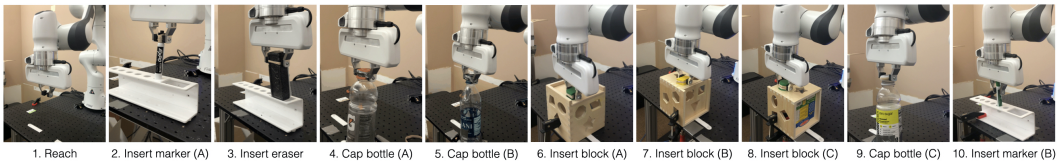


Figure 1: A Franka Emika robot arm learns a sequence of manipulation tasks, including block insertion and bottle capping, by retaining experience from previous tasks.

1 Introduction

General-purpose robots should be capable of learning and performing a multitude of tasks in their, ideally never-ending, lifetimes. Multi-task learning paves a promising path towards such robots by capitalizing on the potential shared structure between tasks to facilitate more efficient learning of each task. While multi-task learning has proven successful in a number of domains, including control tasks in simulated environments [33, 30, 41, 14, 31, 48], there has been a considerably smaller effort in applying these algorithms to real robotic systems [19]. A key but limiting assumption of the multi-task paradigm is that the agent can collect data from each task in round-robin fashion, which can be highly impractical in many robotic learning setups. Imagine asking a robot to learn two tasks with vastly different setups: bottle capping, which requires fixing a bottle on a table, and block sorting, which requires a sorting cube (see Fig. 1). Round-robin data collection for the two tasks would demand significant human assistance, that is, to repeatedly swap the physical setup of the robot's learning environment or to engineer a single environment suitable for learning all tasks.

Beyond the impracticality of the framework, it also offers limited flexibility in the way tasks are assigned. A robot may encounter new tasks that it needs to learn over the course of its lifetime,

NeurIPS 2021 Workshop on Robot Learning: Self-Supervised and Lifelong Learning, Virtual, Virtual

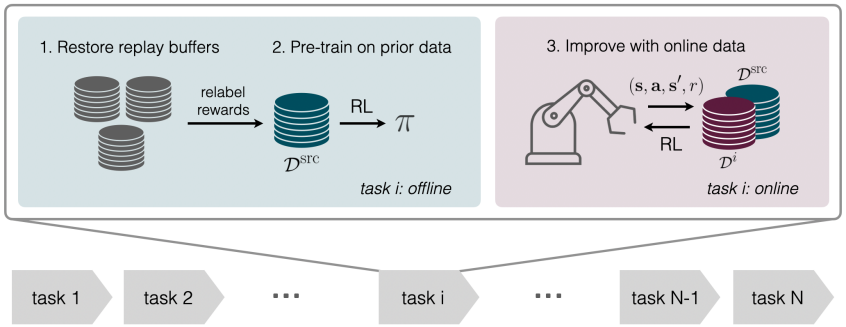


Figure 2: In our framework, we perform two steps during each new task. First, we pre-train on the experience from earlier tasks (left). To align the data to the same objective, we use the underlying reward function of the upcoming task to relabel this experience before pre-training. Second, we learn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

which the current multi-task learning setup fails to address. Instead, solving tasks one after another tremendously reduces the amount of physical modifications to the environment that is required to switch between tasks, and naturally permits human users to dynamically assign new tasks to the robot. This paradigm can thus bring us a step closer to lifelong-learning robots that can continuously accrue a diverse skill-set and learn new tasks more quickly.

Many lifelong learning algorithms limit the amount of data that can be stored over the sequence of tasks and focus on the problem of catastrophic forgetting of old tasks [20]. However, an equally critical aspect of lifelong learning is addressing how to leverage prior experience to accelerate learning of future tasks. Moreover, in most robot learning settings with modern compute and hard drives, it is 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 [23]. Therefore, we develop a simple but powerful technique for selectively transferring relevant experiences and implement it in a sequential robotic task learning system.

The core contribution of this paper is a framework for efficient lifelong reinforcement learning that is suitable for physical robots. In particular, the robot 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 collects data in the new task and *improves* its policy with online data and the relevant prior data. We demonstrate the efficacy of our approach on a Franka arm, which incrementally learns to solve a sequence of ten challenging tasks with differing objectives and physical setups (illustrated in Fig. 1).

2 Lifelong Reinforcement Learning via Experience Transfer

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 $r^i(\mathbf{s}, \mathbf{a})$. We also assume the agent can access the full reward function for each task $r^i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which 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.

Our algorithm, depicted in Fig. 2 and summarized in Alg. 1, begins by learning the first task from scratch with vanilla Soft Actor-Critic [13]. For each following new task, we restore the replay buffer(s) from the previous task(s) and pre-train on this data (Sec. 2.1), and subsequently improve the agent's policy on the task with online experience (Sec. 2.2). Both of these stages reuse the previously collected, offline data in order to maximize the sample efficiency of our algorithm.

2.1 Pre-Training on Prior Experience

At the arrival of a new task $\mathcal{T}^i$, we restore the replay buffers $\mathcal{D}^{1:i-1}$ from the previous tasks, and optionally, the weights from the most recent task. 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 $\mathcal{T}^i$

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

Algorithm 1 Lifelong RL

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

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 classifiers ψ, ω
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 $\mathcal{D}^{\text{src}} \cup \mathcal{D}^i$
    $\diamond\ \theta^i \leftarrow \theta^i - \alpha \nabla_{\theta^i} \mathcal{L}_Q$
    $\diamond\ \phi^i \leftarrow \phi^i - \alpha \nabla_{\phi^i} \mathcal{L}_\pi$
    Classifier updates:
    $\diamond\ \psi \leftarrow \psi - \alpha \nabla_\psi \mathcal{L}_\psi$
    $\diamond\ \omega \leftarrow \omega - \alpha \nabla_\omega \mathcal{L}_\omega$
  Filter source buffer $\mathcal{D}^{\text{src}}$ (Eqn. 2)
Return: $\mathcal{D}^i$, θ^i, ϕ^i

We then 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, and thus accelerate learning in the new task. The pre-training subroutine is summarized in Alg. 2.

2.2 Improving with Online Experience

To improve the pre-trained policy from Sec. 2.1, 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, depending on the dynamics gap.

Likelihood-free importance weights. Let $\mathcal{D}^i$ 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 $w(\mathbf{s}, \mathbf{a}, \mathbf{s}') = p^{\text{tgt}}(\mathbf{s}, \mathbf{a}, \mathbf{s}')/p^{\text{src}}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, which we can also express as $w(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \frac{p(\text{target}|\mathbf{s},\mathbf{a},\mathbf{s}')}{p(\text{source}|\mathbf{s},\mathbf{a},\mathbf{s}')} \cdot \frac{p(\text{source})}{p(\text{target})}$. 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 should 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: $\tilde{w}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \mathbb{1}\left(\frac{c_\psi(\mathbf{s},\mathbf{a},\mathbf{s}')}{1-c_\psi(\mathbf{s},\mathbf{a},\mathbf{s}')} \geq \gamma\right)$, where γ controls how conservatively samples are transferred.

Exploration via a state classifier. Finally, prior experience on related tasks may be informative for what states to explore in future tasks. For example, a robot solving a sequence of table-top manipulation tasks generally needs to move the arm towards objects. We build this heuristic into the filtering scheme through the ratio $p^{\text{src}}(\mathbf{s})/p^{\text{tgt}}(\mathbf{s})$. In practice, we introduce a second classifier on the state $c_\omega(\mathbf{s})$, and transfer samples for which $\frac{c_\omega(\mathbf{s})}{1-c_\omega(\mathbf{s})} < \zeta$. Our overall filtering rule is:

$$\mathcal{D}^{\text{src}} \leftarrow \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \mid (\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \in \mathcal{D}^{\text{src}}, \frac{c_\omega(\mathbf{s})}{1 - c_\omega(\mathbf{s})} < \zeta \text{ or } \frac{c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')}{1 - c_\psi(\mathbf{s}, \mathbf{a}, \mathbf{s}')} \geq \gamma\} \tag{2}$$

The complete online improvement phase is outlined in Alg. 3.

Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Overall
Scratch [13] (50k)	1.00	0.76	0.78	0.36	**0.60**	0.43	0.93	0.72	0.69	0.88	0.43	0.72	0.59	0.34	0.64	0.71	0.64 ± 0.04
Scratch [13] (100k)	1.00	0.75	0.53	0.84	0.57	0.54	1.00	0.66	0.76	1.00	0.52	0.80	0.96	0.46	0.96	0.59	0.73 ± 0.04
Prog. Nets [34]	–	0.57	0.74	0.50	0.38	0.21	0.69	0.24	0.26	0.43	0.53	0.34	0.71	0.03	0.69	0.36	0.45 ± 0.05
Fine-tuning [16]	–	0.83	0.79	0.44	0.54	0.42	0.66	0.53	0.74	0.36	0.51	0.49	0.74	0.23	0.96	0.17	0.56 ± 0.06
DARC [7]	–	0.33	0.67	0.33	0.50	0.21	0.40	0.54	0.33	0.63	**0.83**	0.56	0.62	0.17	0.97	0.62	0.52 ± 0.06
DARC [7] (pre-train)	–	0.48	0.67	0.49	0.43	0.28	0.87	0.39	0.49	0.52	0.79	0.76	0.63	0.23	0.87	0.44	0.56 ± 0.05
Standard IW	–	0.48	0.70	0.69	0.41	0.63	0.34	0.68	0.23	0.30	0.51	0.74	0.91	0.32	0.79	**0.87**	0.57 ± 0.05
Ours	–	**1.00**	0.80	**0.91**	0.53	0.59	**0.98**	**0.97**	0.63	**1.00**	0.73	0.83	0.88	0.37	0.87	0.76	$\mathbf{0.79 \pm 0.04}$
Ours (warm-start)	–	0.90	**0.82**	0.77	0.50	**0.66**	0.70	0.49	0.76	0.94	0.54	**0.86**	0.91	0.24	**0.99**	0.77	0.72 ± 0.05
Ours (w/o state cls)	–	0.52	0.78	0.57	0.57	0.52	0.91	0.90	**0.88**	0.66	0.66	0.58	**0.97**	**0.38**	0.82	0.79	0.70 ± 0.05

Table 1: Task success for 16 tasks averaged across 3 different task sequences. For the first task, we run the SAC algorithm for 100K time-steps. For the remaining tasks, we run each method for 50K time-steps and report the average final task performance. Our approach is significantly more successful than fine-tuning and learning from scratch. Further, our approach is more successful than training from scratch using 100K time-steps.

3 Experiments

We carefully study our method as well as several comparisons in a simulated robot environment. In particular, we compare to two methods that leverage prior weights, **Progressive Nets [34]** and **Fine-tuning [16]**. We also compare to methods that reuse prior data, **DARC [7]**; **DARC [7] (pre-train)**, DARC augmented with our pre-training procedure described in Sec. 2.1; and **Standard IW**, vanilla off-policy RL that reweighs samples with $w^{\text{OP}} = \pi^i(\mathbf{a}|\mathbf{s})/\pi^{i-1}(\mathbf{a}|\mathbf{s})$. We also study three variants of our algorithm: **Ours**, **Ours (warm-start)**, and **Ours (w/o state cls)**. The second restores the trained weights of the policy and critic from the previous task prior to the pre-training phase, and the third removes the filtering criterion based on the state classifier from **Ours**. The videos for the robot experiments are on our project webpage: https://sites.google.com/view/retain-experience/.

3.1 Evaluating Forward Transfer

We first construct a family of key-insertion tasks within Robosuite [51], using the MuJoCo physics engine [43]. 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. A full description of the environment is provided in App. D, and hyperparameter details are reported in App. C. In Table 1, we summarize the results of all methods on the 3 task sequences in terms of the task success. After only 50K time-steps on each task, **Ours** is on average more successful than all other methods, even when each task is learned from scratch with *twice* the number of time-steps. **Ours (warm-start)**, which additionally restores the weights from the previous task, performs slightly worse than the data-only variant, likely due to the diversity of the tasks. The worse performance of **Ours (w/o state cls)** highlights the importance of also filtering based on the state classifier.

Notably, the performance of Progressive Nets [34] is worse than that of naive fine-tuning. The lack of improvement can likely be explained by the low-dimensional states in the tasks, which makes representation transfer less critical and perhaps less useful than data or weight transfer. The comparison of DARC to DARC (pre-train) confirms the benefit of using our pre-training phase, and the lower success rate of DARC (pre-train) compared to our method suggests that the importance weights we define are much better suited for the sequential nature of the lifelong learning problem.

3.2 Learning in the Real World

Next, 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 and App. D for more setup details). 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. 3. The individual learning curves for each task are included in App. E. 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 4 of App. E, we report the average final performance by task for the two methods.

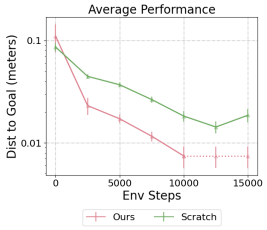


Figure 3: The learning curve averaged across tasks $i > 1$ on the physical robot.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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*, pages 81–88, 2007.

[6] 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.

[7] 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.

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

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

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

[11] 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.

[12] 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)*, 2018.

[13] 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)*, 2018.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] Dmitry Kalashnkov, Jake Varley, Yevgen Chebotar, Ben Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv*, 2021.

[20] 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.

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

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

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

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

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

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

[27] 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.

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

[29] 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.

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

[31] 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.

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

[33] 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)*, 2016.

[34] 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*, 2016.

[35] 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.

[36] 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.

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

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

[39] 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.

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

[41] 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.

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

[43] 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.

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

[45] 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.

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

[47] 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.

[48] 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.

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

[50] 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.

[51] 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.

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

A Related Work

Reinforcement learning has allowed physical robots to autonomously learn an impressive array of skills [21, 28], from locomotion [22, 40, 12, 26] to the manipulation of diverse objects [11, 18, 27, 2]. 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 [38, 24, 45, 10, 29, 50, 3, 36]. In RL agents, this knowledge can be transferred through representations [34, 6], learned models [9], network weights [8, 33], or experiences [17, 39, 25, 1, 42, 37]. Multi-task learning also aims to transfer knowledge across tasks but achieves this by learning the set of tasks together [30, 41, 14, 46, 48]. This framework has allowed robots to learn a range of goal-based tasks [17, 1], but may be less practical when each task has a unique physical setup. Unlike these prior works, we study the *sequential* multi-task 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.

A common form of sequential transfer is to reuse the weights of a policy network by fine-tuning them to the new task [8, 16] or distilling the learned behavior [28, 33, 30, 35]. 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 [39, 25, 42, 37, 7]. 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 [15], combat catastrophic forgetting [32], and accelerate learning of new tasks [1, 47, 37]. 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.

B Preliminaries

A task is defined by a Markov decision process with a continuous state space $\mathcal{S}$ and action space $\mathcal{A}$. In our robotic control problem, the state $\mathbf{s} \in \mathcal{S}$ consists of the Cartesian end-effector position and the action $\mathbf{a} \in \mathcal{A}$ corresponds to motor commands in the form of end-effector displacements. 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. B.1). To reuse experience from prior tasks, we measure the relevance of individual samples for a new task with importance weights (Sec. B.2).

B.1 Soft Actor-Critic

The soft actor-critic [13] (SAC) algorithm optimizes the maximum-entropy RL objective [52, 44], using off-policy data for more sample-efficient learning. In particular, SAC 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 $V(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[Q_\theta(\mathbf{s}, \mathbf{a}) - \alpha \log \pi(\mathbf{a}|\mathbf{s})]$ and α is a temperature parameter. The policy (or actor) π_ϕ is trained to minimize the KL divergence $\mathcal{L}_\pi = \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}} \left[D_{\text{KL}} \left(\pi_\phi(\mathbf{a}|\mathbf{s}) || \exp(Q_\theta(\mathbf{s}, \mathbf{a}))/Z_\theta(\mathbf{s})\right)\right]$ where Z_θ is the partition function that normalizes the distribution on the right. In the overall algorithm, the agent alternates between collecting data, and updating the actor and critic with this data.

B.2 Importance Weighting

Domain adaptation methods typically leverage importance weighting to correct the bias of samples from the source domain [49, 4]. When the tasks have different dynamics but same reward, prior work

has defined the importance weights for each transition sample as the likelihood ratio $w(\mathbf{s}, \mathbf{a}, \mathbf{s}') \coloneqq p^{\text{tgt}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})/p^{\text{src}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, 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 [42] or with classifiers in a likelihood-free manner [5, 7]. Eysenbach et al. [7] 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., $p^{\text{src}}(\mathbf{s}, \mathbf{a}) = p^{\text{tgt}}(\mathbf{s}, \mathbf{a})$. While DARC uses the same policy in the two domains, making this approximately hold, 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, because their 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, the lifelong 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.

C Hyperparameter Details

Policy and critic networks. For all experiments, we implement our algorithm on top of the soft actor-critic (SAC) [13] 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 [7], 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 threshold values $\zeta_1 = 0.5$ and $\zeta_2 = 0.9$ 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 $|\mathcal{D}^i|$ 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.

D Environment Setup

D.1 Simulated Experiments

In our simulated experiments, we use the Robosuite [51] simulation framework which employs the MuJoCo physics engine [43]. 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 values of these parameters are provided for each task in Table 2. The reward function across all tasks is:

$$\begin{aligned}
&\frac{1}{3}(\mathbb{1}(\|\mathbf{s}_{xy} - g_{xy}\|_2 \leq 0.03) \cdot \mathbb{1}(\mathbf{s}_z - l \leq g_{z,u} + 0.005) \\
&+ (1 - \tanh(\|10 \cdot (\mathbf{s}_{xy} - g_{xy}\|_2 + |\mathbf{s}_z - g_{z,l}|))) \\
&+ (1 - \tanh(|\mathbf{s}_\theta - g_\theta|))),
\end{aligned}$$

Task	Sequence 1			Sequence 2			Sequence 3		
	Rotation	Position	Key Type	Rotation	Position	Key Type	Rotation	Position	Key Type
1	0	[0, 0]	Standard	0	[0, 0]	Standard	0	[0, 0]	Standard
2	0	[−0.045, 0.003]	Horizontal	−0.065	[−0.003, 0.005]	Wide	−0.194	[0.004, −0.026]	Wide
3	0.100	[−0.032, −0.024]	Horizontal	−0.510	[−0.037, −0.028]	Standard	0.995	[0.026, 0.024]	Standard
4	−0.441	[0.029, 0.007]	Standard	−0.888	[0.016, 0.049]	Wide	0.979	[0.026, −0.001]	Wide
5	−0.076	[−0.034, −0.021]	Wide	−0.062	[0.043, −0.013]	Horizontal	0.061	[0.002, 0.039]	Horizontal
6	−0.056	[0.023, 0.017]	Horizontal	1.328	[−0.026, 0.021]	Standard	0.109	[0.040, −0.036]	Horizontal
7	1.177	[−0.024, 0.038]	Standard	0.864	[−0.003, −0.018]	Wide	0.282	[−0.025, 0.029]	Standard
8	1.492	[−0.038, 0.018]	Standard	0.598	[0.038, 0.040]	Wide	0.072	[−0.041, 0.038]	Horizontal
9	0.044	[0.045, −0.015]	Horizontal	0.001	[−0.035, 0.027]	Horizontal	0.586	[−0.007, −0.034]	Wide
10	1.521	[0.025, 0.024]	Standard	0.824	[−0.015, 0.039]	Wide	0.099	[−0.041, 0.020]	Horizontal
11	−0.029	[0.037, 0.005]	Horizontal	0.857	[0.010, −0.030]	Wide	0.091	[0.024, −0.027]	Standard
12	0.059	[−0.005, 0.001]	Horizontal	0.479	[−0.003, 0.011]	Wide	0.798	[0.048, 0.018]	Standard
13	−1.131	[−0.035, −0.021]	Standard	−0.580	[0.005, 0.008]	Standard	1.264	[−0.019, −0.043]	Standard
14	0.482	[−0.003, 0.031]	Horizontal	0.129	[0.049, 0.002]	Horizontal	0.030	[−0.023, −0.010]	Horizontal
15	0.340	[−0.038, 0.035]	Wide	0.904	[−0.032, 0.049]	Standard	−0.615	[−0.006, −0.008]	Standard
16	−0.841	[−0.028, −0.006]	Standard	−0.090	[0.033, 0.036]	Standard	−0.082	[−0.028, −0.009]	Horizontal

Table 2: Description for all 16 tasks for each of the 3 task sequences. Between tasks, we randomize the orientation of the key with respect to the box (in radians), position of the box, and the type of key.

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. 4, 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.

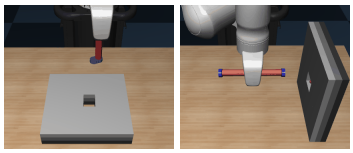


Figure 4: Left: an example of a vertical insertion task. Right: an example of a horizontal insertion task.

D.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 first task we assigned to 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

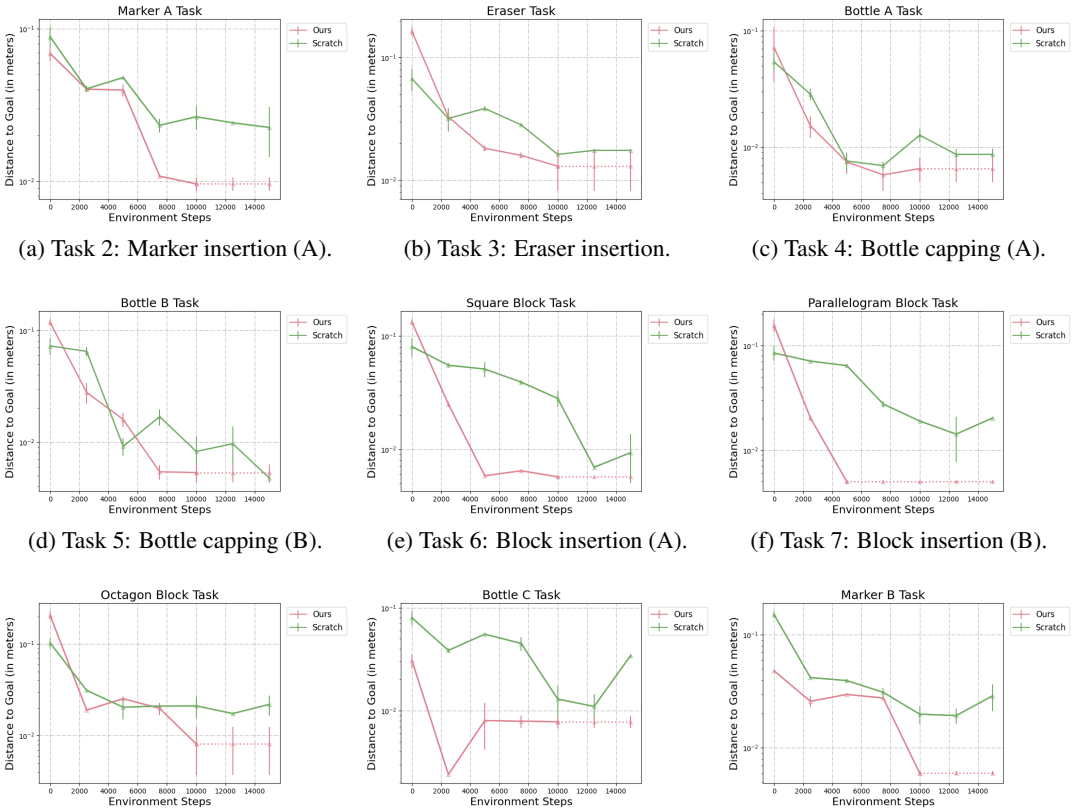
$$r^2(\mathbf{s}, \mathbf{a}) = 1 - \tanh(10 \cdot \|\mathbf{s}_{xyz} - g_{\text{marker-A}}\|_2).$$

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

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



(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 5: Individual learning curves for each task.

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:

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

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

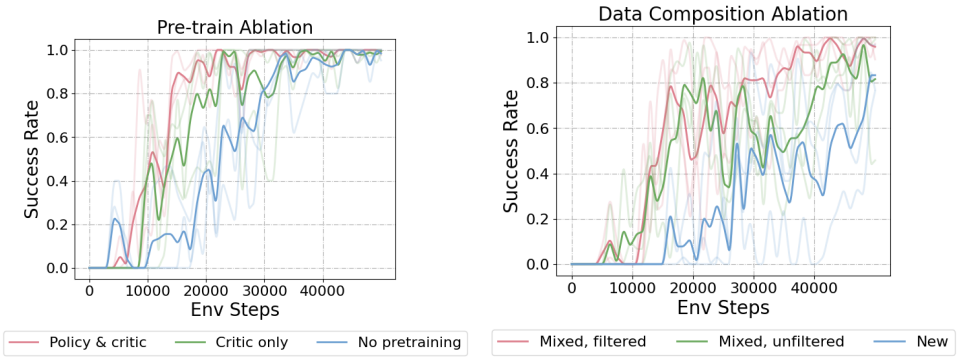
Task 6: Block insertion (A). The objective is to align the square block to the corresponding hole of the toy cube. We specify this objective through a goal position $g_{\text{block-A}} = [0.472, -0.002, 0.125]^T$ for the end-effector. The reward function for this task is

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

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 $g_{\text{block-C}} =$



(a) (b)

Figure 6: Ablations of the pre-training and online phases, evaluated on the same target task. The faint lines correspond to the 3 random seeds; the solid lines represent their average.

$[0.472, -0.060, 0.140]^T$ 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.

E Additional Experimental Results

E.1 Improved data efficiency

One desirable property in sequential learning is compounding learning, i.e., improving 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 3. 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 Tasks / Hor. Tasks	1 / 0	3 / 0	5 / 1	7 / 2
Ours	0.48	0.54	0.81	0.84

Table 3: Average task success after learning a varying number of tasks. The performance trends upwards with more tasks.

E.2 Ablations: Investigating Advantages of Prior Experience

The results from Sec. 3.1 demonstrate that the previously collected data samples, when appropriately leveraged, are a powerful form of knowledge transfer. To verify that our algorithm utilizes this prior experience more effectively than alternative design choices, we ablate the pre-training and online improvement stages of our framework. All ablations are evaluated on the same target task.

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 3 trials for each method, are presented in Fig. 6a, and suggest that pre-training both the policy and critic weights leads to slightly more efficient learning than only pre-training the critic. However, both pre-training schemes outperform the variant with no pre-training, demonstrating the importance of this initial step.

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) training on a mix of *unfiltered* prior and new experience and (2) training on new experience only. For each variant, we randomly initialize the policy and critic weights and do not perform the pre-training stage to study the effects of data composition alone. As shown in Fig. 6b, transferring the prior samples that are likely under the current dynamics leads to improved data efficiency.

E.3 Robot Experiment Results

In Fig. 5, 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. In Table 4, we report the average final performance by task for the two methods.

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

Table 4: 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.