Scalable Robot Learning

by

Ashvin Nair

A dissertation submitted in partial satisfaction

of the requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering & Computer Science

in the Graduate Division of the

University of California, Berkeley

Summer 2022

Committee:

Professor Sergey Levine, Chair
Professor Pieter Abbeel
Professor Alison Gopnik

ABSTRACT

ACKNOWLEDGMENTS

blah blah

CONTENTS

1

INTRODUCTION

TODO

Part I

REINFORCEMENT LEARNING WITH IMAGINED GOALS

2

VISUAL REINFORCEMENT LEARNING WITH IMAGINED GOALS

2.1 INTRODUCTION

Reinforcement learning (RL) algorithms hold the promise of allowing autonomous agents, such as robots, to learn to accomplish arbitrary tasks. However, the standard RL framework involves learning policies that are specific to individual tasks, which are defined by hand-specified reward functions. Agents that exist persistently in the world can prepare to solve diverse tasks by setting their own goals, practicing complex behaviors, and learning about the world around them. In fact, humans are very proficient at setting abstract goals for themselves, and evidence shows that this behavior is already present from early infancy (Smith and Gasser, 2005), albeit with simple goals such as reaching. The behavior and representation of goals grows more complex over time as they learn how to manipulate objects and locomote. How can we begin to devise a reinforcement learning system that sets its own goals and learns from experience with minimal outside intervention and manual engineering?

In this paper, we take a step toward this goal by designing an RL framework that jointly learns representations of raw sensory inputs and policies that achieve arbitrary goals under this representation by practicing to reach self-specified random goals during training. To provide for automated and flexible goal-setting, we must first choose how a general goal can be specified for an agent interacting with a complex and highly variable environment. Even providing the state of such an environment to a policy is a challenge. For instance, a task that requires a robot to manipulate various objects would require a combinatorial representation, reflecting variability in the number and type of objects in the current scene. Directly using raw sensory signals, such as images, avoids this challenge, but learning from raw images is substantially harder. In particular, pixel-wise Euclidean distance is not an effective reward function for visual tasks since distances between images do not correspond to meaningful distances between states (Ponomarenko

et al., 2015; Zhang et al., 2018). Furthermore, although end-to-end model-free reinforcement learning can handle image observations, this comes at a high cost in sample complexity, making it difficult to use in the real world.

We propose to address both challenges by incorporating unsupervised representation learning into goal-conditioned policies. In our method, which is illustrated in Figure 18, a representation of raw sensory inputs is learned by means of a latent variable model, which in our case is based on the variational autoencoder (VAE) (Kingma and Welling, 2014). This model serves three complementary purposes. First, it provides a more structured representation of sensory inputs for RL, making it feasible to learn from images even in the real world. Second, it allows for sampling of new states, which can be used to set synthetic goals during training to allow the goal-conditioned policy to practice diverse behaviors. We can also more efficiently utilize samples from the environment by relabeling synthetic goals in an off-policy RL algorithm, which makes our algorithm substantially more efficient. Third, the learned representation provides a space where distances are more meaningful than the original space of observations, and can therefore provide well-shaped reward functions for RL. By learning to reach random goals sampled from the latent variable model, the goal-conditioned policy learns about the world and can be used to achieve new, user-specified goals at test-time.

The main contribution of our work is a framework for learning general-purpose goal-conditioned policies that can achieve goals specified with target observations. We call our method reinforcement learning with imagined goals (RIG). RIG combines sample-efficient off-policy goal-conditioned reinforcement learning with unsupervised representation learning. We use representation learning to acquire a latent distribution that can be used to sample goals for unsupervised practice and data augmentation, to provide a well-shaped distance function for reinforcement learning, and to provide a more structured representation for the value function and policy. While several prior methods, discussed in the following section, have sought to learn goal-conditioned policies, we can do so with image goals and observations without a manually specified reward signal. Our experimental evaluation illustrates that our method substantially improves the performance of image-based reinforcement learning, can effectively learn policies for complex image-based tasks, and can be used to learn real-world robotic manipulation skills with raw image inputs. Videos of our method in simulated and real-world environments can be found at https://sites.google.com/site/visualrlwithimaginedgoals/.

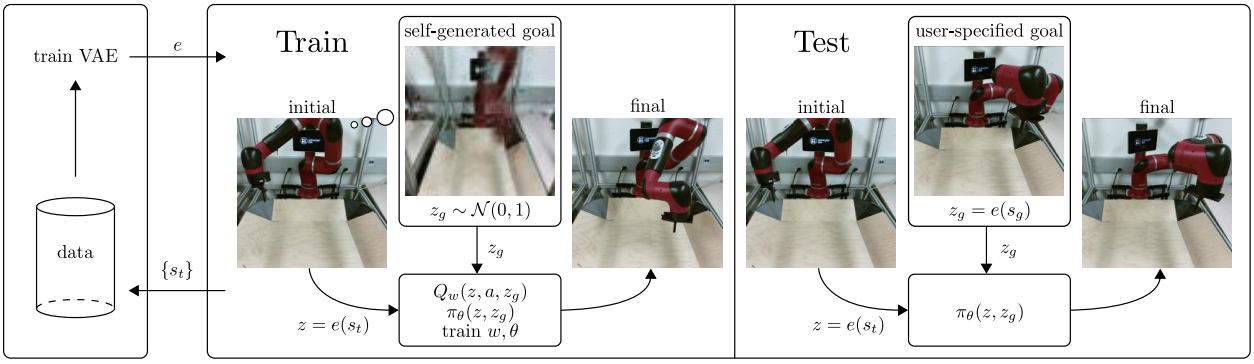


Figure 1: We train a VAE using data generated by our exploration policy (left). We use the VAE for multiple purposes during training time (middle): to sample goals to train the policy, to embed the observations into a latent space, and to compute distances in the latent space. During test time (right), we embed a specified goal observation o_g into a goal latent z_g as input to the policy.

2.2 RELATED WORK

While prior works on vision-based deep reinforcement learning for robotics can efficiently learn a variety of behaviors such as grasping (Pinto et al., 2017; Pinto and Gupta, 2016; Levine et al., 2017), pushing (Agrawal et al., 2016; Ebert et al., 2017; Finn and Levine, 2016), navigation (Pathak et al., 2018; Lange et al., 2012), and other manipulation tasks (Lillicrap et al., 2016; Levine et al., 2016; Pathak et al., 2018), they each make assumptions that limit their applicability to training general-purpose robots. Levine et al. (2016) uses time-varying models, which requires an episodic setup that makes them difficult to extend to non-episodic and continual learning scenarios. Pinto et al. (2017) proposed a similar approach that uses goal images, but requires instrumented training in simulation. Lillicrap et al. (2016) uses fully model-free training, but does not learn goal-conditioned skills. As we show in our experiments, this approach is very difficult to extend to the goal-conditioned setting with image inputs. Model-based methods that predict images (Watter et al., 2015; Finn and Levine, 2016; Ebert et al., 2017; Oh et al., 2015) or learn inverse models (Agrawal et al., 2016) can also accommodate various goals, but tend to limit the horizon length due to model drift. To our knowledge, no prior method uses model-free RL to learn policies conditioned on a single goal image with sufficient efficiency to train directly on real-world robotic systems, without access to ground-truth state or reward information during training.

Our method uses a goal-conditioned value function (Schaul et al., 2015) in order to solve more general tasks (Sutton et al., 2011; Kaelbling, 1993). To improve the sample-

efficiency of our method during off-policy training, we retroactively relabel samples in the replay buffer with goals sampled from the latent representation. Goal relabeling has been explored in prior work (Kaelbling, 1993; Andrychowicz et al., 2017; Rauber et al., 2017; Levy et al., 2017; Pong et al., 2018). Andrychowicz et al. (2017) and Levy et al. (2017) use goal relabeling for sparse rewards problems with known goal spaces, restricting the resampled goals to states encountered along that trajectory, since almost any other goal will have no reward signal. We sample random goals from our learned latent space to use as replay goals for off-policy Q-learning rather than restricting ourselves to states seen along the sampled trajectory, enabling substantially more efficient learning. We use the same goal sampling mechanism for exploration in RL. Goal setting for policy learning has previously been discussed (Baranes and Oudeyer, 2012) and recently Péré et al. (2018) have also proposed using unsupervised learning for setting goals for exploration. However, we use a model-free Q-learning method that operates on raw state observations and actions, allowing us to solve visually and dynamically complex tasks.

A number of prior works have used unsupervised learning to acquire better representations for RL. These methods use the learned representation as a substitute for the state for the policy, but require additional information, such as access to the ground truth reward function based on the true state during training time (Higgins et al., 2017b; Ha and Schmidhuber, 2018; Watter et al., 2015; Finn et al., 2016; Lange et al., 2012; Jonschkowski et al., 2017), expert trajectories (Srinivas et al., 2018), human demonstrations (Sermanet et al., 2017), or pre-trained object-detection features (Lee et al., 2017). In contrast, we learn to generate goals and use the learned representation to obtain a reward function for those goals without any of these extra sources of supervision. Finn et al. (2016) combine unsupervised representation learning with reinforcement learning, but in a framework that trains a policy to reach a single goal. Many prior works have also focused on learning controllable and disentangled representations (Schmidhuber, 1992; Chen et al., 2016; Cheung et al., 2014; Reed et al., 2014; Desjardins et al., 2012; Thomas et al., 2017). We use a method based on variational autoencoders, but these prior techniques are complementary to ours and could be incorporated into our method.

2.3 BACKGROUND

Our method combines reinforcement learning with goal-conditioned value functions and unsupervised representation learning. Here, we briefly review the techniques that we build on in our method.

GOAL-CONDITIONED REINFORCEMENT LEARNING. In reinforcement learning, the goal is to learn a policy $\pi(s_t) = a_t$ that maximizes expected return, which we denote as $R_t = \mathbb{E}[\sum_{i=t}^{T} \gamma^{(i-t)} r_i]$, where $r_i = r(s_i, a_i, s_{i+1})$ and the expectation is under the current policy and environment dynamics. Here, $s \in \mathcal{S}$ is a state observation, $a \in \mathcal{A}$ is an action, and γ is a discount factor. Standard model-free RL learns policies that achieve a single task. If our aim is instead to obtain a policy that can accomplish a variety of tasks, we can construct a goal-conditioned policy and reward, and optimize the expected return with respect to a goal distribution: $\mathbb{E}_{g\sim G}[\mathbb{E}_{r_i,s_i\sim E,a_i\sim\pi}[R_0]]$, where $\mathcal{G}$ is the set of goals and the reward is also a function of g. A variety of algorithms can learn goal-conditioned policies, but to enable sample-efficient learning, we focus on algorithms that acquire goal-conditioned Q-functions, which can be trained off-policy. A goal-conditioned Q-function $Q(s, a, g)$ learns the expected return for the goal g starting from state s and taking action a. Given a state s, action a, next state s', goal g, and correspond reward r, one can train an approximate Q-function parameterized by w by minimizing the following Bellman error

$$\mathcal{E}(w) = \frac{1}{2}||Q_w(s, a, g) - (r + \gamma \max_{a'} Q_{\bar{w}}(s', a', g))||^2 \tag{1}$$

where $\bar{w}$ indicates that $\bar{w}$ is treated as a constant. Crucially, one can optimize this loss using off-policy data (s, a, s', g, r) with a standard actor-critic algorithm (Lillicrap et al., 2016; Fujimoto et al., 2018; Mnih et al., 2016).

VARIATIONAL AUTOENCODERS. Variational autoencoders (VAEs) have been demonstrated to learn structured latent representations of high dimensional data (Kingma and Welling, 2014). The VAE consists of an encoder p_ϕ, which maps states to latent distributions, and a decoder p_ψ, which maps latents to distributions over states. The encoder and decoder parameters, ψ and ϕ respectively, are jointly trained to maximize

$$\mathcal{L}(\psi, \phi; s^{(i)}) = -\beta D_{KL}(q_\phi(z|s^{(i)})||p(z)) + \mathbb{E}_{q_\phi(z|s^{(i)})}[\log p_\psi(s^{(i)} \mid z)], \tag{2}$$

where $p(z)$ is some prior, which we take to be the unit Gaussian, D_{KL} is the Kullback-Leibler divergence, and β is a hyperparameter that balances the two terms. The use of β values other than one is sometimes referred to as a β-VAE (Higgins et al., 2017a). The encoder q_ϕ parameterizes the mean and log-variance diagonal of a Gaussian distribution, $q_\phi(s) = \mathcal{N}(\mu_\phi(s), \sigma^2_\phi(s))$. The decoder p_ψ parameterizes a Bernoulli distribution for each pixel value. This parameterization corresponds to training the decoder with cross-entropy loss on normalized pixel values. Full details of the hyperparameters are in the Supplementary Material.

2.4 GOAL-CONDITIONED POLICIES WITH UNSUPERVISED REPRESENTATION LEARNING

To devise a practical algorithm based on goal-conditioned value functions, we must choose a suitable goal representation. In the absence of domain knowledge and instrumentation, a general-purpose choice is to set the goal space $\mathcal{G}$ to be the same as the state observations space $\mathcal{S}$. This choice is fully general as it can be applied to any task, and still permits considerable user control since the user can choose a "goal state" to set a desired goal for a trained goal-conditioned policy. But when the state space $\mathcal{S}$ corresponds to high-dimensional sensory inputs such as images, [1] learning a goal-conditioned Q-function and policy becomes exceedingly difficult as we illustrate empirically in Section 7.6.

Our method jointly addresses a number of problems that arise when working with high-dimensional inputs such as images: sample efficient learning, reward specification, and automated goal-setting. We address these problems by learning a latent embedding using a β-VAE. We use this latent space to represent the goal and state and retroactively relabel data with latent goals sampled from the VAE prior to improve sample efficiency. We also show that distances in the latent space give us a well-shaped reward function for images. Lastly, we sample from the prior to allow an agent to set and "practice" reaching its own goal, removing the need for humans to specify new goals during training time. We next describe the specific components of our method, and summarize our complete algorithm in Section 2.4.5.

2.4.1 *Sample-Efficient RL with Learned Representations*

One challenging problem with end-to-end approaches for visual RL tasks is that the resulting policy needs to learn both perception and control. Rather than operating directly on observations, we embed the state s_t and goals g into a latent space $\mathcal{Z}$ using an encoder e to obtain a latent state $z_t = e(s_t)$ and latent goal $z_g = e(g)$. To learn a representation of the state and goal space, we train a β-VAE by executing a random policy and collecting state observations, $\{s^{(i)}\}$, and optimize Equation $equation$ 2. We then use the mean of the encoder as the state encoding, i.e. $z = e(s) \triangleq \mu_\phi(s)$.

After training the VAE, we train a goal-conditioned Q-function $Q(z, a, z_g)$ and corresponding policy $\pi_\theta(z, z_g)$ in this latent space. The policy is trained to reach a goal

1 We make the simplifying assumption that the system is Markovian with respect to the sensory input, and one could incorporate memory into the state for partially observed tasks.

z_g using the reward function discussed in Section 2.4.2. For the underlying RL algorithm, we use twin delayed deep deterministic policy gradients (TD3) (Fujimoto et al., 2018), though any value-based RL algorithm could be used. Note that the policy (and Q-function) operates completely in the latent space. During test time, to reach a specific goal state g, we encode the goal $z_g = e(g)$ and input this latent goal to the policy.

As the policy improves, it may visit parts of the state space that the VAE was never trained on, resulting in arbitrary encodings that may not make learning easier. Therefore, in addition to procedure described above, we fine-tune the VAE using both the randomly generated state observations $\{s^{(i)}\}$ and the state observations collected during exploration. We show in Section **??** that this additional training helps the performance of the algorithm.

2.4.2 *Reward Specification*

Training the goal-conditioned value function requires defining a goal-conditioned reward $r(s, g)$. Using Euclidean distances in the space of image pixels provides a poor metric, since similar configurations in the world can be extremely different in image space. In addition to compactly representing high-dimensional observations, we can utilize our representation to obtain a reward function based on a metric that better reflects the similarity between the state and the goal. One choice for such a reward is to use the negative Mahalanobis distance in the latent space:

$$r(s, g) = -\|e(s) - e(g)\|_A = -\|z - z_g\|_A,$$

where the matrix A weights different dimensions in the latent space. This approach has an appealing interpretation when we set A to be the precision matrix of the VAE encoder, q_ϕ. Since we use a Gaussian encoder, we have that

$$r(s, g) = -\|z - z_g\|_A \propto \sqrt{\log e_\phi(z_g \mid s)} \tag{3}$$

In other words, minimizing this squared distance in the latent space is equivalent to rewarding reaching states that maximize the probability of the latent goal z_g. In practice, we found that setting $A = \mathbf{I}$, corresponding to Euclidean distance, performed better than Mahalanobis distance, though its effect is the same — to bring z close to z_g and maximize the probability of the latent goal z_g given the observation. This interpretation would not be possible when using normal autoencoders since distances are not trained to have any probabilistic meaning. Indeed, we show in Section 7.6 that using distances in a normal

autoencoder representation often does not result in meaningful behavior.

2.4.3 *Improving Sample Efficiency with Latent Goal Relabeling*

To further enable sample-efficient learning in the real world, we use the VAE to relabel goals. Note that we can optimize Equation equation 31 using any valid (s, a, s', g, r) tuple. If we could artificially generate these tuples, then we could train our entire RL algorithm without collecting any data. However, we do not know the system dynamics, and therefore have to sample transitions (s, a, s') by interacting with the world. However, we have the freedom to relabel the goal and reward synthetically. In particular, if we have a mechanism for generating goals and computing rewards, then given (s, a, s'), we can generate a new goal g and new reward $r(s, a, s', g)$ to produce a new tuple (s, a, s', g, r). By artificially generating and recomputing rewards, we can convert a single (s, a, s') transition into potentially infinitely many valid training datums.

For image-based tasks, this procedure would require generating goal images, an onerous task on its own. However, our reinforcement learning algorithm operates directly in the latent space for goals and rewards. So rather than generating goals g, we generate latent goals z_g by sampling from the VAE prior $p(z)$. We then recompute rewards using Equation equation 3. By retroactively relabeling the goals and rewards, we obtain much more data to train our value function. This sampling procedure is made possible by our use of a latent variable model, which is explicitly trained so that sampling from the latent distribution is straightforward.

Retroactively generating goals is also explored in tabular domains by Kaelbling (1993) and in continuous domains by Andrychowicz et al. (2017) using hindsight experience replay (HER). However, HER is limited to sampling goals seen along a trajectory, which greatly limits the number and diversity of goals with which one can relabel a given transition. Our final method uses a mixture of the two strategies: half of the goals are generated from the prior and half from goals seen along the trajectory. We show in Section 7.6 that relabeling the goal with samples from the VAE prior results in significantly better sample-efficiency.

2.4.4 *Automated Goal-Generation for Exploration*

If we do not know which particular goals will be provided at test time, we would like our RL agent to carry out a self-supervised "practice" phase during training, where the algorithm proposes its own goals, and then practices how to reach them. Since the VAE

Algorithm 1 RIG: Reinforcement learning with imagined goals

Require: VAE encoder q_ϕ, VAE decoder p_ψ, policy π_θ, goal-conditioned value function Q_w.
Collect $\mathcal{D} = \{s^{(i)}\}$ using exploration policy.
Train β-VAE on $\mathcal{D}$ by optimizing equation 2.
for $n = 0, ..., N-1$ episodes **do**
  Sample latent goal from prior $z_g \sim p(z)$.
  Sample initial state $s_0 \sim E$.
  for $t = 0, ..., H-1$ steps **do**
    Get action $a_t = \pi_\theta(e(s_t), z_g) + \text{noise}$.
    Get next state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.
    Store (s_t, a_t, s_{t+1}, z_g) into replay buffer $\mathcal{R}$.
    Sample transition $(s, a, s', z_g) \sim \mathcal{R}$.
    Encode $z' = e(s')$.
    (Probability 0.5) replace z_g with $z'_g \sim p(z)$.
    Compute new reward $r = -\|z' - z_g\|$.
    Minimize equation 31 using (z, a, z', z_g, r).
  end for
  Fine-tune β-VAE every K episodes on mixture of $\mathcal{D}$ and $\mathcal{R}$.
end for=0

prior represents a distribution over latent goals and state observations, we again sample from this distribution to obtain plausible goals. After sampling a goal latent from the prior $z_g \sim p(z)$, we give this to our policy $\pi(z, z_g)$ to collect data.

2.4.5 *Algorithm Summary*

We call the complete algorithm reinforcement learning with imagined goals (RIG) and summarize it in Algorithm 2. We first collect data with a simple exploration policy, though any exploration strategy could be used for this stage, including off-the-shelf exploration bonuses (Pathak et al., 2017; Bellemare et al., 2016) or unsupervised reinforcement learning methods (Eysenbach et al., 2018; Florensa et al., 2017). Then, we train a VAE latent variable model on state observations and finetune it over the course of training. We use this latent variable model for multiple purposes: We sample a latent goal z_g from the model and condition the policy on this goal. We embed all states and goals using the model's encoder. When we train our goal-conditioned value function, we resample goals from the prior and compute rewards in the latent space using Equation equation 3. Any RL algorithm that trains Q-functions could be used, and we use TD3 (Fujimoto et al., 2018) in our implementation.

2.5 EXPERIMENTS

Our experiments address the following questions:

1. How does our method compare to prior model-free RL algorithms in terms of sample efficiency and performance, when learning continuous control tasks from

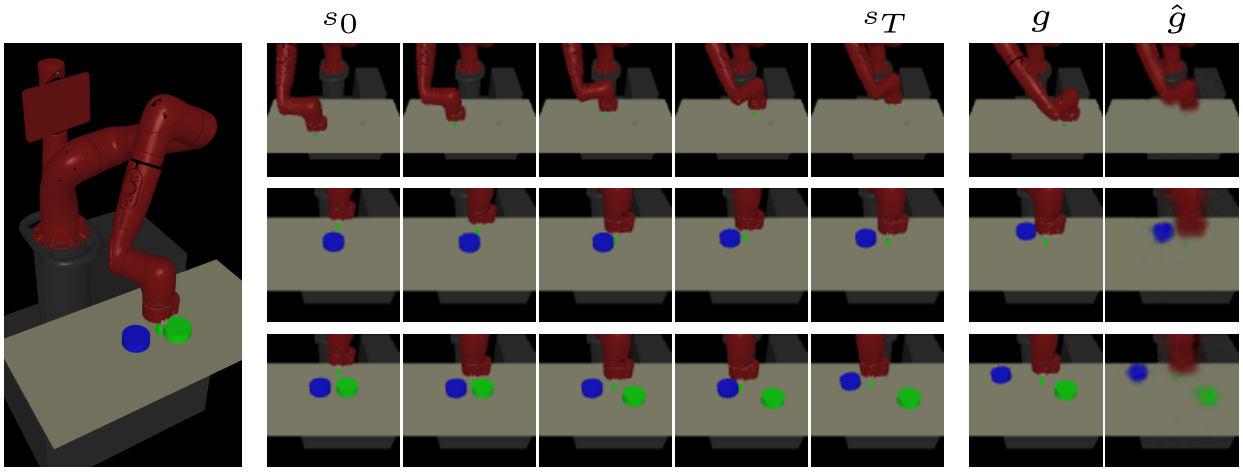


Figure 2: (Left) The simulated environment. (Right) Test rollouts from our learned policy on the three simulated environments. Each row is one rollout. The middle shows the goal image g and its VAE reconstruction $\hat{g}$. The right columns shows frames from the trajectory to reach the given goal.

images?

2. How critical is each component of our algorithm for efficient learning?
3. Does our method work on tasks where the state space cannot be easily specified ahead of time, such as tasks that require interaction with variable numbers of objects?
4. Can our method scale to real world vision-based robotic control tasks?

For the first two questions, we evaluate our method against a number of prior algorithms and ablated versions of our approach on a suite of simulated and real-world tasks: *Visual Reacher*: a MuJoCo (Todorov et al., 2012) environment with a 7-dof Sawyer arm reaching goal positions. The arm is shown the left of Figure 2. The end-effector (EE) is constrained to a 2-dimensional rectangle parallel to a table. The action controls EE velocity within a maximum velocity. *Visual Pusher*: a MuJoCo environment with a 7-dof Sawyer arm and a small puck on a table that the arm must push to a target push. *Visual Multi-Object Pusher*: a copy of the Visual Pusher environment with two pucks. Detailed descriptions of the environments are provided in the Supplementary Material.

Solving these tasks directly from images poses a challenge since the controller must learn both perception and control. The evaluation metric is the distance of objects (including the arm) to their respective goals. To evaluate our policy, we set the environment to a sampled goal position, capture an image, and encode the image to use as the goal. Although we use the ground-truth positions for evaluation, **we do not use the ground-**

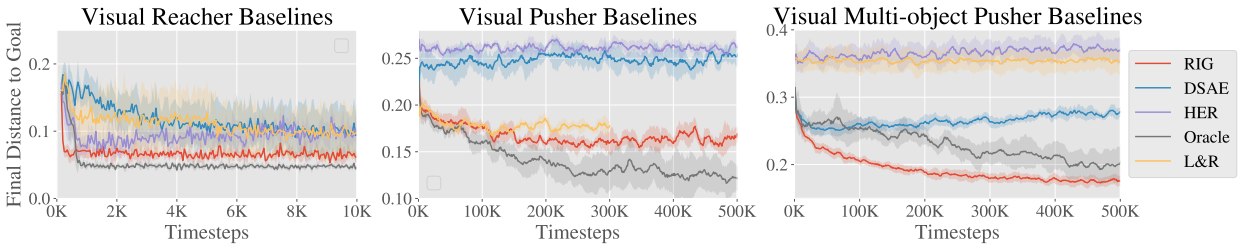


Figure 3: Simulation results, final distance to goal vs simulation steps[2]. RIG (red) consistently outperforms the baselines, except for the oracle which uses ground truth object state for observations and rewards. On the hardest task, only our method and the oracle discover viable solutions.

truth positions for training the policies. The only inputs from the environment that our algorithm receives are the image observations. For Visual Reacher, we pretrained the VAE with 100 images. For other tasks, we used 10,000 images.

We compare our method with the following prior works. *L&R*: Lange and Riedmiller (Lange and Riedmiller, 2010) trains an autoencoder to handle images. *DSAE*: Deep spatial autoencoders (Finn et al., 2016) learns a spatial autoencoder and uses guided policy search (Levine et al., 2016) to achieve a single goal image. *HER*: Hindsight experience replay (Andrychowicz et al., 2017) utilizes a sparse reward signal and relabeling trajectories with achieved goals. *Oracle*: RL with direct access to state information for observations and rewards.

To our knowledge, no prior work demonstrates policies that can reach a variety of goal images without access to a true-state reward function, and so we needed to make modifications to make the comparisons feasible. L&R assumes a reward function from the environment. Since we have no state-based reward function, we specify the reward function as distance in the autoencoder latent space. HER does not embed inputs into a latent space but instead operates directly on the input, so we use pixel-wise mean squared error (MSE) as the metric. DSAE is trained only for a single goal, so we allow the method to generalize to a variety of test goal images by using a goal-conditioned Q-function. To make the implementations comparable, we use the same off-policy algorithm to train L&R, HER, and our method (TD3 (Fujimoto et al., 2018)). Unlike our method, prior methods do not specify how to select goals during training, so we favorably give them real images as goals for rollouts, sampled from the same distribution that we use to test.

We see in Figure 3 that our method can efficiently learn policies from visual inputs to perform simulated reaching and pushing, without access to the object state. Our approach substantially outperforms the prior methods, for which the use of image goals

2 In all our simulation results, each plot shows a 95% confidence interval of the mean across 5 seeds.

and observations poses a major challenge. HER struggles because pixel-wise MSE is hard to optimize. Our latent-space rewards are much better shaped and allow us to learn more complex tasks. Finally, our method is close to the state-based "oracle" method in terms of sample efficiency and performance, without having any access to object state. Notably, in the multi-object environment, our method actually outperforms the oracle, likely because the state-based reward contains local minima. Overall, these result show that our method is capable of handling raw image observations much more effectively than previously proposed goal-conditioned RL methods. Next, we perform ablations to evaluate our contributions in isolation. Results on Visual Pusher are shown but see the Supplementary Material (section **??**) for experiments on all three simulated environments.

REWARD SPECIFICATION COMPARISON We evaluate how effective distances in the VAE latent space are for the Visual Pusher task. We keep our method the same, and only change the reward function that we use to train the goal-conditioned valued function. We include the following methods for comparison: *Latent Distance*, which is the reward used in RIG, i.e. $A = \mathbf{I}$ in Equation equation 3; *Log Probability*, which uses the Mahalanobis distance in Equation equation 3, where A is the precision matrix of the encoder; and *Pixel MSE*, which computes mean-squared error (MSE) between state and goal in pixel space. [3] In Figure 4, we see that latent distance significantly outperforms the log probability. We suspect that small variances of the VAE encoder results in drastically large rewards, making the learning more difficult. We also see that latent distances results in faster learning when compared to pixel MSE.

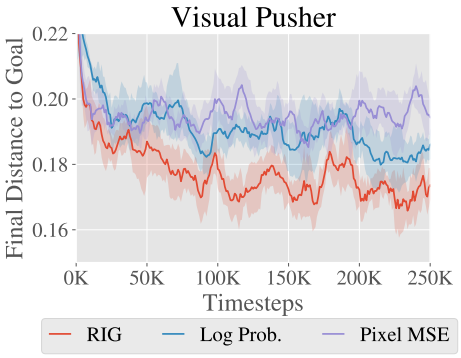


Figure 4: Reward type ablation results. RIG (red), which uses latent Euclidean distance, outperforms the other methods.

3 To compute the pixel MSE for a sampled latent goal, we decode the goal latent using the VAE decoder, p_ψ, to generate the corresponding goal image.

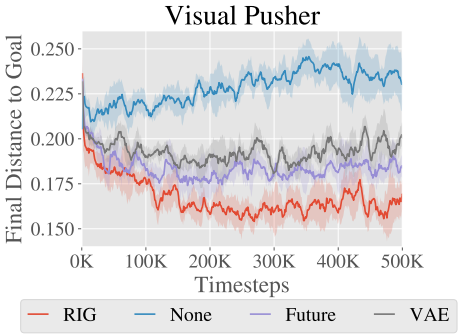


Figure 5: Relabeling ablation results. RIG (red), which uses a mixture of VAE and HER, outperforms the other methods.

RELABELING STRATEGY COMPARISON As described in section 2.4.3, our method uses a novel goal relabeling method based on sampling from the generative model. To isolate how much our new goal relabeling method contributes to our algorithm, we vary the resampling strategy while fixing other components of our algorithm. The resampling strategies that we consider are: *Future*, relabeling the goal for a transition by sampling uniformly from future states in the trajectory as done in Andrychowicz et al. (2017); *VAE*, sampling goals from the VAE only; *RIG*, relabeling goals with probability 0.5 from the VAE and probability 0.5 using the future strategy; and *None*, no relabeling. In Figure 5, we see that sampling from the VAE and Future is significantly better than not relabeling at all. In RIG, we use an equal mixture of the VAE and Future sampling strategies, which performs best by a large margin. Appendix section **??** contains results on all simulated environments, and section **??** considers relabeling strategies with a known goal distribution.

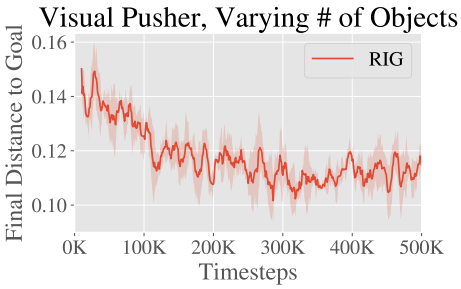


Figure 6: Training curve for learning with varying number of objects.

LEARNING WITH VARIABLE NUMBERS OF OBJECTS A major advantage of working directly from pixels is that the policy input can easily represent combinatorial structure in the environment, which would be difficult to encode into a fixed-length state vector even if a perfect perception system were available. For example, if a robot has to interact with different combinations and numbers of objects, picking a single MDP state representation would be challenging, even with access to object poses. By directly processing images for both the state and the goal, no modification is needed to handle the combinatorial structure: the number of pixels always remains the same, regardless of how many objects are in the scene.

We demonstrate that our method can handle this difficult scenario by evaluating on a task where the environment, based on the Visual Multi-Object Pusher, randomly contains zero, one, or two objects in each episode during testing. During training, each episode still always starts with both objects in the scene, so the experiments tests whether a trained policy can handle variable numbers of objects at test time. Figure 6 shows that

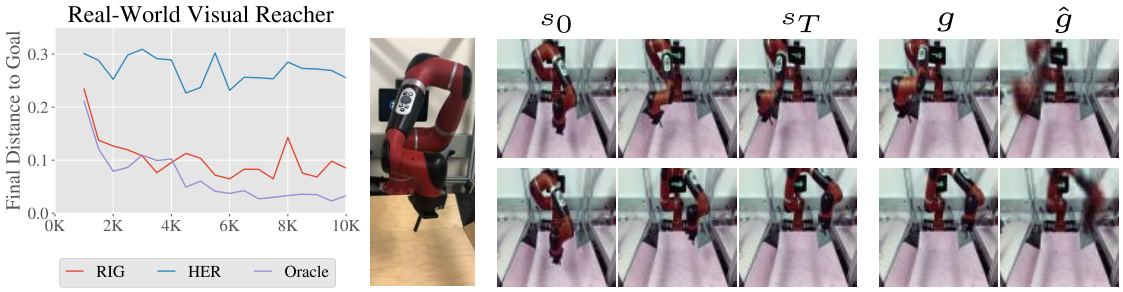
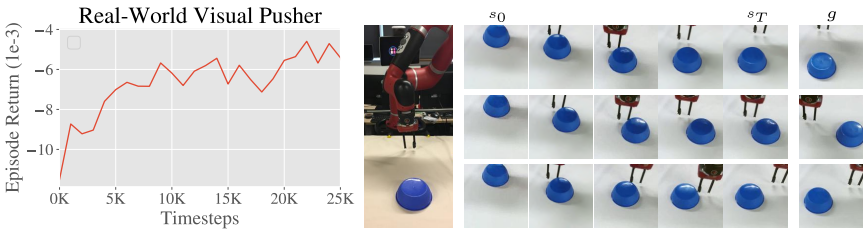


Figure 7: (Left) Our method compared to the HER baseline and oracle on a real-world visual reaching task. (Middle) Our robot setup is pictured. (Right) Test rollouts of our learned policy.



Puck Distance to Goal (cm)

RIG	HER
4.5 ± 2.5	14.9 ± 5.4

Figure 8: (Left) The learning curve for real-world pushing. (Middle) Our robot pushing setup is pictured, with frames from test rollouts of our learned policy. (Right) Our method compared to the HER baseline on the real-world visual pushing task. We evaluated the performance of each method by manually measuring the distance between the goal position of the puck and final position of the puck for 15 test rollouts, reporting mean and standard deviation.

our method can learn to solve this task successfully, without decrease in performance from the base setting where both objects are present (in Figure 3). Developing and demonstrating algorithms that solve tasks with varied underlying structure is an important step toward creating autonomous agents that can handle the diversity of tasks present "in the wild."

2.5.1 *Visual RL with Physical Robots*

RIG is a practical and straightforward algorithm to apply to real physical systems: the efficiency of off-policy learning with goal relabeling makes training times manageable, while the use of image-based rewards through the learned representation frees us from the burden of manually design reward functions, which itself can require hand-engineered perception systems (Rusu et al., 2017). We trained policies for visual reaching

and pushing on a real-world Sawyer robotic arm, shown in Figure 7. The control setup matches Visual Reacher and Visual Pusher respectively, meaning that **the only input from the environment consists of camera images**.

We see in Figure 7 that our method is applicable to real-world robotic tasks, almost matching the state-based oracle method and far exceeding the baseline method on the reaching task. Our method needs just 10,000 samples or about an hour of real-world interaction time to solve visual reaching.

Real-world pushing results are shown in Figure 12. To solve visual pusher, which is more visually complicated and requires reasoning about the contact between the arm and object, our method requires about 25,000 samples, which is still a reasonable amount of real-world training time. Note that unlike previous results, we do not have access to the true puck position during training so for the learning curve we report test episode returns on the VAE latent distance reward. We see RIG making steady progress at optimizing the latent distance as learning proceeds.

2.6 Discussion and Future Work

In this paper, we present a new RL algorithm that can efficiently solve goal-conditioned, vision-based tasks without access to any ground truth state or reward functions. Our method trains a generative model that is used for multiple purposes: we embed the state and goals using the encoder; we sample from the prior to generate goals for exploration; we also sample latents to retroactively relabel goals and rewards; and we use distances in the latent space for rewards to train a goal-conditioned value function. We show that these components culminate in a sample efficient algorithm that works directly from vision. As a result, we are able to apply our method to a variety of simulated visual tasks, including a variable-object task that cannot be easily represented with a fixed length vector, as well as real world robotic tasks. Algorithms that can learn in the real world and directly use raw images can allow a single policy to solve a large and diverse set of tasks, even when these tasks require distinct internal representations.

The method we presented can be extended in a number of ways. First, an exciting line of future work would be to combine our method with existing work on exploration and intrinsic motivation. In particular, our method already provides a natural mechanism for autonomously generating goals by sampling from the prior. Modifying this procedure to not only be goal-oriented but also, e.g., be information seeking or uncertainty aware could provide better and safer exploration. Second, since our method operates directly from images, a single policy could potentially solve a large diverse set of visual

tasks, even if those tasks have different underlying state representations. Combining these ideas with methods from multitask learning and meta-learning is a promising path to creating general-purpose agents that can continuously and efficiently acquire skills. Lastly, while RIG uses goal images, extending the method to allow goals specified by demonstrations or more abstract representations such as language would enable our system to be much more flexible in interfacing with humans and therefore more practical.

3

CONTEXTUAL IMAGINED GOALS FOR SELF-SUPERVISED ROBOTIC LEARNING

3.1 INTRODUCTION

In order for robots to truly become *generalists*, they must be readily taskable by humans, handle raw sensory inputs without instrumentation, and be equipped with a range of skills that generalize effectively to new situations. Reinforcement learning autonomously learns policies that maximize a reward function and is a promising approach towards such generalist robots. However, in a general setting involving diverse objects and tasks, prior information about what tasks to learn is hard to come by without manually designing object detectors and reward functions. How can a robot explore the world in order to learn and fine-tune useful skills on diverse objects, only from acting and observing how its actions affect its sensory stream?

Prior methods have proposed to let an agent learn from its sensor stream by automatically generating plausible goals during an unsupervised training phase, and then learning policies that reach those goals **nair2018rig**; **nachum2018hiro**; **wadefarley2019discern**; **pong2019skewfit**. Such goals can be defined in a variety of ways, but a simple choice is to use goal observations, such that each proposed task requires reaching a different observation. When the robot observes the world via raw camera images, this corresponds to using images as goals. At test time, a user then provides the robot with a new goal image.

While such methods have been demonstrated in both simulated and real-world settings, they are typically used to learn behaviors in domains with relatively little visual diversity. In the real world, a robot might interact with highly diverse scenes and objects, and the tasks that it can perform from each of the many possible initial states will be different. If the robot is presented with an object, it can learn to pick up or grasp it, and when it is presented with a door, it can learn to open it. However, it must generate and

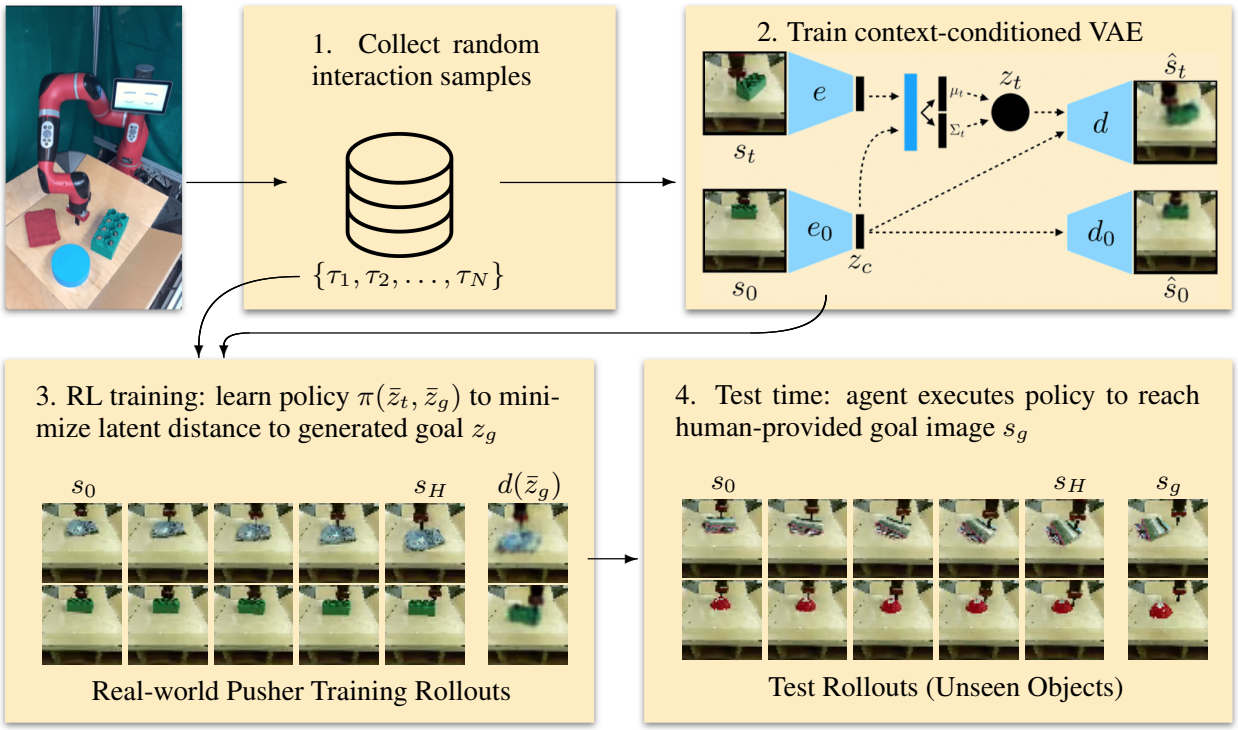


Figure 9: System overview of our self-supervised learning algorithm. (1) The agent collects random interaction data, to be used for both representation learning and as additional off-policy data for RL. (2) We propose a context-conditioned generative model (CC-VAE) to learn generalizable skills. In order to improve the generation of plausible goal states that result from a starting state s_0, our model allows information to flow freely through the context z_c while an information bottleneck on z_t gives us the ability to generate samples by re-sampling only z_t. This architecture provides a compact representation of the scene disentangling information that changes within a rollout (z_t) and information that changes between rollouts (z_c). We then use $\bar{z}_t = (z_t, z_c)$ as the representation for RL. (3) Our proposed CC-RIG algorithm samples latent goals, using the above representation, and learns a policy to minimize the latent distance to the goal with off-policy RL. Rollouts are shown on the real-world Sawyer robot pusher environment with visual variation. We include the initial image s_0, selected frames from the rollout, final image s_H, and the decoded goal latent $d(\bar{z}_g)$. (4) At test time, the agent is given a goal image s_g and executes the policy to reach it. Our method successfully handles pushing novel objects that were unseen at training time. Example rollouts can be found at https://ccrig.github.io/

practice goals that are suitable for each scene. In this paper, we propose and evaluate a self-supervised policy learning method that learns to propose goals that are suitable to the current scene via a conditional goal generation model, allowing it to learn in visually varied settings that prove challenging for prior algorithms.

The key idea in our work is that representing every element in a visually complex scene is often not necessary for control. A scene is a visual form of context that can be factored out, while only the controllable entities in the environment need to be captured for goal setting and representing the state. To this end, we propose learning a context-conditioned generative model that learns a smooth, compressed latent variable with an information bottleneck, while allowing the context, in the form of the initial state image, to be used freely to reconstruct other images during the task. This context-conditioned generative model architecture is shown in Figure 18.

The main contribution in this paper builds on this context-conditioned generative model to devise a complete self-supervised goal-conditioned reinforcement learning algorithm, which can handle visual variability in the scene via context-conditioned goal setting. Our method can learn policies that reach visually indicated goals without any additional supervision during training, using the context-conditioned generative model to set goals that are appropriate to the current scene. We show that our approach learns coherent representations of visually varied environments, capturing controllable dimensions of variation while ignoring dimensions that vary but cannot be influenced by the agent, such as lighting and object appearance. We further show that our approach can learn policies to solve tasks in visually varied environments, including in a real-world robotic pushing task with a wide variety of distinct objects in Figure 12.

3.2 RELATED WORK

While many practical robots today perform tasks by executing hand-engineered sequences of motor commands, machine learning is opening up a new avenue to train a wide variety of robotic tasks from interaction. This body of work includes grasping **ekvall2004interactive**; **kroemer2010grasping**; **bohg2010learning**, and general tasks **PETERS2008682**; **kober2013reinforceme** multi-task learning **6907421**, baseball **peters2008baseball**, ping-pong **peters2010reps**, and various other tasks **deisenroth2011pilco**. More recently, using expressive function approximators such as neural networks has reduced manual feature engineering and has increased task complexity and diversity, finding use in decision-making domains, such as solving Atari games **mnih2013atari** and Go **silver2016alphago**. Deep learning for robotics has proved to be difficult due to a host of challenges including noisy state estima-

tion, specifying reward functions, and handling continuous action spaces, but has been used to investigate grasping Pinto and Gupta, 2016, pushing Agrawal et al., 2016, manipulation of 3D object models **krainin2011autonomous**, active learning **martinez2014active** and pouring liquids **schenck2017visual**. Deep reinforcement learning, which autonomously maximizes a given reward function, has been used to solve precise manipulation tasks Levine et al., 2016, grasping **pinto2017robust**; Levine et al., 2017, door opening **gu2016naf**, and navigation **kahn2018navigation**. These methods have succeeded on specific tasks, often with hard-coded reward functions. However, to scale task generalization robots may need to learn methods that can handle significant environment variation and require relatively little external supervision.

Several works have investigated self-supervised robotic interaction with varied objects in the deep learning setting with the goal of generalizing between objects. For example, in the domain of robotic grasping, several works have studied autonomous data collection to learn to grasp from a hand-specified grasping reward Pinto and Gupta, 2016; Levine et al., 2017. However, hand-specifying such rewards in general settings and for arbitrary manipulation skills is very cumbersome. Other work has focused on self-supervised learning with visual forward models, either by enforcing a simplified dynamical structure **zhang2019solar**; Watter et al., 2015 or with pixel transformer architectures **ebert2018retrying**; **lee2018videoprediction**; **ebert2018journal**; Finn and Levine, 2016; Ebert et al., 2017. However, these methods rely on accurate visual forward modelling, which is itself a very challenging problem. Instead, we build on self-supervised model-free approaches, which allow the agent to efficiently reach visual goals without planning with a visual forward model.

Prior work has also sought to perform self-supervised learning with model-free approaches. Using visual inverse models Agrawal et al., 2016 is one such approach, but may not work well for complex interaction dynamics or longer horizon planning. Most closely related to our approach are prior methods on goal-conditioned reinforcement learning Kaelbling, 1993; Schaul et al., 2015; Andrychowicz et al., 2017. The methods have been extended to frame self-supervised RL as learning goal reaching with automatically proposed goals, including visually-specified goals **nair2018rig**; **pong2019skewfit**; **wadefarley2019discern**; **florensa2019self**; **lin2019rlwithoutstate**. However, they generally focus on learning in narrow environments with little between-trial variability. In this setting, any previously visited state represents a valid goal. However, in the general case, this is no longer true: when the robot is presented with a different scene or different objects on each trial, it must only set those goals that can be accomplished in the current scene. In contrast, we focus on enabling self-supervised learning from off-policy data in

heterogeneous environments with increased factors of variability.

3.3 BACKGROUND

In this section, we provide an overview of relevant prior work necessary to understand our method, including goal-conditioned reinforcement learning and representation learning with variational auto-encoders.

3.3.1 *Goal-Conditioned Reinforcement Learning*

In an MDP consisting of states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, dynamics $p(s_{t+1}|s_t, a_t)$, rewards r_t, horizon H, and discount factor γ, reinforcement learning addresses optimizing a policy $a_t = \pi_\theta(s_t)$ to maximize expected return $\mathbb{E}[\sum_{t=0}^{H} \gamma^t r_t]$. To learn a variety of skills, it is instead convenient to optimize over a family of reward functions parametrized by goals, as in the framework of goal-conditioned RL Kaelbling, 1993. A variety of RL algorithms exist, but as we are primarily interested in sample-efficient off-policy learning, we consider goal-conditioned Q-learning algorithms Schaul et al., 2015. These algorithms learn a parametrized Q-function $Q_w(s, a, g)$ that estimates the expected return of taking action a from state s with goal g. Q-learning methods rely on minimizing the Bellman error:

$$\mathcal{L} = |Q_w(s, a, g) - (r + \max_{a'} Q_w(s', a', g))| \tag{4}$$

This objective can be optimized using standard actor-critic algorithms using a set of transitions (s, a, s', g, r) which can be collected off-policy Lillicrap et al., 2016. In practice, a target network is often used for the second Q-function.

3.3.2 *Variational Auto-Encoders*

Above, the goal description can take many forms. To handle high-dimensional goals, in this work, we learn a latent representation of the state using a variational auto-encoder (VAE). A VAE is a probabilistic generative model that has been shown to learn structured representations of high-dimensional data Kingma and Welling, 2014 successfully. It is trained by reconstructing states s with a parametrized encoder that converts the state into a normal random distribution $q_\phi(z|s)$ with a parametrized decoder that converts the latent variable z into a predicted state distribution $p_\psi(s|z)$, while keeping the latent

z close (in KL divergence) to its prior $p(z)$, a standard normal distribution. To train the encoder and decoder parameters ϕ and ψ, we jointly optimize both objectives when minimizing the negative evidence lower bound:

$$\mathcal{L}_{\text{VAE}} = -\mathbb{E}_{q_\phi(z|s)}[\log p(s|z)] + \beta D_{\text{KL}}(q_\phi(z|s)||p(z)). \tag{5}$$

3.3.3 *Conditional Variational Auto-Encoders*

Instead of a generative model that learns to generate the dataset distribution, one might instead desire a more structured generative model that can generate samples based on structured input. One example of this is a conditional variational auto-encoder (CVAE) that conditions the output on some input variable c and samples from $p(x|c)$ **sohn2015cvae**. For example, to train a model that generates images of digits given the desired digit, the input variable c might be a one-hot encoded vector of the desired digit.

A CVAE trains $q_\phi(z|s,c)$ and $q_\psi(s|z,c)$, where both the encoder and decoder has access to the input variable c. The CVAE then minimizes:

$$\mathcal{L}_{\text{CVAE}} = -\mathbb{E}_{q_\phi(z|s,c)}[\log p(s|z,c)] + \beta D_{\text{KL}}(q_\phi(z|s,c)||p(z)). \tag{6}$$

Samples are generated by first sampling a latent $z \sim p(z)$. Based on c, we can then decode z with $q_\psi(s|z,c)$ and visualize the output, which is in our case an image. In our framework $c = s_0$.

3.3.4 *Reinforcement Learning with Imagined Goals*

To learn skills from raw observations in a self-supervised manner, reinforcement learning with imagined goals (RIG) proposed to use representation learning combined with goal-conditioned RL **nair2018rig**. The aim of RIG is to choose actions in order to make the state s_t reach a goal image s_g at test time. RIG first collects an interaction dataset $\{s_t\}$ and learns a latent representation by training a VAE on this data. Then, a goal-conditioned policy is trained to act in the environment in order to reach a given goal z_g. Exploration data is collected by rolling out the policy with goals that are "imagined" from the VAE prior; at test time, the policy can take in a goal image as input, encode the image to the latent space, and act to reach the goal latent. A key limitation of this method is that sampling goals from the VAE prior during training time assumes that every state in the dataset is reachable at any time. However, in general, this assumption may not be true.

3.4 SELF-SUPERVISED LEARNING WITH CONTEXT-CONDITIONED REPRESENTATIONS

In this work, our goal is to enable the learning of flexible goal-conditioned policies that can be used to successfully perform a variety of tasks in a variety of contexts – e.g., with different objects in the scene. Such policies must learn from large amounts of experience, and although it is in principle possible to use random exploration to collect this experience, this quickly becomes impractical in the real world. It is, therefore, necessary for the robot to set its own goals during self-supervised training, to collect meaningful experience. However, in diverse settings, many randomly generated goals may not be feasible – e.g., the robot cannot push a red puck if the red puck is not present in the scene. We propose to extend off-policy goal-conditioned reinforcement learning with a conditional goal setting model, which proposes only those goals that are currently feasible. This enables a learning regime with imagined goals that is more realistic for real-world robotic systems that must generalize effectively to a range of objects and settings.

3.4.1 *Context-Conditioned VAEs*

To train a generative model that can improve the generation of feasible goals in varied scenes, we use a modified CVAE that uses the initial state s_0 in a rollout as the input c, which we call the "context" for that rollout. The modified CVAE, which we call a context-conditioned VAE (CC-VAE), is shown in Figure 18. While most CVAE applications use a one-hot vector as the input, we use an image s_0. This image is encoded with a convolutional encoder e_0 into a compact representation z_c. Note that by design, e and e_0 do not share weights, as they are intended to encode different factors of variation in the images. The context z_c is used to output the latent representation z_t, as well as the reconstruction of the state $\hat{s}_t$. In addition, z_c is used alone to (deterministically) decode $\hat{s}_0 = d_0(z_c)$. The objective is given by

$$\mathcal{L}_{\text{CC-VAE}} = \mathcal{L}_{\text{CVAE}} + \log p(s_0|z_c). \tag{7}$$

Due to the information bottleneck on z_t, this loss function penalizes information passing through z_t but allows for unrestricted information flow from z_c. Therefore, the optimal solution would encode as much information as possible in z_c, while only including the state information that changes within a trajectory in the latent variable z_t. These are precisely the features of most interest for control.

Algorithm 2 Context-Conditioned RIG

Require: Encoders $\mu(s_t, s_0)$, $e_0(s_0)$, policy $\pi_\theta(\bar{z}, \bar{z_g})$, goal-conditioned value function $Q_w(\bar{z}, \bar{z_g})$, dataset $\mathcal{D} = \{\tau^{(i)}\}$ of trajectories.

Train CC-VAE on $\mathcal{D}$ by optimizing equation 7.
for $n = 0, ..., N-1$ episodes **do**
  Sample latent goal $z_g \sim p(z)$, $\bar{z}_g = (z_g, z_c)$.
  Sample initial state $s_0 \sim p(s_0)$.
  Encode $z_c = e_0(s_0)$
  for $t = 0, ..., H-1$ steps **do**
    Observe s_t and encode $\bar{z}_t = (\mu(s_t, s_0), z_c)$
    Get action $a_t = \pi_\theta(\bar{z}_t, \bar{z}_g) + \text{noise}$.
    Get next state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.
    Store $(\bar{z}_t, a_t, \bar{z}_{t+1}, \bar{z}_g)$ into replay buffer $\mathcal{R}$.
    Sample transition $(\bar{z}, a, \bar{z}', \bar{z}_g) \sim \mathcal{R}$.
    Compute new reward $r = -\|\bar{z}' - \bar{z}_g\|$.
    Minimize equation 31 using $(\bar{z}, a, \bar{z}', \bar{z}_g, r)$.
  end for
end for=0

3.4.2 *Context-Conditioned Reinforcement Learning with Imagined Goals*

We propose to use our context-conditioned VAE in the RIG framework to learn policies over environments with visual diversity, where each episode might involve interacting with a different scene and different objects. We first collect a dataset of trajectories $\mathcal{D} = \{\tau^{(i)}\}$ by executing random actions in the environment. We then learn a CC-VAE, as detailed in Section 3.4.1, to learn a factored representation of the image observations. To use the CC-VAE for self-supervised learning, we save the first image s_0 when starting a rollout. We compute the encoding of s_0, $z_c = e_0(s_0)$. Let $\bar{z}$ denote the context concatenated vector (z, z_c), and let $\mu(s, s_0)$ denote the mean of $q_\phi(z|s, s_0)$. We then use RIG in the $\bar{z}$ latent space by encoding observations with μ, meaning that we train a goal-conditioned policy $\pi(\bar{z}, \bar{z_g})$ and a goal-conditioned Q-function $Q(\bar{z}, \bar{z_g})$.

To collect data, we sample a latent goal for each rollout from the prior $z_g \sim N(0, I)$, as in RIG. For every observation s_t, we compute the mean encoding $\mu_t(s_t, s_0)$. We then obtain a rollout of the policy by executing $\pi(\bar{z}, \bar{z_g})$. The reward at each timestep is the latent distance $\|\bar{\mu}_t - \bar{z}_g\|$.

The policy and Q-function can be trained with any off-policy reinforcement learning algorithm. We use TD3 in our implementation Fujimoto et al., 2018. Our policy and Q-function are goal-conditioned, and we take advantage of being able to relabel the

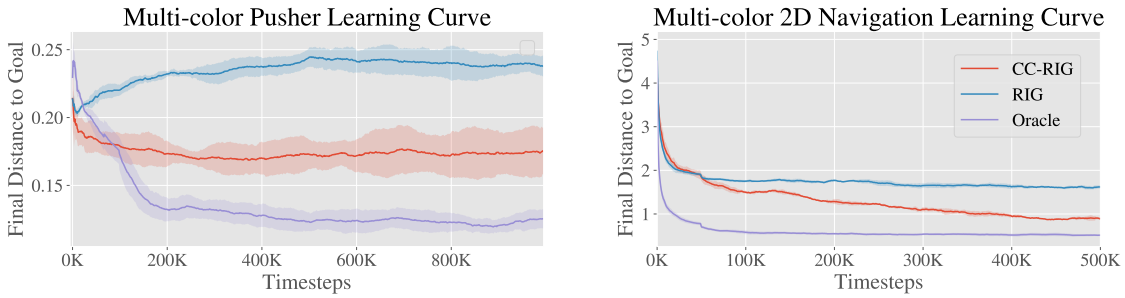


Figure 10: Self-supervised learning results in visually varied simulated environments. CC-RIG significantly outperforms RIG and is competitive with the oracle method that has direct access to ground truth states. The simulated pusher environment (left) is shown in Figure 11 and the navigation environment is shown in Figure **??**.

goals for each transition to improve sample efficiency **nair2018rig**; **pong2019skewfit**; Andrychowicz et al., 2017. However, when relabeling a goal $\bar{z}_g$ with a random goal from the environment, the context-conditioning is still preserved. That is, if $z'_g \sim N(0, 1)$ is the new sampled goal, we use $\bar{z}'_g = (z'_g, z_c)$. This ensures that the relabeled goal is compatible with the scene for the corresponding transition.

After training, we can use the learned policy π to reach a visually indicated goal. Given a goal image s_g, we encode it into a latent goal $z_g = \mu(s_g, s_0)$. Then, we execute the policy with the latent goal $\bar{z}_g$, just as during the training phase. The complete algorithm is presented in Algorithm 2.

3.5 EXPERIMENTS

In our experiments, we aim to answer the following questions:

1. How does our method compare to prior work at learning self-supervised skills in visually diverse environments?
2. Do context-conditioned VAEs learn an image representation that produces coherent and diverse goals that are suitable for the current scene?
3. Can our proposed context-conditioned RIG method handle diverse real-world data and learn effective policies under visual variation in the real world?

3.5.1 *Self-Supervised Learning in Simulation*

In simulation, we can conduct controlled experiments and evaluate against known underlying state values to measure the performance of our approach and prior methods.

As a simulation test-bed, we use a multi-color pusher environment simulated in MuJoCo Todorov et al., 2012. In this environment (Figure 11(left)), a simulated Sawyer arm is tasked with pushing a circular puck to a target position, specified by a goal image at test time. On each rollout, the puck color is set to a random RGB value. Therefore, the goal proposals for each method must adequately account for the color of the puck – a goal that requires moving a red puck to a given location is impossible if only a blue puck is present in the scene.

We compare the following algorithms: **CC-RIG.** Our method using a CC-VAE for representation learning, as described in Section 3.4.2. **RIG.** Reinforcement learning with imagined goals **nair2018rig** using a standard VAE, as described in Section 3.3.4. **Oracle.** The oracle agent runs goal-conditioned RL with direct access to state information. Achieving performance similar to the oracle indicates that an algorithm loses little from using raw image observations over ground truth state.

Learning curves comparing these methods are presented in the plot on the left in Figure 10. CC-RIG outperforms RIG significantly, and standard RIG is not able to improve beyond the initial random policy. The performance of CC-RIG approaches that of the oracle policy, which has access to the true state. This suggests that, in visually varied environments, self-supervised learning is possible so long as the visual complexity is factored out with representation learning, and the proposed goals are consistent with the appearance of the current scene.

3.5.2 *Generalizing to Varying Appearance and Dynamics with Self-Supervised Learning*

In this experiment, to study changing both visual appearance and physical dynamics, we study how well our method can generalize when the environment dynamics change. We use a simulated 2D navigation task, where the goal is to navigate a point robot around an obstacle. The arrangement of the obstacles is chosen from a set of 15 possible configurations, and the color of the point robot is generated from a random RGB value. Learning curves obtained by training the different methods above in this environment are presented in Figure 10. CC-RIG requires more samples to learn, but eventually approaches the oracle performance. RIG, in comparison, plateaus with poor performance. This environment is explained further in the supplementary, in Section **??** and Figure **??**.

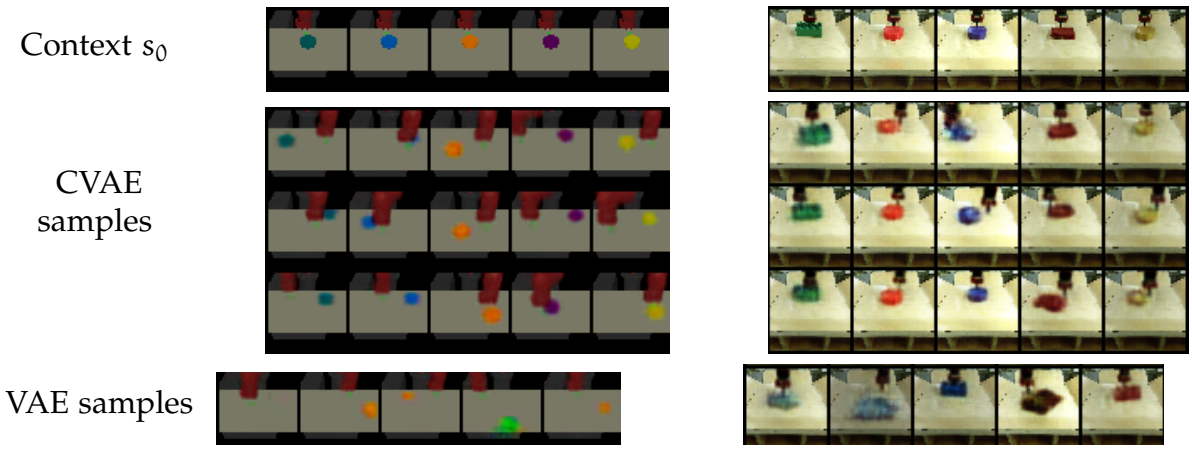


Figure 11: Comparing samples from our CC-VAE model to a standard VAE. The initial image s_0 is shown on the top row, and samples conditioned on s_0 are shown below. Our model coherently maintains object color and geometry in its samples, suggesting that the context conditioned model can successfully factor out the scene-specific object identity from the variable object position. This enables the use of the CC-VAE for goal proposals in visually diverse scenes.

3.5.3 *Context-Conditioned VAE Goal Sampling*

To better understand why CC-RIG outperforms RIG, we compare the samples from our CC-VAE to a standard VAE. Samples from both models are shown in Figure 11. The quality of the samples reveals why the CC-VAE provides better goal setting for self-supervised learning. In all environments, the samples from the CC-VAE maintain the background, object shape, and object color from the initial state. Therefore, the goals are more meaningful in the CC-VAE latent space.

This kind of visualization is a good indicator for the suitability of the representation for self-supervised learning. Diverse, coherent samples indicate that the latent space captures the appropriate factors of change in the environment and can be useful for self-supervised policy learning. Good samples also suggest that the latent space is well-structured, and therefore distances in the latent space should provide a good reward function for goal-reaching. In practice, we also look at the quality of the reconstructions. Good reconstructions confirm that the latent variables capture sufficient information about the image to be used in place of the image itself as a state representation.

3.5.4 *Real-World Robotic Evaluation*

In this experiment, we evaluate whether our method can handle manipulating visually varied objects in the real world. We use CC-RIG to train a Sawyer robot to manipulate a variety of objects, placed one at a time in a bin in front of the robot. As before, the training phase is self-supervised, and the robot must match a given goal image at test time. The robot setup is shown in Figure 18.

We first collect a large dataset with random actions and train a CC-VAE on the data. Samples from the model are shown in Figure 12. The CC-VAE learns to generate goals with the correct object. To handle varying brightness at different times of the day, we added data augmentation by applying a color jitter filter to (s_0, s_t) pairs. As seen in the figure, the model is robust to this factor of variation. Each sample contains the same type of object, brightness level, and background as the initial state that it is conditioned on. However, crucially, these factors of variation are not present in z_t, as evidenced by the fact they do not vary within each column of Figure 12, but the object position does.

Next, we run CC-RIG with the trained CC-VAE to learn to reach visually indicated goals in a self-supervised manner. We first conduct fully off-policy training using the same dataset as was used to train the CC-VAE, consisting of 50,000 samples (about 3 hours) of total interaction with 20 objects. Then, we collect a small amount of additional on-policy data to finetune the policy, analogous to recent work on large-scale vision-based robotic reinforcement learning **pmlr-v87-kalashnikov18a**. The robot learns to push objects to target locations, indicated by a goal image. The real-world results are presented in Figure 12. Because it is difficult to automatically detect the positions of objects, we show some representative rollout examples, and we compute several distance metrics between the final state of a rollout and the goal: **CVAE distance.** CC-VAE latent space distance between final image and goal. **VAE distance.** VAE latent space distance between final image and goal. **Pixel distance.** We manually label the center of mass of the object in the final image and goal image, and compute the distance between them. **Object distance.** We measure the distance between the physical goal position of the object and the final position. In each metric, CC-RIG outperforms RIG.

At training time, the dataset consists of interaction with 20 objects. The result of running CC-RIG on novel objects that were not included in the dataset are shown in the table as "CC-RIG, novel" and in the rollouts in Figure 12. These results show that our method can also generalize its experience to push novel objects it has not seen before.

Real-World Pushing Results

	CVAE distance	VAE distance	Pixel distance	Object distance (cm)
RIG	2.37 ± 0.97	2.41 ± 0.93	93.9 ± 41.7	17.1 ± 8.2
CC-RIG	1.66 ± 0.63	2.17 ± 0.88	56.8 ± 34.5	14.0 ± 6.9
CC-RIG, novel	1.51 ± 0.71	1.91 ± 0.87	53.1 ± 24.9	11.5 ± 2.9

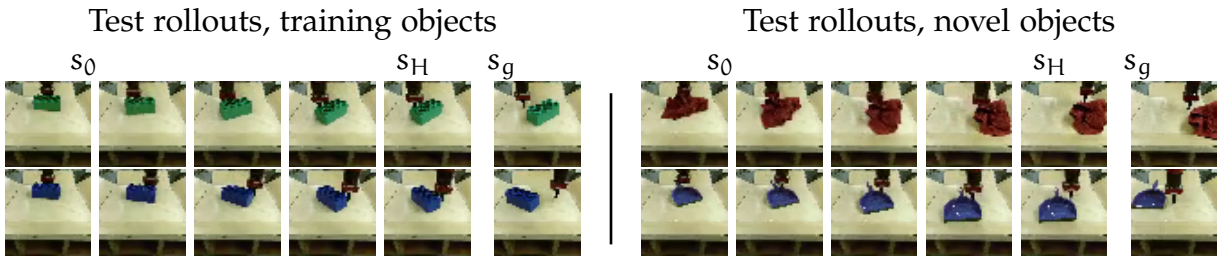


Figure 12: The table above shows the performance of our method in the real-world, evaluated with four different evaluation metrics[1]. CC-RIG outperforms RIG in each one, even when tested on novel objects that it has not been trained on. Test rollouts of our method are shown on training objects on the left and unseen novel objects on the right. Successful rollouts where the object is pushed to the goal location are shown in top row, and failure modes are shown in the bottom row.

3.6 CONCLUSION

We presented a method for sample-efficient, flexible self-supervised task learning for environments with visual diversity. Our method can learn effective behavior without external supervision in simulated environments with randomized colors and layout, and in a real-world pushing task with differently colored pucks. Each environment contains an axis of visual variation that requires our algorithm to utilize an intelligent goal-setting strategy, to ensure that the self-proposed goals are consistent with the tasks and feasible in the current scene.

The main idea behind our method is to devise a context-conditioned goal proposal mechanism, allowing our self-supervised reinforcement learning algorithm to propose goals for itself that are feasible to reach. This context-conditioned VAE model factors out the unchanging context of a rollout, such as which objects are present in the scene, from the controllable aspects, such as the object positions to construct a more generalizable goal proposal model.

We believe this contribution will enable scalable learning in the real world. An agent manipulating objects in the real world must handle many forms of variation: different manipulation skills to learn, objects to manipulate, as well as variation in lighting, textures, etc. Methods that learn from data must be able to represent these variations while at the same time taking advantage of common structure across objects and tasks in order to achieve practical sample efficiency. Future work will address the remaining challenges to achieve this vision.

1 The first three metrics are computed on 40 trajectories per method, and we report mean $\pm$ standard deviation. Object distance is computed on 10 trajectories per method, and we report median $\pm$ standard deviation.

Part II

UTILIZING PRIOR DATA FOR CONTROL

4

OVERCOMING EXPLORATION IN REINFORCEMENT LEARNING WITH DEMONSTRATIONS

4.1 INTRODUCTION

RL has found significant success in decision making for solving games, so what makes it more challenging to apply in robotics? A key difference is the difficulty of exploration, which comes from the choice of reward function and complicated environment dynamics. In games, the reward function is usually given and can be directly optimized. In robotics, we often desire behavior to achieve some binary objective (e.g., move an object to a desired location or achieve a certain state of the system) which naturally induces a sparse reward. Sparse reward functions are easier to specify and recent work suggests that learning with a sparse reward results in learned policies that perform the desired objective instead of getting stuck in local optima **vecerik17ddpgfd**; Andrychowicz et al., 2017. However, exploration in an environment with sparse reward is difficult since with random exploration, the agent rarely sees a reward signal.

The difficulty posed by a sparse reward is exacerbated by the complicated environment dynamics in robotics. For example, system dynamics around contacts are difficult to model and induce a sensitivity in the system to small errors. Many robotics tasks also require executing multiple steps successfully over a long horizon, involve high dimensional control, and require generalization to varying task instances. These conditions further result in a situation where the agent so rarely sees a reward initially that it is not able to learn at all.

All of the above means that random exploration is not a tenable solution. Instead, in this work we show that we can use demonstrations as a guide for our exploration. To test our method, we solve the problem of stacking several blocks at a given location from a random initial state. Stacking blocks has been studied before in the literature **deisenroth2011blocks**; **duan2017oneshotimitation** and exhibits many of the difficulties

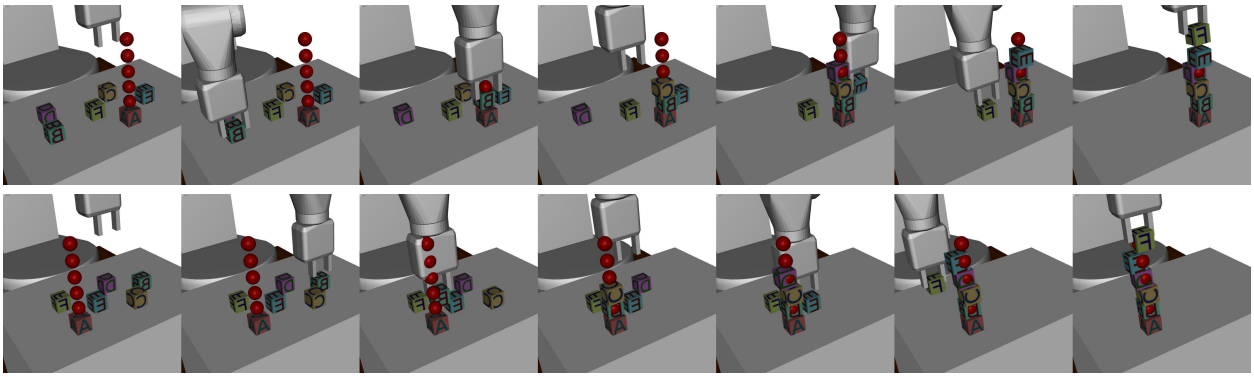


Figure 13: We present a method using reinforcement learning to solve the task of block stacking shown above. The robot starts with 6 blocks labelled A through F on a table in random positions and a target position for each block. The task is to move each block to its target position. The targets are marked in the above visualization with red spheres which do not interact with the environment. These targets are placed in order on top of block A so that the robot forms a tower of blocks. This is a complex, multi-step task where the agent needs to learn to successfully manage multiple contacts to succeed. Frames from rollouts of the learned policy are shown. A video of our experiments can be found at: http://ashvin.me/demoddpg-website

mentioned: long horizons, contacts, and requires generalizing to each instance of the task. We limit ourselves to 100 human demonstrations collected via teleoperation in virtual reality. Using these demonstrations, we are able to solve a complex robotics task in simulation that is beyond the capability of both reinforcement learning and imitation learning.

The primary contribution of this paper is to show that demonstrations can be used with reinforcement learning to solve complex tasks where exploration is difficult. We introduce a simple auxiliary objective on demonstrations, a method of annealing away the effect of the demonstrations when the learned policy is better than the demonstrations, and a method of resetting from demonstration states that significantly improves and speeds up training policies. By effectively incorporating demonstrations into RL, we short-circuit the random exploration phase of RL and reach nonzero rewards and a reasonable policy early on in training. Finally, we extensively evaluate our method against other commonly used methods, such as initialization with learning from demonstrations and fine-tuning with RL, and show that our method significantly outperforms them.

Learning methods for decision making problems such as robotics largely divide into two classes: imitation learning and reinforcement learning (RL). In imitation learning (also called learning from demonstrations) the agent receives behavior examples from an expert and attempts to solve a task by copying the expert's behavior. In RL, an agent attempts to maximize expected reward through interaction with the environment. Our work combines aspects of both to solve complex tasks.

Imitation Learning: Perhaps the most common form of imitation learning is behavior cloning (BC), which learns a policy through supervised learning on demonstration state-action pairs. BC has seen success in autonomous driving (**pomerleau1989alvinn**; **bojarski2016nvidia**), quadcopter navigation (**giusti15trails**), locomotion (**nakanishi2004bipedlfd**; **kalakrishnan09terraintemplates**). BC struggles outside the manifold of demonstration data. Dataset Aggregation (DAGGER) augments the dataset by interleaving the learned and expert policy to address this problem of accumulating errors (**ross2011dagger**). However, DAGGER is difficult to use in practice as it requires access to an expert during all of training, instead of just a set of demonstrations.

Fundamentally, BC approaches are limited because they do not take into account the task or environment. Inverse reinforcement learning (IRL) (**ng2000irl**) is another form of imitation learning where a reward function is inferred from the demonstrations. Among other tasks, IRL has been applied to navigation (**ziebart2008maxent**), autonomous helicopter flight (**abbeel2004apprenticeship**), and manipulation (**finn16guidedcostlearning**). Since our work assumes knowledge of a reward function, we omit comparisons to IRL approaches.

Reinforcement Learning: Reinforcement learning methods have been harder to apply in robotics, but are heavily investigated because of the autonomy they could enable. Through RL, robots have learned to play table tennis (**peters2010reps**), swing up a cartpole, and balance a unicycle (**deisenroth2011pilco**). A renewal of interest in RL cascaded from success in games (**mnih2015human**; **Silver2016**), especially because of the ability of RL with large function approximators (ie. deep RL) to learn control from raw pixels. Robotics has been more challenging in general but there has been significant progress. Deep RL has been applied to manipulation tasks (**LevineFDA15**), grasping (**levine2016learning**; Pinto and Gupta, 2016), opening a door (**Gu2016b**), and locomotion (**schulman2015trpo**; Lillicrap et al., 2016; Mnih et al., 2016). However, results have been attained predominantly in simulation per high sample complexity, typically caused by exploration challenges.

Robotic Block Stacking: Block stacking has been studied from the early days of AI and robotics as a task that encapsulates many difficulties of more complicated tasks we want to solve, including multi-step planning and complex contacts. SHRDLU (**winograd72shrdlr**) was one of the pioneering works, but studied block arrangements only in terms of logic and natural language understanding. More recent work on task and motion planning considers both logical and physical aspects of the task (**Kaelbling2011**; **Kavraki1996**; **srivastava14tamp**), but requires domain-specific engineering. In this work we study how an agent can learn this task without the need of domain-specific engineering.

One RL method, PILCO (**deisenroth2011pilco**) has been applied to a simple version of stacking blocks where the task is to place a block on a tower (**deisenroth2011blocks**). Methods such as PILCO based on learning forward models naturally have trouble modelling the sharply discontinuous dynamics of contacts; although they can learn to place a block, it is a much harder problem to grasp the block in the first place. One-shot Imitation (**duan2017oneshotimitation**) learns to stack blocks in a way that generalizes to new target configurations, but uses more than 100,000 demonstrations to train the system. A heavily shaped reward can be used to learn to stack a Lego block on another with RL (**popov17stacking**). In contrast, our method can succeed from fully sparse rewards and handle stacking several blocks.

Combining RL and Imitation Learning: Previous work has combined reinforcement learning with demonstrations. Demonstrations have been used to accelerate learning on classical tasks such as cart-pole swing-up and balance (**schaal97lfd**). This work initialized policies and (in model-based methods) initialized forward models with demonstrations. Initializing policies from demonstrations for RL has been used for learning to hit a baseball (**peters2008baseball**) and for underactuated swing-up (**kober2008mp**). Beyond initialization, we show how to extract more knowledge from demonstrations by using them effectively throughout the entire training process.

Our method is closest to two recent approaches — Deep Q-Learning From Demonstrations (DQfD) (**hester17dqfd**) and DDPG From Demonstrations (DDPGfD) (**vecerik17ddpgfd**) which combine demonstrations with reinforcement learning. DQfD improves learning speed on Atari, including a margin loss which encourages the expert actions to have higher Q-values than all other actions. This loss can make improving upon the demonstrator policy impossible which is not the case for our method. Prior work has previously explored improving beyond the demonstrator policy in simple environments by introducing slack variables (**kim2013apid**), but our method uses a learned value to actively inform the improvement. DDPGfD solves simple robotics tasks akin to peg insertion using DDPG with demonstrations in the replay buffer. In contrast to this prior work, the

tasks we consider exhibit additional difficulties that are of key interest in robotics: multi-step behaviours, and generalization to varying goal states. While previous work focuses on speeding up already solvable tasks, we show that we can extend the state of the art in RL with demonstrations by introducing new methods to incorporate demonstrations.

4.3 BACKGROUND

4.3.1 *Reinforcement Learning*

We consider the standard Markov Decision Process framework for picking optimal actions to maximize rewards over discrete timesteps in an environment E. We assume that the environment is fully observable. At every timestep t, an agent is in a state x_t, takes an action a_t, receives a reward r_t, and E evolves to state x_{t+1}. In reinforcement learning, the agent must learn a policy $a_t = \pi(x_t)$ to maximize expected returns. We denote the return by $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$ where T is the horizon that the agent optimizes over and γ is a discount factor for future rewards. The agent's objective is to maximize expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_0]$.

A variety of reinforcement learning algorithms have been developed to solve this problem. Many involve constructing an estimate of the expected return from a given state after taking an action:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_t | s_t, a_t] \tag{8}$$

$$= \mathbb{E}_{r_t, s_{t+1} \sim E}[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{9}$$

We call Q^{π} the action-value function. Equation 21 is a recursive version of equation 20, and is known as the Bellman equation. The Bellman equation allows for methods to estimate Q that resemble dynamic programming.

4.3.2 *DDPG*

Our method combines demonstrations with one such method: Deep Deterministic Policy Gradients (DDPG) Lillicrap et al., 2016. DDPG is an off-policy model-free reinforcement learning algorithm for continuous control which can utilize large function approximators such as neural networks. DDPG is an actor-critic method, which bridges the gap between policy gradient methods and value approximation methods for RL. At a high level, DDPG learns an action-value function (critic) by minimizing the Bellman error,

while simultaneously learning a policy (actor) by directly maximizing the estimated action-value function with respect to the parameters of the policy.

Concretely, DDPG maintains an actor function $\pi(s)$ with parameters θ_π, a critic function $Q(s, a)$ with parameters θ_Q, and a replay buffer R as a set of tuples (s_t, a_t, r_t, s_{t+1}) for each transition experienced. DDPG alternates between running the policy to collect experience and updating the parameters. Training rollouts are collected with extra noise for exploration: $a_t = \pi(s) + \mathcal{N}$, where $\mathcal{N}$ is a noise process.

During each training step, DDPG samples a minibatch consisting of N tuples from R to update the actor and critic networks. DDPG minimizes the following loss L w.r.t. θ_Q to update the critic:

$$y_i = r_i + \gamma Q(s_{i+1}, \pi(s_{i+1})) \tag{10}$$

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta_Q))^2 \tag{11}$$

The actor parameters θ_π are updated using the policy gradient:

$$\nabla_{\theta_\pi} J = \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta_Q)|_{s=s_i, a=\pi(s)} \nabla_{\theta_\pi} \pi(s|\theta_\pi)|_{s_i} \tag{12}$$

To stabilize learning, the Q value in equation 10 is usually computed using a separate network (called the *target* network) whose weights are an exponential average over time of the critic network. This results in smoother target values.

Note that DDPG is a natural fit for using demonstrations. Since DDPG can be trained off-policy, we can use demonstration data as off-policy training data. We also take advantage of the action-value function $Q(s, a)$ learned by DDPG to better use demonstrations.

4.3.3 *Multi-Goal RL*

Instead of the standard RL setting, we train agents with parametrized goals, which lead to more general policies Schaul et al., 2015 and have recently been shown to make learning with sparse rewards easier Andrychowicz et al., 2017. Goals describe the task we expect the agent to perform in the given episode, in our case they specify the desired positions of all objects. We sample the goal g at he beginning of every episode. The function approximators, here π and Q, take the current goal as an additional input.

4.3.4 *Hindsight Experience Replay (HER)*

To handle varying task instances and parametrized goals, we use Hindsight Experience Replay (HER) Andrychowicz et al., 2017. The key insight of HER is that even in failed rollouts where no reward was obtained, the agent can transform them into successful ones by assuming that a state it saw in the rollout was the actual goal. HER can be used with any off-policy RL algorithm assuming that for every state we can find a goal corresponding to this state (i.e. a goal which leads to a positive reward in this state).

For every episode the agent experiences, we store it in the replay buffer twice: once with the original goal pursued in the episode and once with the goal corresponding to the final state achieved in the episode, as if the agent intended on reaching this state from the very beginning.

4.4 METHOD

Our method combines DDPG and demonstrations in several ways to maximally use demonstrations to improve learning. We describe our method below and evaluate these ideas in our experiments.

4.4.1 *Demonstration Buffer*

First, we maintain a second replay buffer R_D where we store our demonstration data in the same format as R. In each minibatch, we draw an extra N_D examples from R_D to use as off-policy replay data for the update step. These examples are included in both the actor and critic update. This idea has been introduced in **vecerik17ddpgfd**.

4.4.2 *Behavior Cloning Loss*

Second, we introduce a new loss computed only on the demonstration examples for training the actor.

$$L_{BC} = \sum_{i=1}^{N_D} \pi(s_i|\theta_\pi) - a_i{}^2 \tag{13}$$

This loss is a standard loss in imitation learning, but we show that using it as an auxiliary loss for RL improves learning significantly. The gradient applied to the actor parameters

θ_π is:

$$\lambda_1 \nabla_{\theta_\pi} J - \lambda_2 \nabla_{\theta_\pi} L_{BC} \tag{14}$$

(Note that we maximize J and minimize L_{BC}.) Using this loss directly prevents the learned policy from improving significantly beyond the demonstration policy, as the actor is always tied back to the demonstrations. Next, we show how to account for suboptimal demonstrations using the learned action-value function.

4.4.3 *Q-Filter*

We account for the possibility that demonstrations can be suboptimal by applying the behavior cloning loss only to states where the critic $Q(s, a)$ determines that the demonstrator action is better than the actor action:

$$L_{BC} = \sum_{i=1}^{N_D} \pi(s_i|\theta_\pi) - a_i{}^2 \mathbb{1}_{Q(s_i,a_i)>Q(s_i,\pi(s_i))} \tag{15}$$

The gradient applied to the actor parameters is as in equation 14. We label this method using the behavior cloning loss and Q-filter "Ours" in the following experiments.

4.4.4 *Resets to demonstration states*

To overcome the problem of sparse rewards in very long horizon tasks, we reset some training episodes using states and goals from demonstration episodes. Restarts from within demonstrations expose the agent to higher reward states during training. This method makes the additional assumption that we can restart episodes from a given state, as is true in simulation.

To reset to a demonstration state, we first sample a demonstration $D = (x_0, u_0, x_1, u_1, ...x_N, u_N)$ from the set of demonstrations. We then uniformly sample a state x_i from D. As in HER, we use the final state achieved in the demonstration as the goal. We roll out the trajectory with the given initial state and goal for the usual number of timesteps. At evaluation time, we do not use this procedure.

We label our method with the behavior cloning loss, Q-filter, and resets from demonstration states as "Ours, Resets" in the following experiments.

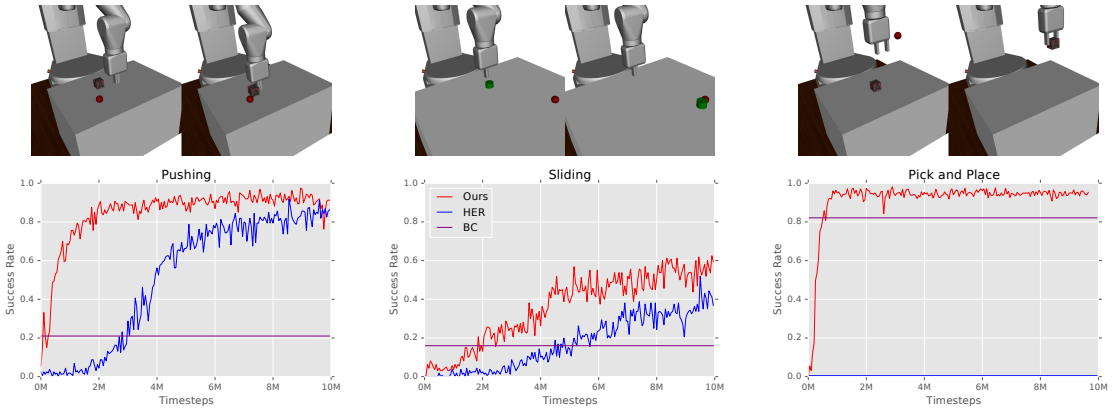


Figure 14: Baseline comparisons on tasks from Andrychowicz et al., 2017. Frames from the learned policy are shown above each task. Our method significantly outperforms the baselines. On the right plot, the HER baseline always fails.

4.5 EXPERIMENTAL SETUP

4.5.1 *Environments*

We evaluate our method on several simulated MuJoCo Todorov et al., 2012 environments. In all experiments, we use a simulated 7-DOF Fetch Robotics arm with parallel grippers to manipulate one or more objects placed on a table in front of the robot.

The agent receives the positions of the relevant objects on the table as its observations. The control for the agent is continuous and 4-dimensional: 3 dimensions that specify the desired end-effector position[1] and 1 dimension that specifies the desired distance between the robot fingers. The agent is controlled at 50Hz frequency.

We collect demonstrations in a virtual reality environment. The demonstrator sees a rendering of the same observations as the agent, and records actions through a HTC Vive interface at the same frequency as the agent. We have the option to accept or reject a demonstration; we only accept demonstrations we judge to be mostly correct. The demonstrations are not optimal. The most extreme example is the "sliding" task, where only 7 of the 100 demonstrations are successful, but the agent still sees rewards for these demonstrations with HER.

1 In the 10cm x 10cm x 10cm cube around the current gripper position

4.5.2 *Training Details*

To train our models, we use Adam **kingma2014adam** as the optimizer with learning rate 10^{-3}. We use $N = 1024, N_D = 128, \lambda_1 = 10^{-3}, \lambda_2 = 1.0/N_D$. The discount factor γ is 0.98. We use 100 demonstrations to initialize R_D. The function approximators π and Q are deep neural networks with ReLU activations and L2 regularization with the coefficient 5×10^{-3}. The final activation function for π is tanh, and the output value is scaled to the range of each action dimension. To explore during training, we sample random actions uniformly within the action space with probability 0.1 at every step, and the noise process $\mathcal{N}$ is uniform over $\pm 10\%$ of the maximum value of each action dimension. Task-specific information, including network architectures, are provided in the next section.

4.5.3 *Overview of Experiments*

We perform three sets of experiments. In Sec. 4.6, we provide a comparison to previous work. In Sec. 4.7 we solve block stacking, a difficult multi-step task with complex contacts that the baselines struggle to solve. In Sec. 4.8 we do ablations of our own method to show the effect of individual components.

4.6 COMPARISON WITH PRIOR WORK

4.6.1 *Tasks*

We first show the results of our method on the simulated tasks presented in the Hindsight Experience Replay paper Andrychowicz et al., 2017. We apply our method to three tasks:

1. *Pushing*. A block placed randomly on the table must be moved to a target location on the table by the robot (fingers are blocked to avoid grasping).
2. *Sliding*. A puck placed randomly on the table must be moved to a given target location. The target is outside the robot's reach so it must apply enough force that the puck reaches the target and stops due to friction.
3. *Pick-and-place*. A block placed randomly on the table must be moved to a target location in the air. Note that the original paper used a form of initializing from favorable states to solve this task. We omit this for our experiment but discuss and evaluate the initialization idea in an ablation.

As in the prior work, we use a fully sparse reward for this task. The agent is penalized if the object is not at its goal position:

$$r_t = \begin{cases} 0, & \text{if } \|x_i - g_i\| < \delta \\ -1, & \text{otherwise} \end{cases} \tag{16}$$

where the threshold δ is 5cm.

4.6.2 *Results*

Fig. 14 compares our method to HER without demonstrations and behavior cloning. Our method is significantly faster at learning these tasks than HER, and achieves significantly better policies than behavior cloning does. Measuring the number of timesteps to get to convergence, we exhibit a 4x speedup over HER in pushing, a 2x speedup over HER in sliding, and our method solves the pick-and-place task while HER baseline cannot solve it at all.

The pick-and-place task showcases the shortcoming of RL in sparse reward settings, even with HER. In pick-and-place, the key action is to grasp the block. If the robot could manage to grasp it a small fraction of the time, HER discovers how to achieve goals in the air and reinforces the grasping behavior. However, grasping the block with random actions is extremely unlikely. Our method pushes the policy towards demonstration actions, which are more likely to succeed.

In the HER paper, HER solves the pick-and-place task by initializing half of the rollouts with the gripper grasping the block. With this addition, pick-and-place becomes the easiest of the three tasks tested. This initialization is similar in spirit to our initialization idea, but takes advantage of the fact that pick-and-place with any goal can be solved starting from a block grasped at a certain location. This is not always true (for example, if there are multiple objects to be moved) and finding such a keyframe for other tasks would be difficult, requiring some engineering and sacrificing autonomy. Instead, our method guides the exploration towards grasping the block through demonstrations. Providing demonstrations does not require expert knowledge of the learning system, which makes it a more compelling way to provide prior information.

4.7 MULTI-STEP EXPERIMENTS

4.7.1 *Block Stacking Task*

To show that our method can solve more complex tasks with longer horizon and sparser reward, we study the task of block stacking in a simulated environment as shown in Fig. 18 with the same physical properties as the previous experiments. Our experiments show that our approach can solve the task in full and learn a policy to stack 6 blocks with demonstrations and RL. To measure and communicate various properties of our method, we also show experiments on stacking fewer blocks, a subset of the full task.

We initialize the task with blocks at 6 random locations $x_1 ... x_6$. We also provide 6 goal locations $g_1 ... g_6$. To form a tower of blocks, we let $g_1 = x_1$ and $g_i = g_{i-1} + (0, 0, 5\text{cm})$ for $i \in 2, 3, 4, 5$.

By stacking N blocks, we mean N blocks reach their target locations. Since the target locations are always on top of x_1, we start with the first block already in position. So stacking N blocks involves $N - 1$ pick-and-place actions. To solve stacking N, we allow the agent $50 * (N - 1)$ timesteps. This means that to stack 6 blocks, the robot executes 250 actions or 5 seconds.

We recorded 100 demonstrations to stack 6 blocks, and use subsets of these demonstrations as demonstrations for stacking fewer blocks. The demonstrations are not perfect; they include occasionally dropping blocks, but our method can handle suboptimal demonstrations. We still rejected more than half the demonstrations and excluded them from the demonstration data because we knocked down the tower of blocks when releasing a block.

4.7.2 *Rewards*

Two different reward functions are used. To test the performance of our method under fully sparse reward, we reward the agent only if all blocks are at their goal positions:

$$r_t = \min_i \mathbb{1}_{\|x_i - g_i\| < \delta} \tag{17}$$

The threshold δ is the size of a block, 5cm. Throughout the paper we call this the "sparse" reward.

To enable solving the longer horizon tasks of stacking 4 or more blocks, we use the "step" reward :

$$r_t = -1 + \sum_i \mathbb{1}_{\|x_i - g_i\| < \delta} \tag{18}$$

Task	Ours	Ours, Resets	BC	HER	BC+ HER
Stack 2, Sparse	99%	97%	65%	0%	65%
Stack 3, Sparse	99%	89%	1%	0%	1%
Stack 4, Sparse	1%	54%	-	-	-
Stack 4, Step	91%	73%	0%	0%	0%
Stack 5, Step	49%	50%	-	-	-
Stack 6, Step	4%	32%	-	-	-

Figure 15: Comparison of our method against baselines. The value reported is the median of the best performance (success rate) of all randomly seeded runs of each method.

Note the step reward is still very sparse; the robot only sees the reward change when it moves a block into its target location. We subtract 1 only to make the reward more interpretable, as in the initial state the first block is already at its target.

Regardless of the reward type, an episode is considered successful for computing success rate if all blocks are at their goal position in their final state.

4.7.3 *Network architectures*

We use a 4 layer networks with 256 hidden units per layer for π and Q for the HER tasks and stacking 3 or fewer blocks. For stacking 4 blocks or more, we use an attention mechanism **bahdanau14attention** for the actor and a larger network. The attention mechanism uses a 3 layer network with 128 hidden units per layer to query the states and goals with one shared head. Once a state and goal is extracted, we use a 5 layer network with 256 hidden units per layer after the attention mechanism. Attention speeds up training slightly but does not change training outcomes.

4.7.4 *Baselines*

We include the following methods to compare our method to baselines on stacking 2 to 6 blocks. [2]

2 Because of computational constraints, we were limited to 5 random seeds per method for stacking 3 blocks, 2 random seeds per method for stacking 4 and 5 blocks, and 1 random seed per method for stacking 6 blocks. Although we are careful to draw conclusions from few random seeds, the results are

Ours: Refers to our method as described in section 4.4.3.
Ours, Resets: Refers to our method as described in section 4.4.3 with resets from demonstration states (Sec. 4.4.4).
BC: This method uses behavior cloning to learn a policy. Given the set of demonstration transitions R_D, we train the policy π by supervised learning. Behavior cloning requires much less computation than RL. For a fairer comparison, we performed a large hyperparameter sweep over various networks sizes, attention hyperparameters, and learning rates and report the success rate achieved by the best policy found.
HER: This method is exactly the one described in Hindsight Experience Replay Andrychowicz et al., 2017, using HER and DDPG.
BC+HER: This method first initializes a policy (actor) with BC, then finetunes the policy with RL as described above.

4.7.5 *Results*

We are able to learn much longer horizon tasks than the other methods, as shown in Fig. 15. The stacking task is extremely difficult using HER without demonstrations because the chance of grasping an object using random actions is close to 0. Initializing a policy with demonstrations and then running RL also fails since the actor updates depend on a reasonable critic and although the actor is pretrained, the critic is not. The pretrained actor weights are therefore destroyed in the very first epoch, and the result is no better than BC alone. We attempted variants of this method where initially the critic was trained from replay data. However, this also fails without seeing on-policy data.

The results with sparse rewards are very encouraging. We are able to stack 3 blocks with a fully sparse reward without resetting to the states from demonstrations, and 4 blocks with a fully sparse reward if we use resetting. With resets from demonstration states and the step reward, we are able to learn a policy to stack 6 blocks.

4.8 ABLATION EXPERIMENTS

In this section we perform a series of ablation experiments to measure the importance of various components of our method. We evaluate our method on stacking 3 to 6 blocks.

We perform the following ablations on the best performing of our models on each task:

consistent with our collective experience training these models. We report the median of the random seeds everywhere applicable.

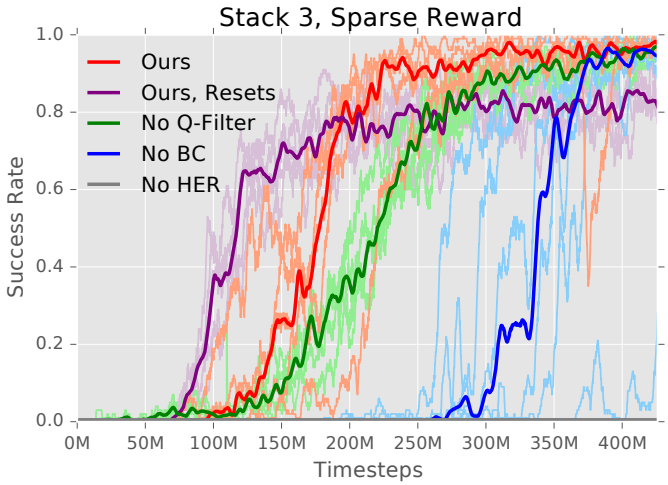


Figure 16: Ablation results on stacking 3 blocks with a fully sparse reward. We run each method 5 times with random seeds. The bold line shows the median of the 5 runs while each training run is plotted in a lighter color. Note "No HER" is always at 0% success rate. Our method without resets learns faster than the ablations. Our method with resets initially learns faster but converges to a worse success rate.

No BC Loss: This method does not apply the behavior cloning gradient during training. It still has access to demonstrations through the demonstration replay buffer.
No Q-Filter: This method uses standard behavioral cloning loss instead of the loss from equation Eq. 15, which means that the actor tries to mimic the demonstrator's behaviour regardless of the critic.
No HER: Hindsight Experience Replay is not used.

4.8.1 *Behavior Cloning Loss*

Without the behavior cloning loss, the method is significantly worse in every task we try. Fig. 16 shows the training curve for learning to stack 3 blocks with a fully sparse reward. Without the behavior cloning loss, the system is about 2x slower to learn. On longer horizon tasks, we do not achieve any success without this loss.

To see why, consider the training curves for stacking 4 blocks shown in Fig. 17. The "No BC" policy learns to stack only one additional block. Without the behavior cloning loss, the agent only has access to the demonstrations through the demonstration replay buffer. This allows it to view high-reward states and incentivizes the agent to stack more

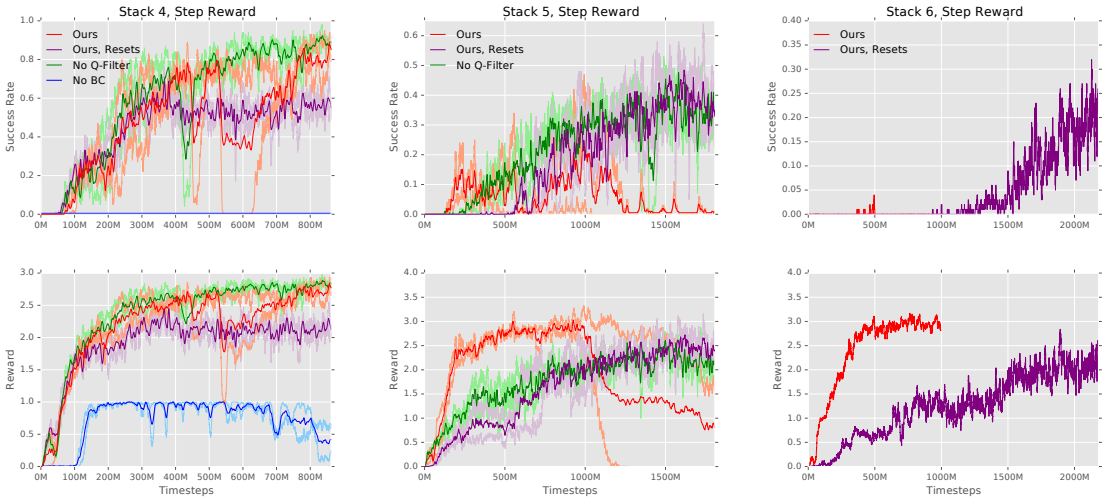


Figure 17: Ablation results on longer horizon tasks with a step reward. The upper row shows the success rate while the lower row shows the average reward at the final step of each episode obtained by different algorithms. For stacking 4 and 5 blocks, we use 2 random seeds per method. The median of the runs is shown in bold and each training run is plotted in a lighter color. Note that for stacking 4 blocks, the "No BC" method is always at 0% success rate. As the number of blocks increases, resets from demonstrations becomes more important to learn the task.

blocks, but there is a stronger disincentive: stacking the tower higher is risky and could result in lower reward if the agent knocks over a block that is already correctly placed. Because of this risk, which is fundamentally just another instance of the agent finding a local optimum in a shaped reward, the agent learns the safer behavior of pausing after achieving a certain reward. Explicitly weighting behavior cloning steps into gradient updates forces the policy to continue the task.

4.8.2 *Q-Filter*

The Q-Filter is effective in accelerating learning and achieving optimal performance. Fig. 16 shows that the method without filtering is slower to learn. One issue with the behavior cloning loss is that if the demonstrations are suboptimal, the learned policy will also be suboptimal. Filtering by Q-value gives a natural way to anneal the effect of the demonstrations as it automatically disables the BC loss when a better action is found. However, it gives mixed results on the longer horizon tasks. One explanation is that

in the step reward case, learning relies less on the demonstrations because the reward signal is stronger. Therefore, the training is less affected by suboptimal demonstrations.

4.8.3 *Resets From Demonstrations*

We find that initializing rollouts from within demonstration states greatly helps to learn to stack 5 and 6 blocks but hurts training with fewer blocks, as shown in Fig. 17. Note that even where resets from demonstration states helps the final success rate, learning takes off faster when this technique is not used. However, since stacking the tower higher is risky, the agent learns the safer behavior of stopping after achieving a certain reward. Resetting from demonstration states alleviates this problem because the agent regularly experiences higher rewards.

This method changes the sampled state distribution, biasing it towards later states. It also inflates the Q values unrealistically. Therefore, on tasks where the RL algorithm does not get stuck in solving a subset of the full problem, it could hurt performance.

4.9 DISCUSSION AND FUTURE WORK

We present a system to utilize demonstrations along with reinforcement learning to solve complicated multi-step tasks. We believe this can accelerate learning of many tasks, especially those with sparse rewards or other difficulties in exploration. Our method is very general, and can be applied on any continuous control task where a success criterion can be specified and demonstrations obtained.

An exciting future direction is to train policies directly on a physical robot. Fig. 14 shows that learning the pick-and-place task takes about 1 million timesteps, which is about 6 hours of real world interaction time. This can realistically be trained on a physical robot, short-cutting the simulation-reality gap entirely. Many automation tasks found in factories and warehouses are similar to pick-and-place but without the variation in initial and goal states, so the samples required could be much lower. With our method, no expert needs to be in the loop to train these systems: demonstrations can be collected by users without knowledge about machine learning or robotics and rewards could be directly obtained from human feedback.

A major limitation of this work is sample efficiency on solving harder tasks. While we could not solve these tasks with other learning methods, our method requires a large amount of experience which is impractical outside of simulation. To run these tasks on physical robots, the sample efficiency will have to improved considerably. We

also require demonstrations which are not easy to collect for all tasks. If demonstrations are not available but the environment can be reset to arbitrary states, one way to learn goal-reaching but avoid using demonstrations is to reuse successful rollouts as in **florensa2017resets**.

Finally, our method of resets from demonstration states requires the ability to reset to arbitrary states. Although we can solve many long-horizon tasks without this ability, it is very effective for the hardest tasks. Resetting from demonstration rollouts resembles curriculum learning: we solve a hard task by first solving easier tasks. If the environment does not afford setting arbitrary states, then other curriculum methods will have to be used.

5

RESIDUAL REINFORCEMENT LEARNING FOR ROBOT CONTROL

5.1 INTRODUCTION

Robots in today's manufacturing environments typically perform repetitive tasks, and often lack the ability to handle variability and uncertainty. Commonly used control algorithms, such as PID regulators and the computed torque method, usually follow predefined trajectories with little adaptive behavior. Many manufacturing tasks require some degree of adaptability or feedback to the environment, but significant engineering effort and expertise is required to design feedback control algorithms for these industrial robots. The engineering time for fine tuning such a controller might be similar in cost to the robot hardware itself. Being able to quickly and easily design feedback controllers for industrial robots would significantly broaden the space of manufacturing tasks that can be automated by robots.

Why is designing a feedback controller for many tasks hard with classical methods? While conventional feedback control methods can solve tasks such as path following efficiently, applications that involve contacts between the robot and its environment are difficult to approach with conventional control methods. Identifying and characterizing contacts and friction is difficult—even if a physical model provides reasonable contact behavior, identifying the physical parameters of a contact interaction accurately is very hard. Hence, it is often difficult to achieve adaptable yet robust control behavior, and significant control tuning effort is required as soon as these elements are introduced. Another drawback of conventional control methods is their lack of behavior generalization. Thus, all possible system behaviors must be considered a priori at design time.

Reinforcement learning (RL) methods hold the promise of solving these challenges because they allow agents to learn behaviors through interaction with their surrounding environments and ideally generalize to new scenarios that differ from the specifications at the control design stage. Moreover, RL can handle control problems that are difficult

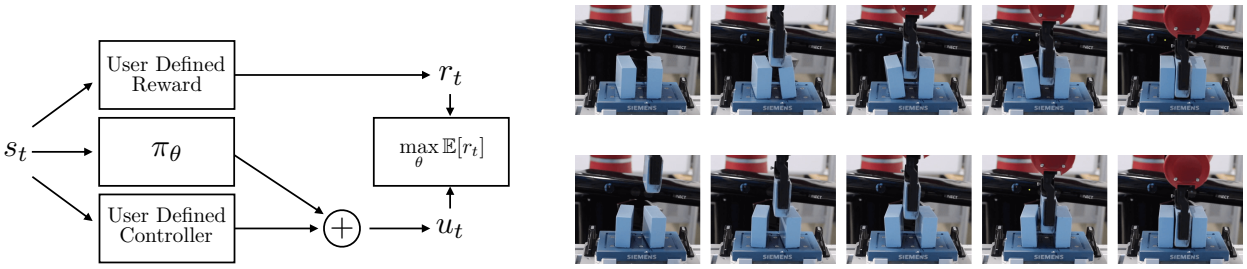


Figure 18: We train an agent directly in the real world to solve a model assembly task involving contacts and unstable objects. An outline of our method, which consists of combining hand-engineered controllers with a residual RL controller, is shown on the left. Rollouts of residual RL solving the block insertion task are shown on the right. Residual RL is capable of learning a feedback controller that adapts to variations in the orientations of the standing blocks and successfully completes the task of inserting a block between them. Videos are available at residualrl.github.io

.

to approach with conventional controllers because the control goal can be specified indirectly as a term in a reward function and not explicitly as the result of a control action. All of these aspects are considered enablers for truly autonomous manufacturing systems and important for fully flexible lot-size one manufacturing **lotsizeone**. However, standard RL methods require the robot learn through interaction, which can be unsafe initially, and collecting the amount of interaction that is needed to learn a complex skill from scratch can be time consuming.

In this paper, we study control problems that are difficult to approach with conventional feedback control methods. However, the problems possess structure that can be partially handled with conventional feedback control, e.g. with impedance control. The residual part of the control task, which is the part that must consider contacts and external object dynamics, is solved with RL. The outputs of the conventional controller and RL are superposed to form the commanded control. The main contribution of this paper is a methodology that combines conventional feedback control with deep RL methods and is illustrated in Fig. 18. Our main motivation is a control approach that is suitable for real-world control problems in manufacturing, where the exploratory behavior of RL is a safety concern and the data requirements of deep RL can be expensive. We provide a thorough evaluation of our method on a block assembly task in simulation and on physical hardware. When the initial orientation of the blocks is noisy, our hand-designed controller fails to solve the task, while residual RL successfully learns to perform the task in under 3 hours. This suggests that we could usefully apply our method to practi-

cal manufacturing problems.

5.2 PRELIMINARIES

In this section, we set up our problem and summarize the foundations of classical control and reinforcement learning that we build on in our approach.

5.2.1 *Problem Statement - System Theoretic Interpretation*

The class of control problems that we are dealing with in this paper can be viewed from a dynamical systems point of view as follows. Consider a dynamical system that consists of a fully actuated robot and underactuated objects in the robot's environment. The robot and the objects in its environment are described by their states s_m and s_o, respectively. The robot can be controlled through the control input u while the objects cannot be directly controlled. However, the robot's states are coupled with the objects' states so that indirect control of s_o is possible through u. This is for example the case if the agent has large inertia and is interacting with small parts as is common in manufacturing. The states of agent and objects can either be fully observable or they can be estimated from measurements.

The time-discrete equations of motion of the overall dynamical system comprise the robot and objects and can be stated as

$$s_{t+1} = \begin{bmatrix} s_{m,t+1} \\ s_{o,t+1} \end{bmatrix} = \begin{bmatrix} A(s_{m,t}) & 0 \\ B(s_{m,t}, s_{o,t}) & C(s_{o,t}) \end{bmatrix} \begin{bmatrix} s_{m,t} \\ s_{o,t} \end{bmatrix} + D \begin{bmatrix} u_t \\ 0 \end{bmatrix}, \tag{19}$$

where the states can also be subject to algebraic constraints, which we do no state explicitly here.

The type of control objectives that we are interested in can be summarized as controlling the agent in order to manipulate the objects while also fulfilling a geometric objective such as trajectory following. It is difficult to solve the control problem directly with conventional feedback control approaches, which compute the difference between a desired and a measured state variable. In order to achieve best system performance feedback control methods require well understood and modeled state transition dynamics. Finding the optimal control parameters can be difficult or even impossible if the system dynamics are not fully known.

In equation 19 the state transition matrices although $A(s_m)$ and $C(s_o)$ are usually

known to a certain extent, because they represent rigid body dynamics, the coupling matrix $B(s_m, s_o)$ is usually not known. Physical interactions such as contacts and friction forces are the dominant effects that $B(s_m, s_o)$ needs to capture, which also applies to algebraic constraints, which are functions of s_m and s_o as well. Hence, conventional feedback control synthesis for determining u to control s_o is very difficult, and requires trial and error in practice. Another difficulty for directly designing feedback controllers is due to the fact that, for many control objectives, the states s_o need to fulfill conditions that cannot be expressed as deviations (errors) from desired states. This is often the case when we only know the final goal rather than a full trajectory.

Instead of directly designing a feedback control system, we can instead specify the goal via a reward function. These reward functions can depend on both s_m and s_o, where the terms that depend on s_m are position related objectives.

5.2.2 *Interpretation as a Reinforcement Learning Problem*

In reinforcement learning, we consider the standard Markov decision process framework for picking optimal actions to maximize rewards over discrete timesteps in an environment E. At every timestep t, an agent is in a state s_t, takes an action u_t, receives a reward r_t, and E evolves to state s_{t+1}. In reinforcement learning, the agent must learn a policy $u_t = \pi(s_t)$ to maximize expected returns. We denote the return by $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$, where T is the horizon that the agent optimizes over and γ is a discount factor for future rewards. The agent's objective is to maximize expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_0]$.

Unlike the previous section, RL does not attempt to model the unknown coupled dynamics of the agent and the object. Instead, it finds actions that maximizes rewards, without making any assumptions about the system dynamics. In this paper, we use value-based RL methods. These methods estimate the state-action value function:

$$Q^{\pi}(s_t, u_t) = \mathbb{E}_{r_i, s_i \sim E, u_i \sim \pi}[R_t | s_t, u_t] \tag{20}$$

$$= \mathbb{E}_{r_t, s_{t+1} \sim E}[r_t + \gamma \mathbb{E}_{u_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, u_{t+1})]] \tag{21}$$

Equation 21 is a recursive version of Equation 20, and is known as the Bellman equation. The Bellman equation allows us to estimate Q via approximate dynamic programming. Value-based methods can be learned off-policy, making them very sample efficient, which is vital for real-world robot learning.

Based on the analysis in Sec. 9.3, we introduce a control system that consists of two parts. The first part is based on conventional feedback control theory and maximize all reward terms that are functions of s_m. An RL method is superposed and maximizes the reward terms that are functions of s_o.

5.3.1 *Residual Reinforcement Learning*

In most robotics tasks, we consider rewards of the form:

$$r_t = f(s_m) + g(s_o). \tag{22}$$

The term $f(s_m)$ is assumed to be a function, which represents a geometric relationship of robot states, such as a Euclidean distance or a desired trajectory. The second term of the sum $g(s_o)$ can be a general class of functions. Concretely, in our model assembly task, $f(s_m)$ is the reward for moving the robot gripper between the standing blocks, while $g(s_o)$ is the reward for keeping the standing blocks upright and in their original positions.

The key insight of residual RL is that in many tasks, $f(s_m)$ can be easily optimized a priori of any environment interaction by conventional controllers, while $g(s_o)$ may be easier to learn with RL which can learn fine-grained hand-engineered feedback controllers even with friction and contacts. To take advantage of the efficiency of conventional controllers but also the flexbility of RL, we choose:

$$u = \pi_H(s_m) + \pi_\theta(s_m, s_o) \tag{23}$$

as the control action, where $\pi_H(s_m)$ is the human-designed controller and $\pi_\theta(s_m, s_o)$ is a learned policy parametrized by θ and optimized by an RL algorithm to maximize expected returns on the task.

Inserting equation 28 into equation 19 one can see that a properly designed feedback control law for $\pi_H(s_m)$ is able to provide exponentially stable error dynamics of s_m if the learned controller π_θ is neglected and the sub statespace is stabilizable. This is equivalent to maximizing equation 22 for the case f represents errors between actual and desired states.

The residual controller $\pi_\theta(s_m, s_o)$ can now be used to maximize the reward term $g(s_o)$ in equation 22. Since the control sequence equation 28 enters equation 19 through the

dynamics of s_m and s_m is in fact the control input to the dynamics of s_o, we cannot simply use the a-priori hand-engineered feedback controller to achieve zero error of s_m and independently achieve the control objective on s_o. Through the coupling of states we need to perform an overall optimization of equation 28, whereby the hand-engineered feedback controller provides internal structures and eases the optimization related to the reward term $f(s_m)$.

Algorithm 3 Residual reinforcement learning

Require: policy π_θ, hand-engineered controller π_H.

for $n = 0, ..., N-1$ episodes **do**
  Initialize random process $\mathcal{N}$ for exploration
  Sample initial state $s_0 \sim E$.
  for $t = 0, ..., H-1$ steps **do**
    Get policy action $u_t = \pi_\theta(s_t) + \mathcal{N}_t$.
    Get action to execute $u'_t = u_t + \pi_H(s_t)$.
    Get next state $s_{t+1} \sim p(\cdot \mid s_t, u'_t)$.
    Store (s_t, u_t, s_{t+1}) into replay buffer $\mathcal{R}$.
    Sample set of transitions $(s, u, s') \sim \mathcal{R}$.
    Optimize θ using RL with sampled transitions.
  end for
end for=0

5.3.2 *Method Summary*

Our method is summarized in Algorithm 4. The key idea is to combine the flexibility of RL with the efficiency of conventional controllers by additively combining a learnable parametrized policy with a fixed hand-engineered controller.

As our underlying RL algorithm, we use a variant of twin delayed deep deterministic policy gradients (TD3) as described in Fujimoto et al., 2018. TD3 is a value-based RL algorithm for continuous control based off of the deep deterministic policy gradient (DDPG) algorithm Lillicrap et al., 2016. We have found that TD3 is stable, sample-efficient, and requires little manual tuning compared to DDPG. We used the publicly available `rlkit` implementation of TD3 Pong et al., 2018. Our method is independent of the choice of RL algorithm, and we could apply residual RL to any other RL algorithm.

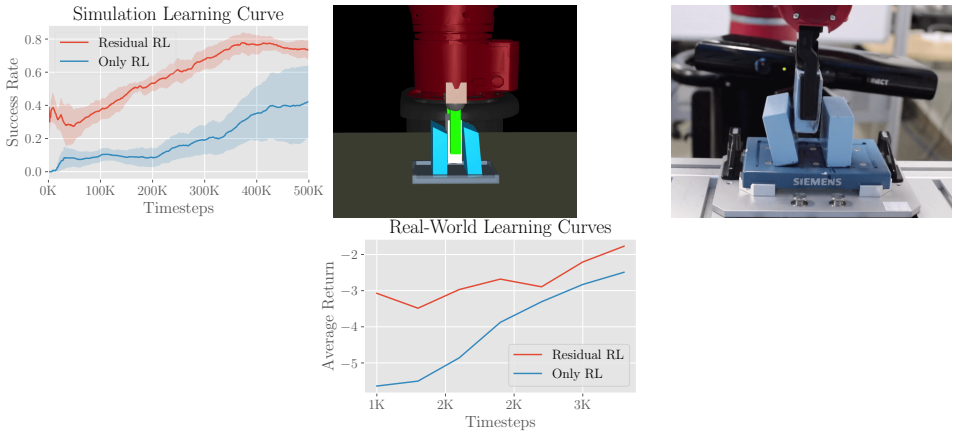


Figure 19: Block assembly task in simulation (left) and real-world (right). The task is to insert a block between the two blocks on the table without moving the blocks or tipping them over. In the learning curves, we compare our method with RL without any hand-engineered controller[1]. In both simulation and real-world experiments, we see that residual RL learns faster than RL alone, while achieving better performance than the hand-engineered controller.

5.4 EXPERIMENTAL SETUP

We evaluate our method on the task shown in Fig. 19, both in simulation and in the real world. This section introduces the details of the experimental setup and provides an overview of the experiments.

5.4.1 *Simulated Environment*

We use MuJoCo Todorov et al., 2012, a full-featured simulator for model-based optimization considering body contacts, to evaluate our method in simulation. This environment consists of a simulated Sawyer robot arm with seven degrees of freedom and a parallel gripper. We command the robot with a Cartesian-space position controller.

5.4.2 *Real-World Environment*

The real-world environment is largely the same as the simulated environment, except for the controller, rewards, and observations. We command the robot with a compliant joint-space impedance controller we have developed to be smooth and tolerant of contacts. The positioning of the block being inserted is similar to the simulation but the

observation is estimated from a camera-based tracking system as we do not have access to ground truth position information.

5.4.3 *Overview of Experiments*

In our experiments we evaluate the following research questions:

1. Does incorporating a hand-designed controller improve the performance and sample-efficiency of RL algorithms, while still being able to recover from an imperfect hand-designed controller?
2. Can our method allow robots to be more tolerant of variation in the environment?
3. Can our method successfully control noisy systems, compared to classical control methods?

5.5 EXPERIMENTS

5.5.1 *Sample Efficiency of Residual RL*

In this section, we compare our residual RL method with the human controller alone and RL alone. The following methods are compared:

1. Only RL: using the same underlying RL algorithm as our method but without adding a hand-engineered policy
2. Residual RL: our method which trains a superposition of the hand-engineered controller and a neural network policy, with RL

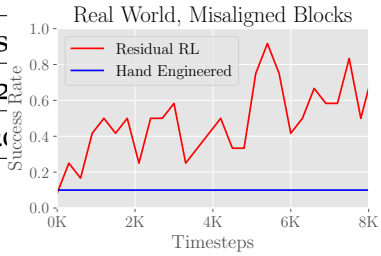
5.5.2 *Effect of Environment Variation*

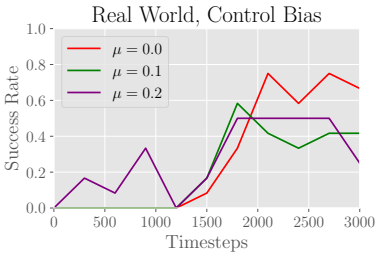
In automation, environments can be subject to noise and solving manufacturing tasks become more difficult as variability in the environment increases. It is difficult to manually design feedback controllers that are robust to environment variation, as it might require significant human expertise and tuning. In this experiment, we vary the initial orientation of the blocks during each episode and demonstrate that residual RL can still solve the task. We compare its performance to that of the hand-engineered controller.

To introduce variability in simulation, on every reset we sampled the rotation of each block independently from a uniform distribution $U[-r, r], r \in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$.

1 In all simulation plots, we use 10 random seeds and report a 95% confidence interval for the mean.

Misaligned?	No	Yes
Residual RL	20/20	15/2
Hand-engineered	20/20	2/2





(a) (b) (c)

Figure 20: Outcome of our residual RL method in different experiments during the block assembly task in the real-world. Success rate is recorded manually by human judgment of whether the blocks stayed upright and ended in the correct position. Plot (a) compares the insertion success of residual RL and hand-designed controller depending on the block orientation during run time. Plot (b) shows the success rate of the insertion process during training, where on every reset the blocks are randomly rotated: straight, tilted clockwise, or tilted counterclockwise ($\pm$ 20°) and plot (c) shows the increasing success rate of our method for biased controllers as well even as control bias increases.

Similarly, in the real world experiments, on every reset we randomly rotated each block to one of three orientations: straight, tilted clockwise, or tilted counterclockwise (tilt was $\pm$ 20° from original position).

5.5.3 *Recovering from Control Noise*

Due to a host of issues, such as defective hardware or poorly tuned controller parameters, feedback controllers might have induced noise. Conventional feedback control policies are determined a priori and do not adapt their behavior from data. However, RL methods are known for their ability to cope with shifting noise distributions and are capable of recovering from such issues.

In this experiment, we introduce a control noise, including biased control noise, and demonstrate that residual RL can still successfully solve the task, while a hand-engineered controller cannot. The control noise follows a normal distribution and is added to the control output of the system at every step:

$$u_t' = u_t + \mathcal{N}(\mu, \sigma^2) \tag{24}$$

To test tolerance to control noise, we set $\mu = 0$ and vary $\sigma \in [0.01, 0.1]$. In theory, RL could adapt to a noisy controller by learning more robust solutions to the task which are

less sensitive to perturbations.

Furthermore, to test tolerance to a biased controller, we set $\sigma = 0.05$ and vary $\mu \in [0, 0.2]$. To optimize the task reward, RL can learn to simply counteract the bias.

5.5.4 *Sim-to-Real with Residual RL*

As an alternative to analytic solutions of real-world control problems, we can often instead model the forward dynamics of the problem (ie. a simulator). With access to such a model, we can first find a solution to the problem with our possibly inaccurate model, and then use residual RL to find a realistic control solution in the real world.

In this experiment, we attempt the block insertion task with the side blocks fixed in place. The hand-engineered policy π_H in this case comes from training a parametric policy in simulation of the same scenario (with deep RL). We then use this policy as initialization for residual RL in the real world.

5.6 RESULTS

We trained our method to optimize the insertion process in simulation as well as on physical hardware. This section provides the results of our discussed experiments and shows the functionality of our method.

5.6.1 *Sample Efficiency of Residual RL*

First, we compare residual RL and pure RL without a hand-engineered controller on the insertion task. Fig. 19 shows in simulation and real-world that residual RL achieves a better final performance and requires less samples than RL alone, both in simulation and on physical hardware. Unlike residual RL, the pure RL approach needs to learn the structure of the position control problem from scratch, which explains the difference in sample efficiency. As samples are expensive to collect in the real world, residual RL is better suited for solving real-world tasks. Moreover, RL shows a broader spatial variance during training and needs to explore a wider set of states compared to residual RL, which can be potentially dangerous in hardware deployments.

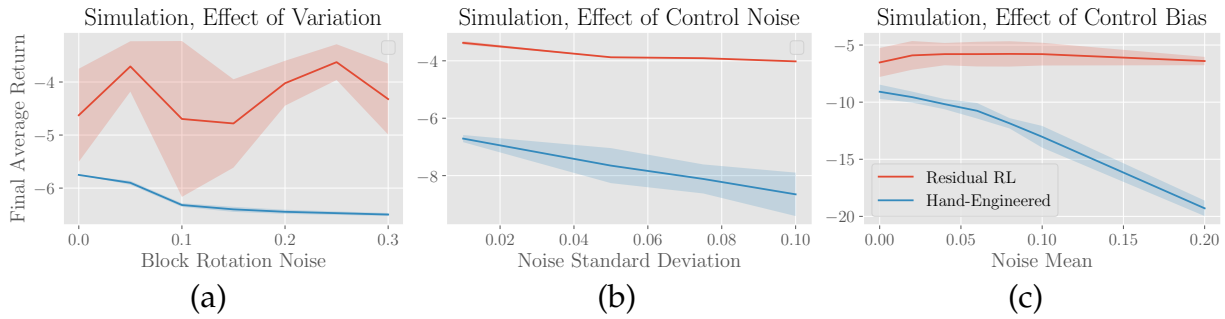


Figure 21: Simulation results for different experiments. In each plot, the final average return obtained by running the method for various settings of a parameter is shown. Plot (a) shows that residual RL can adjust to noise in the environment caused by rotation of the blocks in a range of 0 to 0.3 rad. In plot (b), residual RL finds robust strategies in order to reduce the effect of control noise, as the final average return is not greatly affected by the magnitude of noise. Plot (c) shows that residual RL can compensate for biased controllers and maintains good performance as control bias increases, while the performance of the hand-designed controller dramatically deteriorates with higher control bias.

5.6.2 *Effect of Environment Variation*

In previous set of experiments, both standing blocks were placed in their initial position without any position or orientation error. In this case, the hand-engineered controller performs well, as both blocks are placed such that there is a sufficiently large defined gap for insertion. However, once the initial orientation of the blocks is randomized, the gap between the blocks and the goal position does not afford directly inserting from above. Therefore, the hand-engineered controller struggles to solve the insertion task, succeeding in only 2/20 trials, while residual RL still succeeds in 15/20 trials. These results are summarized in Fig. 20 (a). Rollouts from the learned policy are included in 18. In this experiment, the agent demonstrably learns consistent small corrective feedback behaviors in order to slightly nudge the blocks in the right direction without tipping them over, a behavior that is very difficult to manually specify.

The result of this experiment showcases the strength of residual RL. Since the human controller specifies the general trajectory of the optimal policy, environment samples are required only to learn this corrective feedback behavior. The real-world learning curve for the experiment in Fig. 20 (b) shows that this behavior is gradually acquired over the course of eight thousand samples, which is only about three hours of real-world training time.

We further studied the effect of the block orientation changing after every reset in

simulation. The results are shown in Fig. 21 (a). The simulation results show that the performance of the hand-engineered controller decreases as the block rotation angle increases, whereas our control method maintains a constant average performance over different variations.

5.6.3 *Recovering from Control Noise*

In this experiment, we observe that residual RL is able to cope with actuator noise, including biased actuator noise. Quantitative results for simulation are shown in Fig. 21 (b) and (c). In Fig. 21 (c) our method keeps the average return constant and correct for biased controllers even as control bias increases, whereas the hand-engineered controller cannot compensate biased input and its performance deteriorates as control bias increases. The same applies for adding control noise to the control output as shown in Fig. 21 (b).

For the hardware experiments, only biased actuator noise is investigated. These results are shown in Fig. 20 (c). These learning curves show that even as more control bias is introduced, training in the real world proceeds without significant issues. This result suggests the potential for RL to address practical issues in automation such as sensor drift.

5.6.4 *Sim-to-Real with Residual RL*

The result of the sim-to-real experiment is shown in Fig. 22. In this experiment, each setting was run with three random seeds. Adding policy initialization from simulation significantly speeds up both RL and residual RL. In particular, residual RL with policy initialization from simulation successfully solves the task extremely quickly: in under one thousand timesteps of environment interaction. This method poses a highly sample efficient, practical way to solve robotics problems with difficult contact dynamics.

5.7 RELATED WORK

Reinforcement learning for robotics holds the promise of greater autonomy and reliability, which could be vital to improving our manufacturing processes beyond its current limitations. RL methods have been difficult to apply in robotics because of sample efficiency, safety, and stability issues. Still, RL has been used to allow robots to learn tasks such as playing table tennis **peters2010reps**, swinging up a cartpole and balancing a unicycle **deisenroth2011pilco**, grasping **pinto2017robust**; Levine et al., 2017, opening a

door **Gu2016b**, and general manipulation tasks **haarnoja2018sac**; Levine et al., 2016. RL, particularly deep RL, tends to be data-hungry; even learning simple tasks can require many hours of interaction. To bring these methods into factories and warehouses, they must be able to consistently solve complex tasks, multi-step tasks. One way to enable these methods to solve these complex tasks is to introduce prior human knowledge into the learning system, as our method does.

Prior work in RL has incorporated human prior knowledge for solving tasks in various ways. One such way is reward shaping **ng1999rewardshaping**, where additional rewards auxiliary to the real objective are included in order to guide the agent towards the desired behavior. Reward shaping can effectively encode a policy. For example, to train an agent to perform block stacking, each step can be encoded into the reward **popov17stacking**.

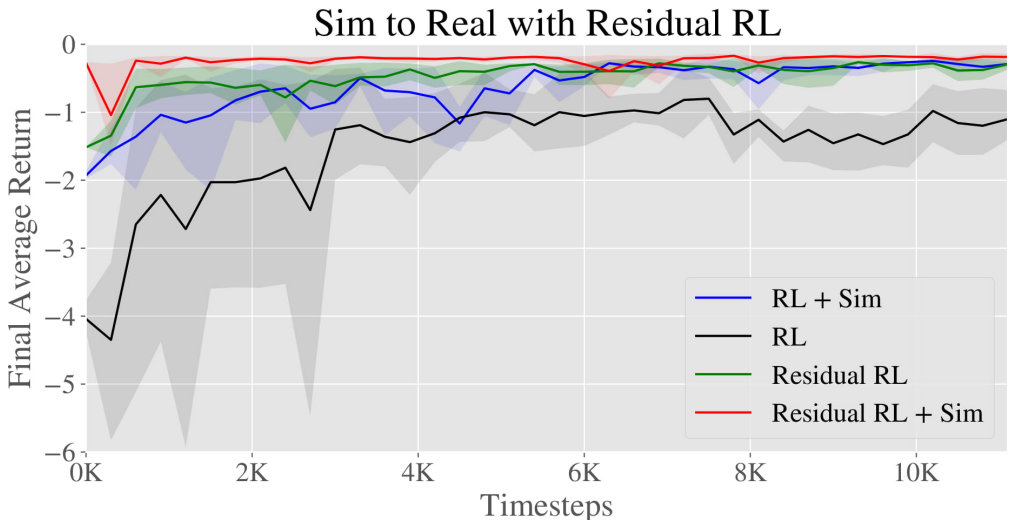


Figure 22: Real-world block insertion results using residual RL for sim-to-real transfer. "Sim" indicates that the real-world policy was initialized by reinforcement learning in simulation. Residual RL with simulation initialization successfully solves the task with little environment experience required.

Often, intensive reward shaping is key for RL to succeed at a task and prior work has even considered reward shaping as part of the learning system **daniel2014activereward**. Reward shaping in order to tune agent behavior is a very manual process and recovering a good policy with reward shaping can be as difficult as specifying the policy itself. Hence, in our method we allow for human specification of both rewards and policies—whichever might be more practical for a particular task.

Further work has incorporated more specialized human knowledge into RL systems. One approach is to use trajectory planners in RL in order to solve robotics tasks **thomas2018cad**. However, since the method optimizes trajectory following instead of the task reward, generalization can be difficult when aspects of the environment change. Other work has focused on human feedback **loftin2014discretehumanfeedback**; **saunders2018trialwithouterror**; **torrey2013teachingbudget**; **frank2008rareevents**; **christiano2017humanpreferences** to inform the agent about rewards or to encourage safety. However, in many robotics tasks, providing enough information about the task through incremental human feedback is difficult.

Another way to include prior knowledge in RL is through demonstrations **peters2008baseball**; **kober2008mp**; **rajeswaran2018dextrous**; **hester17dqfd**; **vecerik17ddpgfd**; **nair2018demonstrations**. Demonstrations can substantially simplify the exploration problem as the agent begins training having already received examples of high-reward transitions and therefore knows where to explore **subramanian2016efd**. However, providing demonstrations requires humans to be able to teleoperate the robot to perform the task. In contrast, our method only requires a conventional controller for motion, which ships with most robots.

Prior knowledge can also be induced through neural network architecture choices. Deep residual networks with additive residual blocks achieved state of the art results in many computer vision tasks by enabling training of extremely deep networks **he2016resnet**. In RL, structured control nets showed improvement on several tasks by splitting the policy into a linear module and a non-linear module **srouji18structuredcontrolnets**. Prior work in adaptive flight control has also considered compensating linearized controllers with neural networks **johnson2000hedging**. Most closely related to our work, residual policy learning concurrently and independently explores training a residual policy in the context of simulated long-horizon, sparse-reward tasks with environment variation and sensor noise **silver18residualpolicylearning**. Our work instead focuses on achieving practical real-world training of contact-intensive tasks.

5.8 CONCLUSION

In this paper we study the combination of conventional feedback control methods with deep RL. We presented a control method that utilizes conventional feedback control along with RL to solve complicated manipulation tasks involving friction and contacts with unstable objects. We believe this approach can accelerate learning of many tasks, especially those where the control problem can be solved in large part by prior knowledge but requires some model-free reasoning to solve perfectly. Our results demonstrate that the combination of conventional feedback control and RL can circumvent the disadvantages of both and provide a sample efficient controller that can cope with contact dynamics.

6

DEEP REINFORCEMENT LEARNING FOR INDUSTRIAL INSERTION TASKS WITH VISUAL INPUTS AND NATURAL REWARDS

6.1 INTRODUCTION

Many industrial tasks on the edge of automation require a degree of adaptability that is difficult to achieve with conventional robotic automation techniques. While standard control methods, such as PID controllers, are heavily employed to automate many tasks in the context of positioning, tasks that require significant adaptability or tight visual perception-control loops are often beyond the capabilities of such methods, and therefore are typically performed manually. Standard control methods can struggle in presence of complex dynamical phenomena that are hard to model analytically, such as complex contacts. Reinforcement learning (RL) offers a different solution, relying on trial and error learning instead of accurate modeling to construct an effective controller. RL with expressive function approximation, i.e. deep RL, has further shown to automatically handle high dimensional inputs such as images **mnih2013atari**.

However, deep RL has thus far not seen wide adoption in the automation community due to several practical obstacles. Sample efficiency is one obstacle: tasks must be completed without excessive interaction time or wear and tear on the robot. Progress in recent years on developing better RL algorithms has led to significantly better sample efficiency, even in dynamically complicated tasks **haarnoja2018sac**; **hessel2018rainbow**, but remains a challenge for deploying RL in real-world robotics contexts. Another major, often underappreciated, obstacle is goal specification: while prior work in RL assumes a reward signal to optimize, it is often carefully shaped to allow the system to learn **ng1999rewardshaping**; **popov17stacking**; **daniel2014activereward**. Obtaining such dense reward signals can be a significant challenge, as one must additionally build a perception system that allows computing dense rewards on state representations. Shaping a

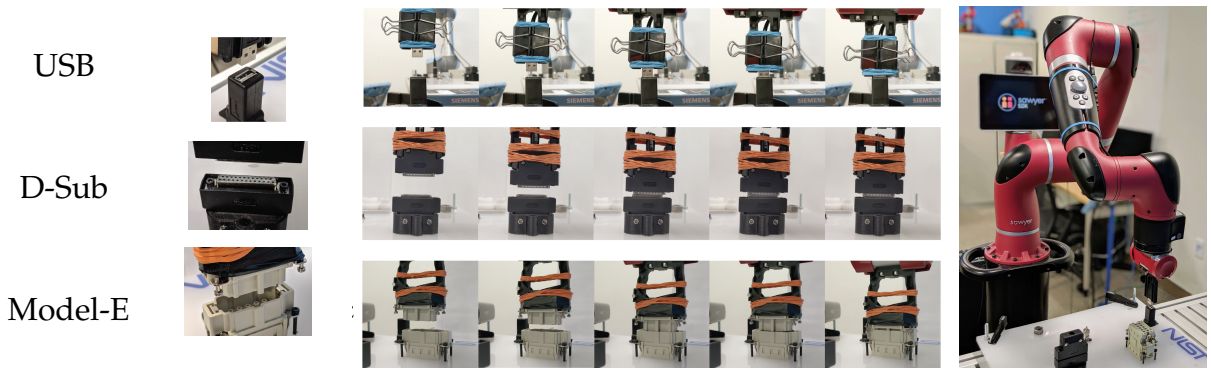


Figure 23: We train policies directly in the real world to solve connector insertion tasks from raw pixel input and without access to ground-truth state information for reward functions. Left: top-down views of the connectors. Middle: a rollout from a learned policy that successfully completes the insertion task for each connector is shown. Right: a full view of the robot setup. Videos of the results are available at industrial-insertion-rl.github.io

reward function so that an agent can learn from it is also a manual process that requires considerable manual effort. An ideal RL system would learn from rewards that are natural and easy to specify. How can we enable robots to autonomously perform complex tasks without significant engineering effort to design perception and reward systems?

We first consider an end-to-end approach that learns a policy from images, where the images serve as both the state representation and the goal specification. Using goal images is not fully general, but can successfully represent tasks when the task is to reach a final desired state **nair2018rig**. Specifying goals via goal images is convenient, and makes it possible to specify goals with minimal manual effort. Using images as the state representation also allows a robot to learn behaviors that utilize direct visual feedback, which provides some robustness to sensor and actuator noise.

Secondly, we consider learning from simple and sparse reward signals. Sparse rewards can often be obtained conveniently, for instance from human-provided labels or simple instrumentation. In many electronic assembly tasks, which we consider here, we can directly detect whether the electronics are functional, and use that signal as a reward. Learning from sparse rewards poses a challenge, as exploration with sparse reward signals is difficult, but by using sufficient prior information about the task, one can overcome this challenge. To handle this challenge, we extend the residual RL approach **johannink18residualrl**; **silver18residualpolicylearning**, which learns a parametric policy on top of a fixed, hand-specified controller, to the setting of vision-based manipulation.

In our experiments, we show that we can successfully complete real-world tight tol-

erance assembly tasks, such as inserting USB connectors, using RL from images with reward signals that are convenient for users to specify. We can learn from only a sparse reward based on the electrical connection for a USB adapter plug, and we demonstrate learning insertion skills with rewards based only on goal images. These reward signals require no extra engineering and are easy to specify for many tasks. Beyond showing the feasibility of RL for solving these tasks, we evaluate multiple RL algorithms across three tasks and study their robustness to imprecise positioning and noise.

6.2 RELATED WORK

Learning has been applied previously in a variety of robotics contexts. Different forms of learning have enabled autonomous driving **pomerleau1989alvinn**, biped locomotion **nakanishi2004bipedlfd**, block stacking **deisenroth2011stacking**, grasping Pinto and Gupta, 2016, and navigation **giusti15trails**; Pathak et al., 2018. Among these methods, many involve reinforcement learning, where an agent learns to perform a task by maximizing a reward signal. Reinforcement learning algorithms have been developed and applied to teach robots to perform tasks such as balancing a robot **deisenroth2011pilco**, playing ping-pong **peters2010reps** and baseball **peters2008baseball**. The use of large function approximators, such as neural networks, in RL has further broadened the generality of RL **mnih2013atari**. Such techniques, called "deep" RL, have further allowed robots to be trained directly in the real world to perform fine-grained manipulation tasks from vision Levine et al., 2016, open doors **gu2016naf**, play hockey **chebotar2017pilqr**, stack Lego blocks **zhang2019solar**, use dexterous hands **zhu2019hands**, and grasp objects **kalashnikov2018qtopt**. In this work we further explore solving real-world robotics tasks using RL.

Many RL algorithms introduce prior information about the specific task to be solved. One common method is reward shaping **ng1999rewardshaping**, but reward shaping can become arbitrarily difficult as the complexity of the task increases. Other methods incorporate a trajectory planner **thomas2018cad** but for complex assembly tasks, trajectory planners require a host of information about objects and geometries which can be difficult to provide.

Another body of work on incorporating prior information studies using demonstrations either to initialize a policy **peters2008baseball**; **kober2008mp**, infer reward functions using inverse reinforcement learning **finn16guidedcostlearning**; **ziebart2008maxent** or to improve the policy throughout the learning procedure **hester17dqfd**; **nair2018demonstrations**; **rajeswaran2018dextrous**. These methods require multiple demonstrations, which can

be difficult to collect, especially for assembly tasks, although learning a reward function by classifying goal states **singh2019raq** may partially alleviate this issue. More recently, manually specifying a policy and learning the residual task has been proposed **johannink18residualrl**; **silver18residualpolicylearning**. In this work we evaluate both residual RL and combining RL with learning from demonstrations.

Previous work has also tackled high precision assembly tasks, especially insertion-type tasks. One line of work focuses on obtaining high dimensional observations, including geometry, forces, joint positions and velocities **li2014usbgelsight**; **tamar2017hindsightplan**; **inoue2017deeprlassembly**; **lu019variableimpedance**, but this information is not easily procured, increasing complexity of the experiments and the supervision required. Other work relies on external trajectory planning or very high precision control **inoue2017deeprlassembly**; **tamar2017hindsightplan**, but this can be brittle to error in other components of the system, such as perception. We show how our method not only solves insertion tasks with much less information about the environment, but also does so under noisy conditions.

6.3 ELECTRIC CONNECTOR PLUG INSERTION TASKS

In this work, we empirically evaluate learning methods on a set of electric connector assembly tasks, pictured in Fig. 23. Connector plug insertions are difficult for two reasons. First, the robot must be very precise in lining up the plug with its socket. As we show in our experiments, errors as small as ± 1 mm can lead to consistent failure. Second, there is significant friction when the connector plug touches the socket, and the robot must learn to apply sufficient force in order to insert the plug. Image sequences of successful insertions are shown in Fig. 23, where it is also possible to see details of the gripper setup that we used to ensure a failure free, fully automated training process. In our experiments, we use a 7 degrees of freedom Sawyer robot with end-effector control, meaning that the action signal u_t can be interpreted as the relative end-effector movement in Cartesian coordinates. The robot's underlying internal control pipeline is illustrated in Figure 24.

To comprehensively evaluate connector assembly tasks, we experiment on a variety of connectors. Each connector offers a different challenge in terms of required precision and force to overcome friction. We chose to benchmark the controllers performance on the insertion of a USB connector, a U-Sub connector, and

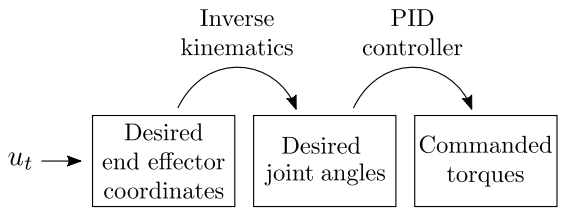


Figure 24: Illustration of the robot's cascade control scheme. The actions u_t are computed at a frequency of up to 10 Hz, desired joint angles are obtained by inverse kinematics, and a joint-space impedance controller with anti-windup PID control commands actuator torques at 1000 Hz.

a waterproof Model-E connector manufactured by MISUMI. All the explored use cases were part of the IROS 2017 Robotic Grasping and Manipulation Competition **roboticgrasping2017iros**, included as part of a task board developed by NIST to benchmark the performance of assembly robots.

6.3.1 *Adapters*

In the following we describe the used adapters, USB, D-Sub, and Model-E. The observed difficulty of the insertion increases in that order.

USB. The USB connector is a ubiquitous, widely-used connector and offers a challenging insertion task. Because the adapter becomes smoother and therefore easier to insert over time due to wear and tear, we periodically replace the adapter. Of the three tested adapters, the USB adapter is the easiest.

D-sub. Inserting this adapter requires aligning several pins correctly, and is therefore more sensitive than inserting the USB adapter. It also requires more downward force due to a tighter fit.

Model-E. This adapter is the most difficult of the three tested connectors as it contains several edges and grooves to align and requires significant downward force to successfully insert the part.

6.3.2 *Experimental Settings*

We consider three settings in our experiments in order to evaluate how plausible it is to solve these tasks with more convenient state representations and reward functions and to evaluate the performance of different algorithms changes as the setting is modified.

3.2.1 Visual. In this experiment, we evaluate whether the RL algorithms can learn to perform the connector assembly tasks from vision without having access to state information. The state provided to the learned policy is a 32×32 grayscale image, such as shown in Fig. 25. For goal

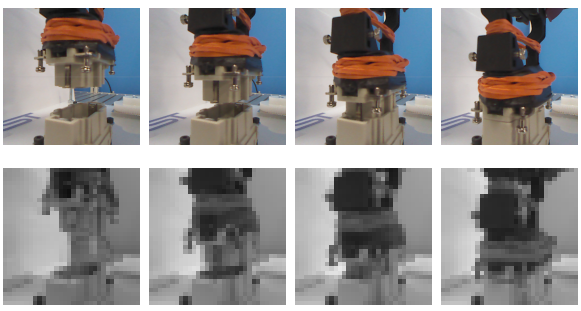


Figure 25: Successful insertion on the Model-E connector. The image-based RL algorithms receives only receives the 32×32 grayscale image as the observation.

specification, we use a goal image, avoiding the need for state information to compute rewards. The reward is the pixelwise L1 distance to the given goal image. Being able to learn from such a setup is compelling as it does not require any extra state estimation and many tasks can be specified easily by a goal image.

3.2.2. Sparse. In this experiment, the reward is obtained by directly measuring whether the connection is alive and transmitting:

$$r = \begin{cases} 1, & \text{if insertion signal detected} \\ 0, & \text{else.} \end{cases} \tag{25}$$

This is the exact true reward for the task of connecting a cable, and can be naturally obtained in many manufacturing systems. As state, the robot is given the Cartesian coordinates of the end-effector x_t and the vertical force f_z that is acting on the end-effector. We could only automatically detect the USB connection, so we only include the USB adapter for the sparse experiments.

3.2.3. Dense. In this experiment, the robot receives a manually shaped reward based on the distance to the target location x^*. We use the reward function

$$r_t = -\alpha \cdot \|x_t - x^*\|_1 - \frac{\beta}{(\|x_t - x^*\|_2 + \varepsilon)} - \varphi \cdot f_z, \tag{26}$$

where $0 < \varepsilon \ll 1$. The hyperparameters are set to $\alpha = 100$, $\beta = 0.002$, and $\varphi = 0.1$. When an insertion is indicated through a distance measurement, the sign of the force term flips, so that $\varphi = -0.1$ when the connector is inserted. This rewards the agent for pressing down after an insertion and showed to improve the learning process. The force measurements are calibrated before each rollout to account for measurement bias and to decouple the measurements from the robot pose.

6.4 METHODS

To solve the connector insertion tasks, we consider and evaluate a variety of RL algorithms.

6.4.1 *Preliminaries*

In a Markov decision process (MDP), an agent at every time step is at state $s_t \in \mathcal{S}$, takes actions $u_t \in \mathcal{U}$, receives a reward $r_t \in \mathbb{R}$, and the state evolves according to environment transition dynamics $p(s_{t+1}|s_t, u_t)$. The goal of reinforcement learning is to choose actions $u_t \sim \pi(u_t|s_t)$ to maximize the expected returns $\mathbb{E}[\sum_{t=0}^{H} \gamma^t r_t]$ where H is the horizon and γ is a discount factor. The policy $\pi(u_t|s_t)$ is often chosen to be an expressive parametric function approximator, such as a neural network, as we use in this work.

6.4.2 *Efficient Off-Policy Reinforcement Learning*

One class of RL methods additionally estimates the expected discounted return after taking action u from state s, the Q-value $Q(s, u)$. Q-values can be recursively defined with the Bellman equation:

$$Q(s_t, u_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{u_{t+1}} Q(s_{t+1}, u_{t+1})] \tag{27}$$

and learned from off-policy transitions (s_t, u_t, r_t, s_{t+1}). Because we are interested in sample-efficient real-world learning, we use such RL algorithms that can take advantage of off-policy data.

For control with continuous actions, computing the required maximum in the Bellman equation is difficult. Continuous control algorithms such as deep deterministic policy gradients (DDPG) Lillicrap et al., 2016 additionally learn a policy $\pi_\theta(u_t|s_t)$ to approximately choose the maximizing action. In this paper we specifically consider two related reinforcement learning algorithms that lend themselves well to real-world learning as they are sample efficient, stable, and require little hyperparameter tuning.

Twin Delayed Deep Deterministic Policy Gradients (TD3). Like DDPG, TD3 optimizes a deterministic policy Fujimoto et al., 2018 but uses two Q-function approximators to reduce value overestimation **vanhasselt2016doubledqn** and delayed policy updates to stabilize training.

Soft Actor Critic (SAC). SAC is an off-policy value-based reinforcement learning method based on the maximum entropy reinforcement learning framework with a stochastic policy **haarnoja2018sac**.

We used the implementation of these RL algorithms publicly available at `rlkit` Pong et al., 2018.

Algorithm 4 Residual reinforcement learning

Require: policy π_θ, hand-engineered controller π_H.

for $n = 0, ..., N-1$ episodes **do**
  Sample initial state $s_0 \sim E$.
  for $t = 0, ..., H-1$ steps **do**
    Get policy action $u_t \sim \pi_\theta(u_t|s_t)$.
    Get action to execute $u'_t = u_t + \pi_H(s_t)$.
    Get next state $s_{t+1} \sim p(\cdot \mid s_t, u'_t)$.
    Store (s_t, u_t, s_{t+1}) into replay buffer $\mathcal{R}$.
    Sample set of transitions $(s, u, s') \sim \mathcal{R}$.
    Optimize θ using RL with transitions.
  end for
end for=0

6.4.3 *Residual Reinforcement Learning*

Instead of randomly exploring from scratch, we can inject prior information into an RL algorithm in order to speed up the training process, as well as to minimize unsafe exploration behavior. In residual RL, actions u_t are chosen by additively combining a fixed policy $\pi_H(s_t)$ with a parametric policy $\pi_\theta(u_t|s_t)$:

$$u_t = \pi_H(s_t) + \pi_\theta(s_t). \tag{28}$$

The parameters θ can be learned using any RL algorithm. In this work, we evaluate both SAC and TD3, explained in the previous section. The residual RL implementation that we use in our experiments is summarized in Algorithm 4.

A simple P-controller serves as the hand-designed controller π_H of our experiments. The P-controller operates in Cartesian space and calculates the current control action by

$$\pi_H(s_t) = -k_p \cdot (x_t - x^*), \tag{29}$$

where x^* denotes the commanded goal location. As control gains we use $k_p = [1, 1, 0.3]$. This P-controller quickly centers the end-effector above the goal position and reaches the goal after about 10 time steps from the reset position, which is located 5cm above the

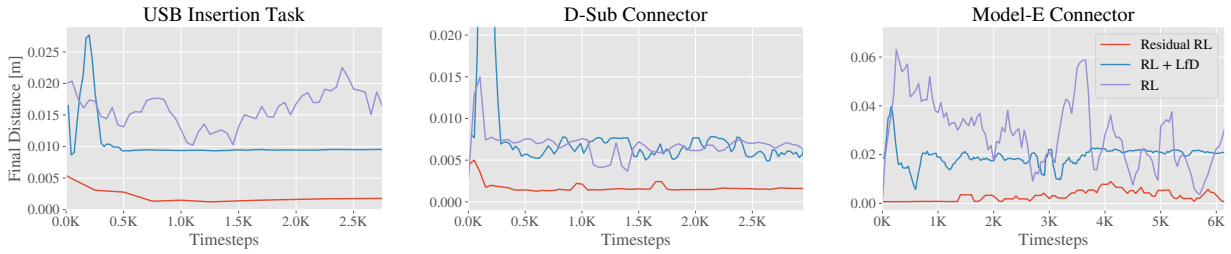


Figure 26: Resulting final mean distance during the vision-based training. The comparison includes RL, residual RL, and RL with learning from demonstrations. Only residual RL manages to deal with the high-dimensional input and consistently solve all the tasks after the given amount of training. The other methods learn to move downwards, but often get stuck in the beginning of the insertion and fail to recover from unsuccessful attempts.

goal.

6.4.4 *Learning from Demonstrations*

Another method to incorporate prior information is to use demonstrations from an expert policy to guide exploration during RL. We first collected demonstrations with a joystick controller. Then, we add a behavior cloning loss while performing RL that pushes the policy towards the demonstrator actions, as previously considered in **nair2018demonstrations**. Instead of DDPG, the underlying algorithm RL algorithm used is TD3.

6.5 EXPERIMENTS

We evaluate our method, which combines residual RL with easy-to-obtain reward signals, on a variety of connector assembly tasks performed on a real robot. In this section, we consider two types of natural rewards that are intuitive for users to provide: an image directly specifying a goal and a binary sparse reward indicating success. For both cases, we report success rates on tasks they solve. We aim to answer the following questions: (1) Can such trained policies provide comparable performance to policies that are trained with densely-shaped rewards? (2) Are these trained policies robust to small variations and noise?

5.1 Vision-based Learning. For the vision-based learning experiments, we use only raw image observations and ℓ_1 distance between the current image and goal image as the reward signal. Sample images that the robot received are shown in Fig. 25. We evaluate this type of reward on all three connectors. In our experiments, we use 32×32 grayscale images.

5.2 Learning from Sparse Rewards. In the sparse reward experiment, we use the binary signal of the connector being electrically connected as the reward signal. This experiment is most applicable to electronic manufacturing settings where the electrical connection between connectors can be directly measured. We only evaluate the sparse reward setting on the USB connector, as it was straightforward to obtain the electrical connection signal.

5.3 Perfect State Information. After evaluating the tasks in the above settings, we further evaluate with full state information with a dense and carefully shaped reward signal, given in Eq. 26, that incorporates distance to the goal and force information. Evaluating in this setting gives us an "oracle" that can be compared to the previous experiments in order to understand how much of a challenge sparse or image rewards pose for various algorithms.

5.4 Robustness. For safe and reliable future usage, it is required that the insertion

controller is robust against small measurement or calibration errors that can occur when disassembling and reassembling a mechanical system. In this experiment, small goal perturbations are introduced in order to uncover the required setup precision of our algorithms.

5.5 Exploration Comparison. One advantage of using reinforcement learning is the exploratory behavior that allows the controller to adapt from new experiences unlike a deterministic control law. The two RL algorithms we consider in this paper, SAC and TD3, explore differently. SAC maintains a stochastic policy, and the algorithm also adapts the stochasticity through training. TD3 has a deterministic policy, but uses another noise process (in our case Gaussian) to inject exploratory behavior during training time. We compare the two algorithms, as well as when they are used in conjunction with residual RL, in order to evaluate the effect of the different exploration schemes.

6.6 RESULTS

We analyze the performance of policies learned with residual RL, as well as other methods, based on their ability to achieve the task goal, as well as the distance of the final object location to the goal pose over the course of training. To study the robustness of the learned policies, we also evaluate them in conditions where the goal connector position is perturbed, in order to understand the tolerance of RL policies to imprecise object placement.

6.6.1 *Vision-based Learning*

The results of the vision-based experiment are shown in Fig. 26. Our experiments show that a successful and consistent vision-based insertion policy can be learned from relatively few samples using residual RL.

This result suggests that goal-specification through images is a practical way to solve these types of industrial tasks. Although image-based rewards are often very sparse and hard to learn from, in this case the distance between images corresponds to a relatively dense reward signal which is sufficient to distinguish the different stages of the insertion process.

USB Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	28%	20%
	Sparse, SAC	16%	8%
	Sparse, TD3	44%	28%
	Images, SAC	36%	32%
	Images, TD3	28%	28%
RL + LfD	Sparse	**100%**	32%
	Images	88%	60%
Residual RL	Dense	**100%**	**84%**
	Sparse, SAC	88%	**84%**
	Sparse, TD3	**100%**	36%
	Images, SAC	**100%**	80%

D-Sub Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	16%	0%
	Images, SAC	0%	0%
	Images, TD3	12%	12%
RL + LfD	Images	52%	52%
Residual RL	Dense	**100%**	60%
	Images, SAC	**100%**	**64%**
	Images, TD3	52%	52%
Human	P-Controller	**100%**	44%
Model-E Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	0%	0%
	Images, SAC	0%	0%
	Images, TD3	0%	0%
RL + LfD	Images	20%	20%
Residual RL	Dense	**100%**	**76%**
	Images, SAC	**100%**	**76%**
	Images, TD3	0%	0%
Human	P-Controller	52%	24%

Figure 27: We report average success out of 25 policy executions after training is finished for each method. For noisy goals, noise is added in form of ± 1 mm perturbations of the goal location. Residual RL, particularly with SAC, tends to be the best performing method across all three connectors. For the Model-E connector, only residual RL solves the task in the given amount of training time.

Interestingly, during training with standard RL, the policy would sometimes learn to "hack" the reward signal by moving down in the image in front of or behind the socket. In contrast, the stabilizing human-engineered controller in residual RL provides sufficient horizontal control to prevent this. The initial controller also scaffolds the learning process, by providing a very strong initialization that requires the reinforcement learning algorithm to only learn the final phase of the insertion. This produces substantially better performance in conjunction with vision-based rewards.

6.6.2 *Learning From Sparse Rewards*

In this experiment, we compare these methods on the USB insertion task with sparse rewards. The results are reported in Fig. **??**. All methods are able to achieve very high success rates in the sparse setting. This result shows that we can learn precise industrial insertion tasks in sparse-reward settings, which can often be obtained much more easily than a dense, shaped reward. In fact, prior work has found that the final policy for sparse rewards can outperform the final policy for dense rewards as it does not suffer from a misspecified objective Andrychowicz et al., 2017.

6.6.3 *Perfect State Information*

The results of the experiment with perfect state information and dense rewards is shown in Fig. 30. In this case, residual RL still outperforms standard RL, though the better-shaped reward enables standard RL to make more initial progress than with the other reward signals. However, the hand-designed shaped reward function makes it harder for the policy to actually perform the full insertion, potentially because the more complex reward land-

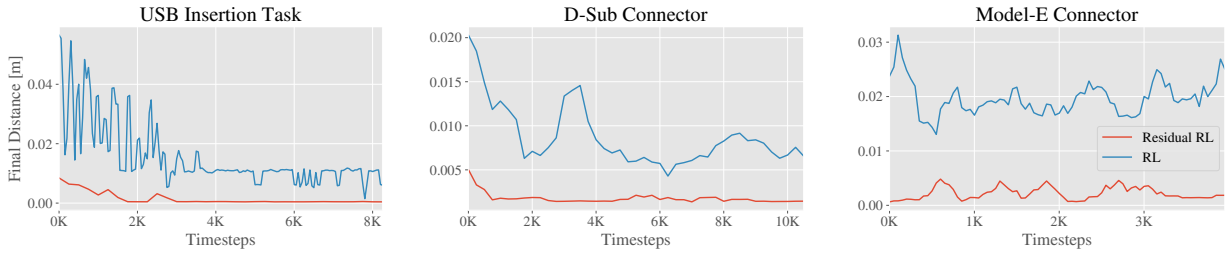


Figure 30: Plots of the final mean distance to the goal during the state-based training. Final distances greater than 0.01 m indicate unsuccessful insertions. Here, the residual RL approach performs noticeably better than pure RL and is often able to solve the task during the exploration in the early stages of the training.

scape provides other competing goals to the policy. The final performance with sparse rewards on the USB insertion task is substantially better.

6.6.4 *Robustness*

In the previous set of experiments, the goal locations were known exactly. In this case, the hand-engineered controller performs well. However, once noise is added to the goal location, the deterministic P-controller struggles. To test robustness, a goal perturbation is created artificially, and the controllers are tested under this condition. All results of our robustness evaluations are listed in Fig. 27 and Fig. 28. In the presence of a $\pm$1mm perturbation, the residual RL controller succeeds more often on the USB and D-Sub tasks, and significantly more often on the Model-E task. Unlike the hand-engineered controller, residual RL consistently solved this task and overcame goal perturbations in 16/25 trials. The agent demonstrably learns small but consistent corrective feedback behaviors in order to move in the right direction during the descent motion, a behavior that is very difficult to specify manually. This behavior illustrates the strength of residual RL. Since the human controller already specifies the general trajectory of the optimal policy, environment samples are only required to learn this corrective feedback behavior.

6.6.5 *Exploration Comparison*

All experiments were also performed using TD3 instead of SAC. The final success rates of these experiments are included in Fig. 27. When combined with residual RL, SAC and TD3 perform comparably. However, TD3 is often substantially less robust. These results are likely explained by the exploration strategy of the two algorithms. TD3 has a deterministic policy and fixed noise during training, so once it observes some high-reward states, it quickly learns to repeat that trajectory. SAC adapts the noise to the correct scale, helping SAC stay robust to small perturbations, and because SAC learns the value function for a stochastic policy, it is able to handle some degree of additive noise effectively. We found that the outputted action of TD3 approaches the extreme values at the edge of the allowed action space, while SAC executed less extreme actions, which likely further improved robustness.

6.7 CONCLUSION

In this paper, we studied deep reinforcement learning in a practical setting, and demonstrated that deep RL can solve complex industrial assembly tasks with tight tolerances. We showed that we can learn insertion policies with raw image observations with either binary outcome-based rewards, or rewards based on on goal images. We conducted a series of experiments for various connector type assemblies, and demonstrated the feasibility of our method under challenging conditions, such as noisy goal specification and complex connector geometries. Reinforcement learning algorithms that can automatically learn complex assembly tasks with easy-to-specify reward functions have the potential to automate a wide range of assembly tasks, making this technology a promising direction forward for flexible and capable robotic manipulators.

There remains significant challenges for applying these techniques in more complex environments. One practical direction for future work is focusing on multi-stage assembly tasks through vi-

sion. This would pose a challenge to the goal-based policies as the background would be visually more complex. Moreover, multi-step tasks involve adapting to previous mistakes or inaccuracies, which could be difficult but should be able to be handled by RL. Extending the presented approach to multi-stage assembly tasks will pave the road to a higher robot autonomy in flexible manufacturing.

7

AWAC: ACCELERATING ONLINE REINFORCEMENT LEARNING WITH OFFLINE DATASETS

7.1 INTRODUCTION

Learning models that generalize effectively to complex open-world settings, from image recognition (Krizhevsky et al., 2012) to natural language processing (**devlin2019bert**), relies on large, high-capacity models as well as large, diverse, and representative datasets. Leveraging this recipe of pre-training from large-scale offline datasets has the potential to provide significant benefits for reinforcement learning (RL) as well, both in terms of generalization and sample complexity. But most existing RL algorithms collect data online from scratch every time a new policy is learned, which can quickly become impractical in domains like robotics where physical data collection has a non-trivial cost. In the same way that powerful models in computer vision and NLP are often pre-trained on large, general-purpose datasets and then fine-tuned on task-specific data, practical instantiations of reinforcement learning for real world robotics problems will need to be able to incorporate large amounts of prior data effectively into the learning process, while still collecting additional data online for the task at hand. Doing so effectively will make the online data collection process much more practical while still allowing robots operating in the real world to continue improving their behavior.

For data-driven reinforcement learning, offline datasets consist of trajectories of states, actions and associated rewards. This data can potentially come from demonstrations for the desired task (**schaal97lfd**; **atkeson1997lfd**), suboptimal policies (**ga02018imperfect**), demonstrations for related tasks (**zhou2019wtl**), or even just random exploration in the environment. Depending on the quality of the data that is provided, useful knowledge can be extracted about the dynamics of the world, about the task being solved, or both. Effective data-driven methods for deep reinforcement learning should be able to use this data to pre-train offline while improving with online fine-tuning.

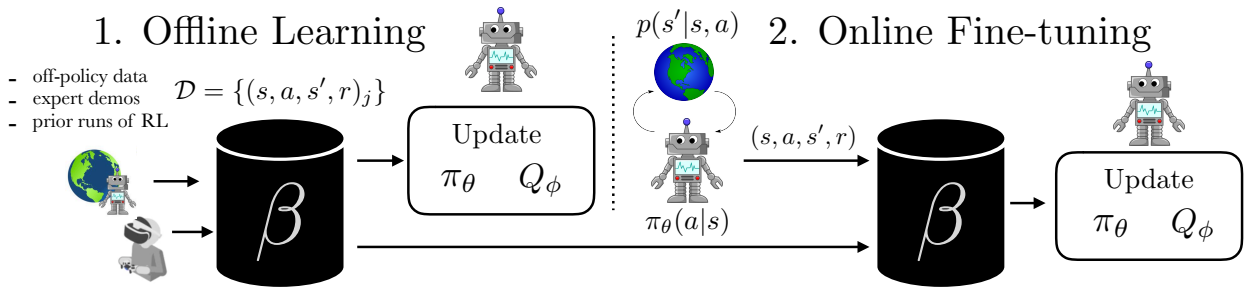


Figure 31: We study learning policies by offline learning on a prior dataset $\mathcal{D}$ and then fine-tuning with online interaction. The prior data could be obtained via prior runs of RL, expert demonstrations, or any other source of transitions. Our method, advantage weighted actor critic (AWAC) is able to learn effectively from offline data and fine-tune in order to reach expert-level performance after collecting a limited amount of interaction data. Videos and data are available at *awacrl.github.io*

Since this prior data can come from a variety of sources, we would like to design an algorithm that does not utilize different types of data in any privileged way. For example, prior methods that incorporate demonstrations into RL directly aim to mimic these demonstrations (**nair2018demonstrations**), which is desirable when the demonstrations are known to be optimal, but imposes strict requirements on the type of offline data, and can cause undesirable bias when the prior data is not optimal. While prior methods for fully offline RL provide a mechanism for utilizing offline data (**fujimoto19bcq**; **kumar19bear**), as we will show in our experiments, such methods generally are not effective for fine-tuning with online data as they are often too conservative. In effect, prior methods require us to choose: Do we assume prior data is optimal or not? Do we use only offline data, or only online data? To make it feasible to learn policies for open-world settings, we need algorithms that learn successfully in any of these cases.

In this work, we study how to build RL algorithms that are effective for pre-training from off-policy datasets, but also well suited to continuous improvement with online data collection. We systematically analyze the challenges with using standard off-policy RL algorithms (**haarnoja2018sac**; **kumar19bear**; **we2018mpo**) for this problem, and introduce a simple actor critic algorithm that elegantly bridges data-driven pre-training from offline data and improvement with online data collection. Our method, which uses dynamic programming to train a critic but a supervised learning style update to train a constrained actor, combines the best of supervised learning and actor-critic algorithms. Dynamic programming can leverage off-policy data and enable sample-efficient learning. The simple supervised actor update implicitly enforces a constraint that mitigates the effects of distribution shift when learning from offline data (**fujimoto19bcq**; **kumar19bear**), while avoiding overly conservative updates.

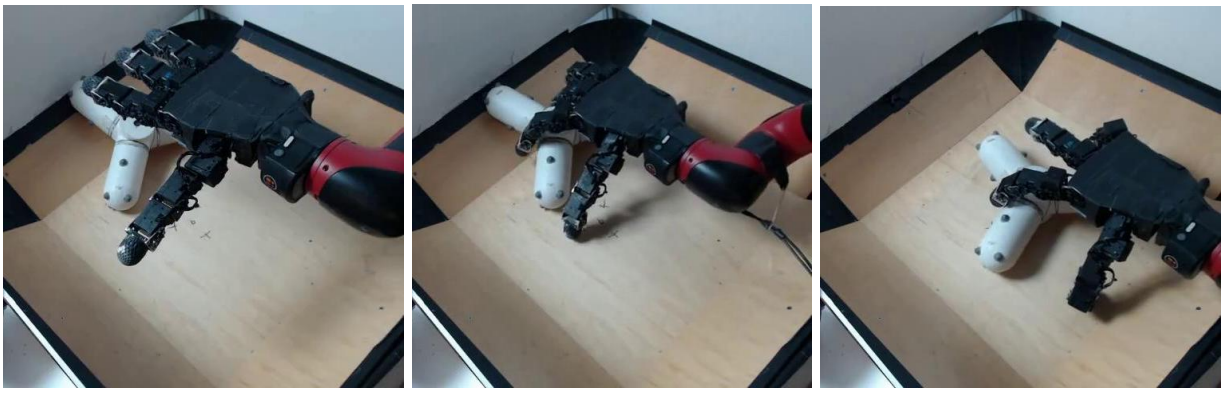


Figure 32: Utilizing prior data for online learning allows us to solve challenging real-world robotics tasks, such as this dexterous manipulation task where the learned policy must control a 4-fingered hand to reposition an object.

We evaluate our algorithm on a wide variety of robotic control tasks, using a set of simulated dexterous manipulation problems as well as three separate real-world robots: drawer opening with a 7-DoF robotic arm, picking up an object with a multi-fingered hand, and rotating a valve with a 3-fingered claw. Our algorithm, Advantage Weighted Actor Critic (AWAC), is able to quickly learn successful policies for these challenging tasks, in spite of high dimensional action spaces and uninformative, sparse reward signals. We show that AWAC finetunes much more efficiently after offline pretraining as compared to prior methods and, given a fixed time budget, attains significantly better performance on the real-world tasks. Moreover, AWAC can utilize different types of prior data without any algorithmic changes: demonstrations, suboptimal data, or random exploration data. The contribution of this work is not just another RL algorithm, but a systematic study of what makes offline pre-training with online fine-tuning unique compared to the standard RL paradigm, which then directly motivates a simple algorithm, AWAC, to address these challenges. We additionally discuss the design decisions required for applying AWAC as a practical tool for real-world robotic skill learning.

7.2 PRELIMINARIES

We use the standard reinforcement learning notation, with states $\mathbf{s}$, actions $\mathbf{a}$, policy $\pi(\mathbf{a}|\mathbf{s})$, rewards $r(\mathbf{s}, \mathbf{a})$, and dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. The discounted return is defined as $R_t = \sum_{i=t}^{T} \gamma^i r(\mathbf{s}_i, \mathbf{a}_i)$, for a discount factor γ and horizon T which may be infinite. The objective of an RL agent is to maximize the expected discounted return $J(\pi) = \mathbb{E}_{p_\pi(\tau)}[R_0]$ under the distribution induced by the policy. The optimal policy can be learned directly (e.g., using

policy gradient to estimate $\nabla J(\pi)$ (**williams1992reinforce**)), but this is often ineffective due to high variance of the estimator. Many algorithms attempt to reduce this variance by making use of the value function $V^\pi(\mathbf{s}) = \mathbb{E}_{p_\pi(\tau)}[R_t|\mathbf{s}]$, action-value function $Q^\pi(\mathbf{s},\mathbf{a}) = \mathbb{E}_{p_\pi(\tau)}[R_t|\mathbf{s},\mathbf{a}]$, or advantage $A^\pi(\mathbf{s},\mathbf{a}) = Q^\pi(\mathbf{s},\mathbf{a}) - V^\pi(\mathbf{s})$. The action-value function for a policy can be written recursively via the Bellman equation:

$$Q^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')] \tag{30}$$

$$= r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}',\mathbf{a}')]]. \tag{31}$$

Instead of estimating policy gradients directly, actor-critic algorithms maximize returns by alternating between two phases (**konda2000actorcritic**): policy evaluation and policy improvement. During the policy evaluation phase, the critic $Q^\pi(\mathbf{s},\mathbf{a})$ is estimated for the current policy π. This can be accomplished by repeatedly applying the Bellman operator $\mathcal{B}$, corresponding to the right-hand side of Equation 31, as defined below:

$$\mathcal{B}^\pi Q(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}',\mathbf{a}')]]. \tag{32}$$

By iterating according to $Q^{k+1} = \mathcal{B}^\pi Q^k$, Q^k converges to Q^π (**sutton1998rl**). With function approximation, we cannot apply the Bellman operator exactly, and instead minimize the Bellman error with respect to Q-function parameters ϕ_k:

$$\phi_k = \arg\min_\phi \ \mathbb{E}_\mathcal{D}[(Q_\phi(\mathbf{s},\mathbf{a}) - y)^2], \tag{33}$$

$$y = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}',\mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}',\mathbf{a}')]. \tag{34}$$

During policy improvement, the actor π is typically updated based on the current estimate of Q^π. A commonly used technique (**haarnoja2018sac**; Lillicrap et al., 2016; Fujimoto et al., 2018) is to update the actor $\pi_{\theta_k}(\mathbf{a}|\mathbf{s})$ via likelihood ratio or pathwise derivatives to optimize the following objective, such that the expected value of the Q-function Q^π is maximized:

$$\theta_k = \arg\max_\theta \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s},\mathbf{a})]] \tag{35}$$

Actor-critic algorithms are widely used in deep RL (**haarnoja2018sac**; Mnih et al., 2016; Lillicrap et al., 2016; Fujimoto et al., 2018). With a Q-function estimator, they can in principle utilize off-policy data when used with a replay buffer for storing prior transition tuples, which we will denote β, to sample previous transitions, although we show that

this by itself is insufficient for our problem setting.

7.3 CHALLENGES IN OFFLINE RL WITH ONLINE FINE-TUNING

In this section, we study the unique challenges that exist when pre-training using offline data, followed by fine-tuning with online data collection. We first describe the problem, and then analyze what makes this problem difficult for prior methods.

7.3.1 *Problem Definition*

A static dataset of transitions, $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$, is provided to the algorithm at the beginning of training. This dataset can be sampled from an arbitrary policy or mixture of policies, and may even be collected by a human expert. This definition is general and encompasses many scenarios: learning from demonstrations, random data, prior RL experiments, or even from multi-task data. Given the dataset $\mathcal{D}$, our goal is to leverage $\mathcal{D}$ for pre-training and use a small amount of online interaction to learn the optimal policy $\pi^*(\mathbf{a}|\mathbf{s})$, as depicted in Fig 31. This setting is representative of many real-world RL settings, where prior data is available and the aim is to learn new skills efficiently. We first study existing algorithms empirically in this setting on the HalfCheetah-v2 Gym environment[1]. The prior dataset consists of 15 demonstrations from an expert policy and 100 suboptimal trajectories sampled from a behavioral clone of these demonstrations. All methods for the remainder of this paper incorporate the prior dataset, unless explicitly labeled "scratch".

7.3.2 *Data Efficiency*

One of the simplest ways to utilize prior data such as demonstrations for RL is to pre-train a policy with imitation learning, and fine-tune with on-policy RL (**gupta2019relay**; **rajeswaran2018dextrous**). This approach has two drawbacks: (1) prior data may not be optimal; (2) on-policy fine-tuning is data inefficient as it does not reuse the prior data in the RL stage. In our setting, data efficiency is vital. To this end, we require algorithms that are able to reuse arbitrary off-policy data during online RL for data-efficient fine-tuning. We find that algorithms that use on-policy fine-tuning (**rajeswaran2018dextrous**;

1 We use this environment for analysis because it helps understand and accentuate the differences between different algorithms. More challenging environments like the ones shown in Fig 34 are too hard to solve to analyze variants of different methods.

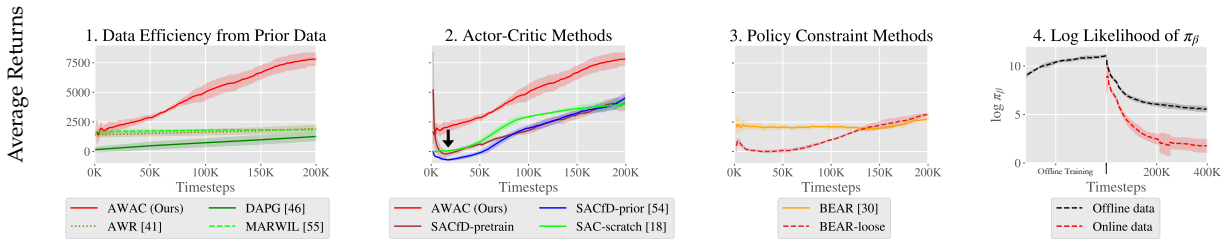


Figure 33: Analysis of prior methods on HalfCheetah-v2 using offline RL with online fine-tuning. (1) On-policy methods (DAPG, AWR, MARWIL) learn relatively slowly, even with access to prior data. We present our method, AWAC, as an example of how off-policy RL methods can learn much faster. (2) Variants of soft actor-critic (SAC) with offline training (performed before timestep 0) and fine-tuning. We see a "dip" in the initial performance, even if the policy is pretrained with behavioral cloning. (3) Offline RL method BEAR (**kumar19bear**) on offline training and fine-tuning, including a "loose" variant of BEAR with a weakened constraint. Standard offline RL methods fine-tune slowly, while the "loose" BEAR variant experiences a similar dip as SAC. (4) We show that the fit of the behavior models $\hat{\pi}_\beta$ used by these offline methods degrades as new data is added to the buffer during fine-tuning, potentially explaining their poor fine-tuning performance.

gupta2019relay)), or Monte-Carlo return estimation (**peters2007rwr**; **wang2018marwil**; **peng2019awr**) are generally much less efficient than off-policy actor-critic algorithms, which iterate between improving π and estimating Q^π via Bellman backups. This can be seen from the results in Figure 33 plot 1, where on-policy methods like DAPG (**rajeswaran2018dextrous** and Monte-Carlo return methods like AWR (**peng2019awr**) and MARWIL (**wang2018marwil**) are an order of magnitude slower than off-policy actor-critic methods. Actor-critic methods, shown in Figure 33 plot 2, can in principle use off-policy data. However, as we will discuss next, naïvely applying these algorithms to our problem suffers from a different set of challenges.

7.3.3 *Bootstrap Error in Offline Learning with Actor-Critic Methods*

When standard off-policy actor-critic methods are applied to this problem setting, they perform poorly, as shown in the second plot in Figure 33: despite having a prior dataset in the replay buffer, these algorithms do not benefit significantly from offline training. We evaluate soft actor critic (**haarnoja2018sac**), a state-of-the-art actor-critic algorithm for continuous control. Note that "SAC-scratch," which does not receive the prior data, performs similarly to "SACfD-prior," which does have access to the prior data, indicating that the off-policy RL algorithm is not actually able to make use of the off-policy data for pre-training. Moreover, even if the SAC is policy is pre-trained by behavior

cloning, labeled "SACfD-pretrain", we still observe an initial decrease in performance, and performance similar to learning from scratch.

This challenge can be attributed to off-policy bootstrapping error accumulation, as observed in several prior works (**sutton1998rl**; **kumar19bear**; **wu2019brac**; **levine2020offlinetutorial**; **fujimoto19bcq**). In actor-critic algorithms, the target value $Q(\mathbf{s}', \mathbf{a}')$, with $\mathbf{a}' \sim \pi$, is used to update $Q(\mathbf{s}, \mathbf{a})$. When $\mathbf{a}'$ is outside of the data distribution, $Q(\mathbf{s}', \mathbf{a}')$ will be inaccurate, leading to accumulation of error on static datasets.

Offline RL algorithms (**fujimoto19bcq**; **kumar19bear**; **wu2019brac**) propose to address this issue by explicitly adding constraints on the policy improvement update (Equation 35) to avoid bootstrapping on out-of-distribution actions, leading to a policy update of this form:

$$\arg\max_{\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s}, \mathbf{a})]] \text{ s.t. } D(\pi_\theta, \pi_\beta) \leqslant \epsilon. \tag{36}$$

Here, π_θ is the actor being updated, and $\pi_\beta(a|s)$ represents the (potentially unknown) distribution from which all of the data seen so far (both offline data and online data) was generated. In the case of a replay buffer, π_β corresponds to a mixture distribution over all past policies. Typically, π_β is not known, especially for offline data, and must be estimated from the data itself. Many offline RL algorithms (**kumar19bear**; **fujimoto19bcq**; **siegel2020abm**) explicitly fit a parametric model to samples for the distribution π_β via maximum likelihood estimation, where samples from π_β are obtained simply by sampling uniformly from the data seen thus far: $\hat{\pi}_\beta = \max_{\hat{\pi}_\beta} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi_\beta}[\log \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})]$. After estimating $\hat{\pi}_\beta$, prior methods implement the constraint given in Equation 36 in various ways, including penalties on the policy update (**kumar19bear**; **wu2019brac**) or architecture choices for sampling actions for policy training (**fujimoto19bcq**; **siegel2020abm**). As we will see next, the requirement for accurate estimation of $\hat{\pi}_\beta$ makes these methods difficult to use with online fine-tuning.

7.3.4 *Excessively Conservative Online Learning*

While offline RL algorithms with constraints (**kumar19bear**; **fujimoto19bcq**; **wu2019brac**) perform well offline, they struggle to improve with fine-tuning, as shown in the third plot in Figure 33. We see that the purely offline RL performance (at "0K" in Fig. 33) is much better than the standard off-policy methods shown in Section 7.3.3. However, with additional iterations of online fine-tuning, the performance increases very slowly (as seen from the slope of the BEAR curve in Fig 33). What causes this phenomenon?

This can be attributed to challenges in fitting an accurate behavior model as data is collected online during fine-tuning. In the offline setting, behavior models must only be trained once via maximum likelihood, but in the online setting, the behavior model must be updated online to track incoming data. Training density models online (in the "streaming" setting) is a challenging research problem (**ramapuram2017lifelonggm**), made more difficult by a potentially complex multi-modal behavior distribution induced by the mixture of online and offline data. To understand this, we plot the log likelihood of learned behavior models on the dataset during online and offline training for the HalfCheetah task. As we can see in the plot, the accuracy of the behavior models ($\log \pi_\beta$ on the y-axis) reduces during online fine-tuning, indicating that it is not fitting the new data well during online training. When the behavior models are inaccurate or unable to model new data well, constrained optimization becomes too conservative, resulting in limited improvement with fine-tuning. This analysis suggests that, in order to address our problem setting, we require an off-policy RL algorithm that constrains the policy to prevent offline instability and error accumulation, but not so conservatively that it prevents online fine-tuning due to imperfect behavior modeling. Our proposed algorithm, which we discuss in the next section, accomplishes this by employing an *implicit* constraint, which does not require *any* explicit modeling of the behavior policy.

7.4 Advantage Weighted Actor Critic: A Simple Algorithm for Fine-Tuning from Offline Datasets

In this section, we will describe the advantage weighted actor-critic (AWAC) algorithm, which trains an off-policy critic and an actor with an *implicit* policy constraint. We will show AWAC mitigates the challenges outlined in Section 7.3. AWAC follows the design for actor-critic algorithms as described in Section 9.3, with a policy evaluation step to learn Q^π and a policy improvement step to update π. AWAC uses off-policy temporal-difference learning to estimate Q^π in the policy evaluation step, and a policy improvement update that is able to obtain the benefits of offline RL algorithms at training from prior datasets, while avoiding the overly conservative behavior described in Section 7.3.4. We describe the policy improvement step in AWAC below, and then summarize the entire algorithm.

Policy improvement for AWAC proceeds by learning a policy that maximizes the value of the critic learned in the policy evaluation step via TD bootstrapping. If done naively, this can lead to the issues described in Section 7.3.4, but we can avoid the challenges of bootstrap error accumulation by restricting the policy distribution to stay close to

the data observed thus far during the actor update, while maximizing the value of the critic. At iteration k, AWAC therefore optimizes the policy to maximize the estimated Q-function $Q^{\pi_k}(\mathbf{s}, \mathbf{a})$ at every state, while constraining it to stay close to the actions observed in the data, similar to prior offline RL methods, though this constraint will be enforced differently. Note from the definition of the advantage in Section 9.3 that optimizing $Q^{\pi_k}(\mathbf{s}, \mathbf{a})$ is equivalent to optimizing $A^{\pi_k}(\mathbf{s}, \mathbf{a})$. We can therefore write this optimization as:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] \tag{37}$$

$$\text{s.t. } D_{KL}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s})) \leqslant \epsilon. \tag{38}$$

As we saw in Section 7.3.3, enforcing the constraint by incorporating an explicit learned behavior model (**kumar19bear**; **fujimoto19bcq**; **wu2019brac**; **siegel2020abm**) leads to poor fine-tuning performance. Instead, we enforce the constraint *implicitly*, without learning a behavior model. We first derive the solution to the constrained optimization in Equation 37 to obtain a non-parametric closed form for the actor. This solution is then projected onto the parametric policy class *without* any explicit behavior model. The analytic solution to Equation 37 can be obtained by enforcing the KKT conditions (**peters2007rwr**; **peters2010reps**; **peng2019awr**). The Lagrangian is:

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] + \lambda(\epsilon - D_{KL}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s}))), \tag{39}$$

and the closed form solution to this problem is $\pi^*(\mathbf{a}|\mathbf{s}) \propto \pi_\beta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right)$. When using function approximators, such as deep neural networks as we do, we need to project the non-parametric solution into our policy space. For a policy π_θ with parameters θ, this can be done by minimizing the KL divergence of π_θ from the optimal non-parametric solution π^* under the data distribution $\rho_{\pi_\beta}(\mathbf{s})$:

$$\arg\min_\theta \; _{\rho_{\pi_\beta}(\mathbf{s})}[D_{KL}(\pi^*(\cdot|\mathbf{s}) || \pi_\theta(\cdot|\mathbf{s}))] \tag{40}$$

$$= \arg\min_\theta \; _{\rho_{\pi_\beta}(\mathbf{s})}\left[_{\pi^*(\cdot|\mathbf{s})}[-\log \pi_\theta(\cdot|\mathbf{s})]\right] \tag{41}$$

Note that the parametric policy could be projected with either direction of KL divergence. Choosing the reverse KL results in explicit penalty methods (**wu2019brac**) that rely on evaluating the density of a learned behavior model. Instead, by using forward KL, we

can compute the policy update by sampling directly from β:

$$\theta_{k+1} = \arg\max_{\theta} \; \mathbf{s},\mathbf{a}\sim\beta \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a}) \right) \right]. \tag{42}$$

This actor update amounts to weighted maximum likelihood (i.e., supervised learning), where the targets are obtained by re-weighting the state-action pairs observed in the current dataset by the predicted advantages from the learned critic, *without* explicitly learning any parametric behavior model, simply sampling (s, a) from the replay buffer β. See Appendix **??** for a more detailed derivation and Appendix **??** for specific implementation details.

Avoiding explicit behavior modeling. Note that the update in Equation 42 completely avoids any modeling of the previously observed data β with a parametric model. By avoiding any explicit learning of the behavior model AWAC is far less conservative than methods which fit a model $\hat{\pi}_\beta$ explicitly, and better incorporates new data during online fine-tuning, as seen from our results in Section 7.6. This derivation is related to AWR (**peng2019awr**), with the main difference that AWAC uses an off-policy Q-function Q^π to estimate the advantage, which greatly improves efficiency and even final performance (see results in Section 7.6.1.1). The update also resembles ABM-MPO, but ABM-MPO *does* require modeling the behavior policy which, as discussed in Section 7.3.4, can lead to poor fine-tuning. In Section 7.6.1.1, AWAC outperforms ABM-MPO on a range of challenging tasks.

Policy evaluation. During policy evaluation, we estimate the action-value $Q^\pi(\mathbf{s},\mathbf{a})$ for the current policy π, as described in Section 9.3. We utilize a temporal difference learning scheme for policy evaluation (**haarnoja2018sac**; Fujimoto et al., 2018), minimizing the Bellman error as described in Equation 32. This enables us to learn very efficiently from off-policy data. This is particularly important in our problem setting to effectively use the offline dataset, and allows us to significantly outperform alternatives using Monte-Carlo evaluation or TD(λ) to estimate returns (**peng2019awr**).

Algorithm 5 Advantage Weighted Actor Critic (AWAC)

Dataset $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$
Initialize buffer $\beta = \mathcal{D}$
Initialize π_θ, Q_ϕ
for iteration $i = 1, 2, \ldots$ **do**
  Sample batch $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \sim \beta$
  $y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]$
  $\phi \leftarrow \arg\min_\phi \ \mathbb{E}_{\mathcal{D}}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2]$
  $\theta \leftarrow \arg\max_\theta \ _{\mathbf{s},\mathbf{a}\sim\beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right)\right]$
  if $i >$ num_offline_steps **then**
    $\tau_1, \ldots, \tau_K \sim p_{\pi_\theta}(\tau)$
    $\beta \leftarrow \beta \cup \{\tau_1, \ldots, \tau_K\}$
  end if
end for=0

Algorithm summary. The full AWAC algorithm for offline RL with online fine-tuning is summarized in Algorithm 5. In a practical implementation, we can parameterize the actor and the critic by neural networks and perform SGD updates from Eqn. 42 and Eqn. 33. Specific details are provided in Appendix **??**. AWAC ensures data efficiency with off-policy critic estimation via bootstrapping, and avoids offline bootstrap error with a constrained actor update. By avoiding explicit modeling of the behavior policy, AWAC avoids overly conservative updates.

While AWAC is related to several prior RL algorithms, we note that there are key differences that make it particularly amenable to the problem setting we are considering – offline RL with online fine-tuning – that other methods are unable to tackle. As we show in our experimental analysis with direct comparisons to prior work, every one of the design decisions being made in this work are important for algorithm performance. As compared to AWR (**peng2019awr**), AWAC uses TD bootstrapping for significantly more efficient and even asymptotically better performance. As compared to offline RL techniques like ABM (**siegel2020abm**), MPO (**we2018mpo**), BEAR (**kumar19bear**) or BCQ (**fujimoto19bcq**) this work is able to avoid the need for any behavior modeling, thereby enabling the *online* fine-tuning part of the problem much better. As shown in Fig 34, when these seemingly ablations are made to AWAC, the algorithm performs significantly worse.

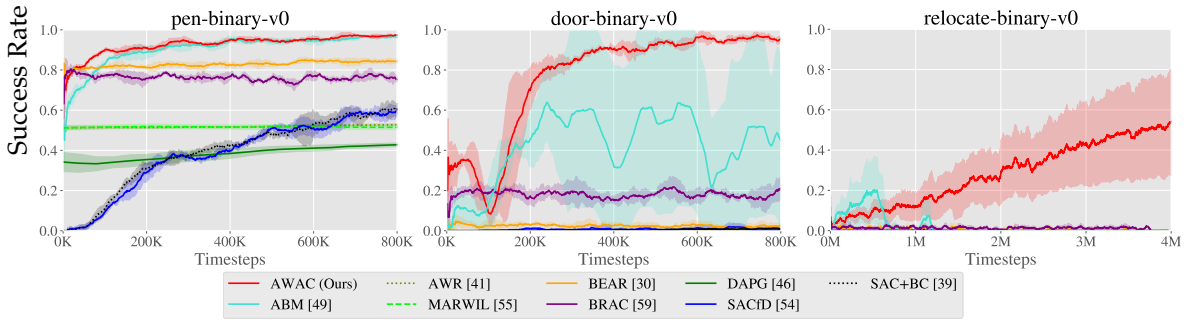


Figure 34: Comparative evaluation on the dexterous manipulation tasks. These tasks are difficult due to their high action dimensionality and reward sparsity. We see that AWAC is able to learn these tasks with little online data collection required (100K samples $\approx$ 16 minutes of equivalent real-world interaction time). Meanwhile, most prior methods are not able to solve the harder two tasks: door opening and object relocation.

7.5 Related Work

Off-policy RL algorithms are designed to reuse off-policy data during training, and have been studied extensively (**konda2000actorcritic**; **degris2012**; **haarnoja2018sac**; **bhatnagar2009**; **peters2008**; **zhang2019**; **pawel2009**; **balduzzi2015**; Mnih et al., 2016; Fujimoto et al., 2018). While standard off-policy methods are able to benefit from including data seen *during* a training run, as we show in Section 7.3.3 they struggle when training from previously collected offline data from other policies, due to error accumulation with distribution shift (**fujimoto19bcq**; **kumar19bear**). Offline RL methods aim to address this issue, often by constraining the actor updates to avoid excessive deviation from the data distribution (**lange2012**; **thomas2016**; **hallak2015offpolicy**; **hallak2016td**; **hallak2017onlineoffpolicy**; **agarwal2019optimism**; **kumar19bear**; **fujimoto19bcq**; **rasool2019p30**; **nachum2019dualdice**; **siegel2020abm**; **levine2020offlinetutorial**; **zhang2020gendice**). One class of these methods utilize importance sampling (**thomas2016**; **zhang2020gendice**; **nachum2019dualdice**; **degris2012**; **jiang2016doublyrobust**; **hallak2017onlineoffpolicy**). Another class of methods perform offline reinforcement learning via dynamic programming, with an explicit constraint to prevent deviation from the data distribution (**lange2012**; **kumar19bear**; **fujimoto19bcq**; **wu2019brac**; **jaques2019**). While these algorithms perform well in the purely offline settings, we show in Section 7.3.4 that such methods tend to be overly conservative, and therefore may not learn efficiently when fine-tuning with online data collection. In contrast, our algorithm AWAC is comparable to these algorithms for offline pre-training, but learns much more efficiently during subsequent fine-tuning.

Prior work has also considered the special case of learning from *demonstration* data. One class of algorithms initializes the policy via behavioral cloning from demonstrations,

and then fine-tunes with reinforcement learning (**peters2008baseball**; **ijspeert2002attractor**; **Theodorou2010**; **kim2013apid**; **rajeswaran2018dextrous**; **gupta2019relay**; **zhu2019hands**). Most such methods use on-policy fine-tuning, which is less sample-efficient than off-policy methods that perform value function estimation. Other prior works have incorporated demonstration data into the replay buffer using off-policy RL methods (**vecerik17ddpgfd**; **nair2017icra**). We show in Section 7.3.3 that these strategies can result in a large dip in performance during online fine-tuning, due to the inability to pre-train an effective value function from offline data. In contrast, our work shows that using supervised learning style policy updates can allow for better bootstrapping from demonstrations as compared to **vecerik17ddpgfd** and **nair2017icra**.

Our method builds on algorithms that implement a maximum likelihood objective for the actor, based on an expectation-maximization formulation of RL (**peters2007rwr**; **neumann2008fqiawr**; **Theodorou2010**; **peters2010reps**; **peng2019awr**; **we2018mpo**; **wang2018marwil**). Most closely related to our method in this respect are the algorithms proposed by **peng2019awr** (AWR) and **siegel2020abm** (ABM). Unlike AWR, which estimates the value function of the *behavior* policy, V^{π_β} via Monte-Carlo estimation or TD$-\lambda$, our algorithm estimates the Q-function of the *current* policy Q^π via bootstrapping, enabling much more efficient learning, as shown in our experiments. Unlike ABM, our method does not require learning a separate function approximator to model the behavior policy π_β, and instead directly samples the dataset. As we discussed in Section 7.3.4, modeling π_β can be a major challenge for online fine-tuning. While these distinctions may seem somewhat subtle, they are important and we show in our experiments that they result in a large difference in algorithm performance. Finally, our work goes beyond the analysis in prior work, by studying the issues associated with pre-training and fine-tuning in Section 7.3. Closely to our work, **wang2020crr** proposed critic regularized regression for offline RL, which uses off-policy Q-learning and an equivalent policy update. In contrast to this concurrent work, we specifically study the offline pretraining online fine-tuning problem, which this prior work does not address, analyze why other methods are ineffective in this setting, and show that our approach enables strong fine-tuning results on challenging dextrous manipulation tasks and real-world robotic systems.

The idea of bootstrapping learning from prior data for real-world robotic learning is not a new one; in fact, it has been extensively explored in the context of providing initial rollouts to bootstrap policy search **kober2008mp**; **peters2008baseball**; **kormushev2010**, initializing dynamic motion primitives **bentivagna2004**; **kormushev2010**; **mulling2013** in the context of on-policy reinforcement learning algorithms **rajeswaran2018dextrous**; **zhu2019hands**, inferring reward shaping **wu2020shaping** and even for inferring reward

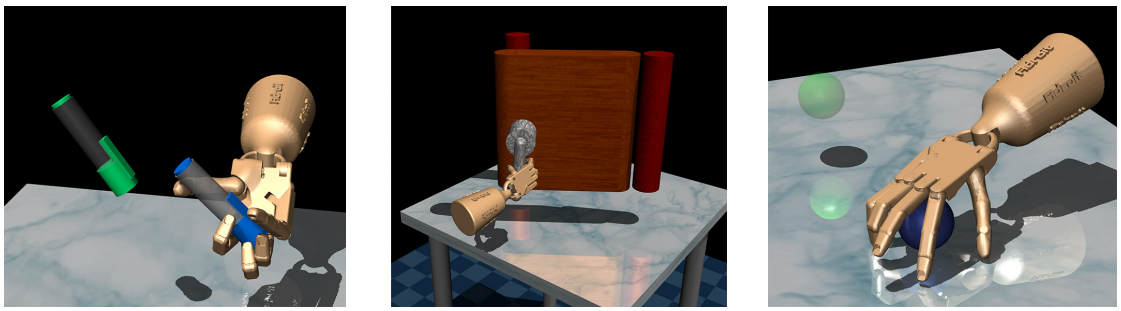


Figure 35: Illustration of dexterous manipulation tasks in simulation. These tasks exhibit sparse rewards, high-dimensional control, and complex contact physics.

functions **ziebart2008maxent**; **abbeel2004apprenticeship**. Our work shows how we can generalize the idea of bootstrapping robotic learning from prior data to include arbitrary sub-optimal data rather than just demonstration data and shows the ability to continue improving beyond this data as well.

7.6 EXPERIMENTAL EVALUATION

In our experimental evaluation we aim to answer the following question:

1. Does AWAC effectively combine prior data with online experience to learn complex robotic control tasks more efficiently than prior methods?
2. Is AWAC able to learn from sub-optimal or random data?
3. Does AWAC provide a practical way to bootstrap real-world robotic reinforcement learning?

In the following sections, we study these questions using several challenging and high-dimensional simulated robotic tasks, as well as three separate real-world robotic platforms. Videos of all experiments are available at *awacrl.github.io*

7.6.1 *Simulated Experiments*

We study the first two questions in challenging simulation environments.

7.6.1.1 *Comparative Evaluation When Bootstrapping From Prior Data*

We study tasks in simulation that have significant exploration challenges, where offline learning and online fine-tuning are likely to be effective. We begin our analysis with a set of challenging sparse reward dexterous manipulation tasks proposed by

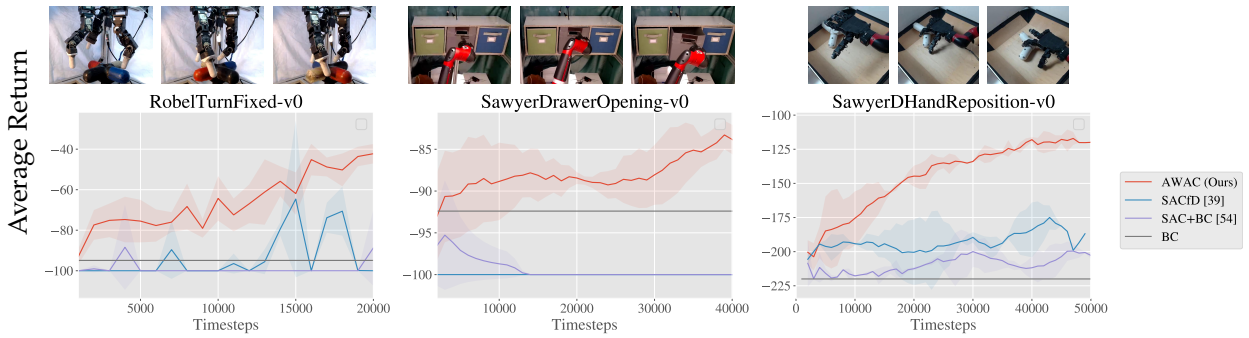


Figure 36: Algorithm comparison on three real-world robotic environments. Images of real world robotic tasks are pictured above. Left: a three fingered D'claw must rotate a valve 180°. Middle: a Sawyer robot must slide open a drawer using a hook attachment. Right: a dexterous hand attached to a Sawyer robot must re-position an object to to the center of the table. On each task, AWAC trained offline achieves reasonable performance (shown at timestep 0) and then steadily improves from online interaction. Other methods, which also all have access to prior data, fail to utilize the prior data effectively offline and therefore exhibit slow or no online improvement. Videos of all experiments are available at *awacrl.github.io*

rajeswaran2018dextrous in simulation. These tasks involve complex manipulation skills using a 28-DoF five-fingered hand in the MuJoCo simulator (Todorov et al., 2012) shown in Figure 34: in-hand rotation of a pen, opening a door by unlatching the handle, and picking up a sphere and relocating it to a target location. The reward functions in these environments are binary 0-1 rewards for task completion. [2] **rajeswaran2018dextrous** provide 25 human demonstrations for each task, which are not fully optimal but do solve the task. Since this dataset is small, we generated another 500 trajectories of interaction data by constructing a behavioral cloned policy, and then sampling from this policy.

First, we compare our method on these dexterous manipulation tasks against prior methods for off-policy learning, offline learning, and bootstrapping from demonstrations. Specific implementation details are discussed in Appendix **??**. The results are shown in Fig. 34. Our method is able to leverage the prior data to quickly attain good performance, and the efficient off-policy actor-critic component of our approach fine-tunes much more quickly than demonstration augmented policy gradient (DAPG), the method proposed by **rajeswaran2018dextrous**. For example, our method solves the pen task in 120K timesteps, the equivalent of just 20 minutes of online interaction. While the baseline comparisons and ablations make some amount of progress on the pen task, alternative

2 **rajeswaran2018dextrous** use a combination of task completion factors as the sparse reward. For instance, in the door task, the sparse reward as a function of the door position d was $r = 10\mathbb{1}_{d>1.35} + 8\mathbb{1}_{d>1.0} + 2\mathbb{1}_{d>1.2} - 0.1\|d - 1.57\|_2$. We only use the fully sparse success measure $r = \mathbb{1}_{d>1.4}$, which is substantially more difficult.

off-policy RL and offline RL algorithms are largely unable to solve the door and relocate task in the time-frame considered. We find that the design decisions to use off-policy critic estimation allow AWAC to outperform AWR (**peng2019awr**) while the implicit behavior modeling allows AWAC to significantly outperform ABM (**siegel2020abm**), although ABM does make some progress. **rajeswaran2018dextrous** show that DAPG can solve variants of these tasks with more well-shaped rewards, but requires considerably more samples.

Additionally, we evaluated all methods on the Gym MuJoCo locomotion benchmarks, similarly providing demonstrations as offline data. The results plots for these experiments are included in Appendix **??** in the supplementary materials. These tasks are substantially easier than the sparse reward manipulation tasks described above, and a number of prior methods also perform well. SAC+BC and BRAC perform on par with our method on the HalfCheetah task, and ABM performs on par with our method on the Ant task, while our method outperforms all others on the Walker2D task. However, our method matches or exceeds the best prior method in all cases, whereas no other single prior method attains good performance on all tasks.

7.6.1.2 *Fine-Tuning from Random Policy Data*

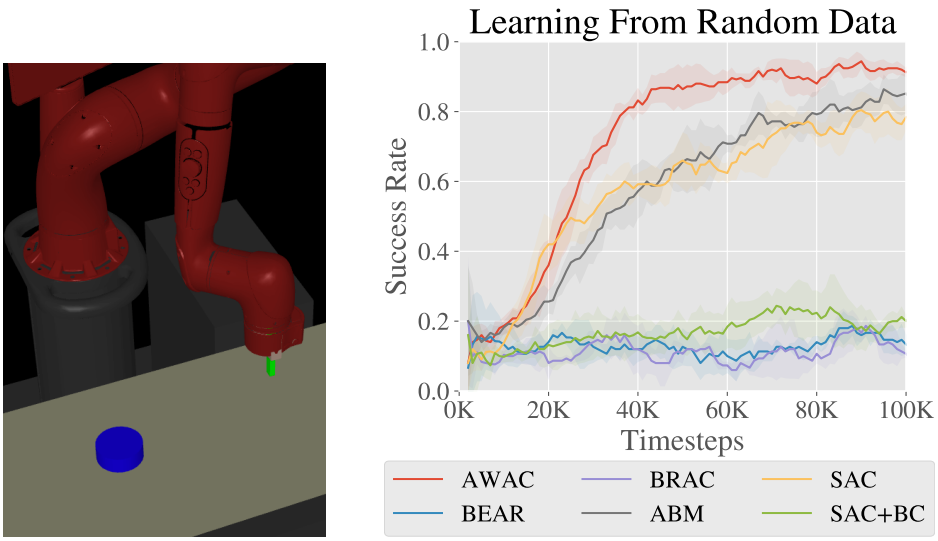


Figure 37: Comparison of fine-tuning from an initial dataset of suboptimal data on a Sawyer robot pushing task.

An advantage of using off-policy RL for reinforcement learning is that we can also incorporate suboptimal data, rather than demonstrations. In this experiment, we evaluate on a simulated tabletop pushing environment with a Sawyer robot pictured in Fig 34 and described further in Appendix **??**. To study the potential to learn from suboptimal data, we use an off-policy dataset of 500 trajectories generated by a random process. The task is to push an object to a target location in a 40cm x 20cm goal space. The results are shown in Figure 37. We see that while many methods begin at the same initial performance, AWAC learns the fastest online and is actually able to make use of the offline dataset effectively.

7.6.2 *Real-World Robot Learning with Prior Data*

We next evaluate AWAC and several baselines on a range of real-world robotic systems, shown in the top row of Fig 36. We study the following tasks: rotating a valve with a 3-fingered claw, repositioning an object with a 4-fingered hand, and opening a drawer with a Sawyer robotic arm. The dexterous manipulation tasks involve fine finger coordination to properly reorient and reposition objects, as well as high dimensional state and action spaces. The Sawyer drawer opening task requires accurate arm movements to properly hook the end-effector into the handle of the drawer. To ensure continuous operation, all environments are fitted with an automated reset mechanism that executes before each trajectory is collected, allowing us to run real-world experiments without human supervision. Since real-world experiments are significantly more time-consuming, we could not compare to the full range of prior methods in the real world, but we include comparisons with the following methods: direct behavioral cloning (BC) of the provided data (which is reasonable in these settings, since the prior data includes demonstrations), off-policy RL with soft actor-critic (SAC) **haarnoja2018sac**, where the prior data is included in the replay buffer and used to pretrain the policy (which refer to as SACfD), and a modified version of SAC that includes an added behavioral cloning loss (SAC+BC), which is analogous to **nair2018demonstrations** or an off-policy version of **rajeswaran2018dextrous**. Further implementation details of these algorithms are provided in Appendix **??** in the supplementary materials.

Next, we describe the experimental setup for hardware experiments. Precise details of the hardware setup can be found in Appendix **??** in the supplementary materials.

Dexterous Manipulation with a 3-Fingered Claw. This task requires controlling a 3-fingered, 9 DoF robotic hand, introduced by **ahn2019robel**, to rotate a 4-pronged valve object by 180 degrees. To properly perform this task, multiple fingers need to coordinate

to stably and efficiently rotate the valve into the desired orientation. The state space of the system consists of the joint angles of all the 9 joints in the claw, and the action space consists of the joint positions of the fingers, which are followed by the robot using a low-level PID controller. The reward for this task is sparse: -1 if the valve is rotated within 0.25 radians of the target, and 0 otherwise. Note that this reward function is significantly more difficult than the dense, well-shaped reward function typically used in prior work **ahn2019robel**. The prior data consists of 10 trajectories collected using kinesthetic teaching, combined this with 200 trajectories obtained through executing a policy trained via imitation learning in the environment.

Drawer Opening with a Sawyer Arm. This task requires controlling a Sawyer arm to slide open a drawer. The robot uses 3-dimensional end-effector control, and is equipped with a hook attachment to make the drawer opening possible. The state space is 4-dimensional, consisting of the position of the robot end-effector and the linear position of the drawer, measured using an encoder. The reward is sparse: -1 if the drawer is open beyond a threshold and 0 otherwise. For this task, the prior data consists of 10 demonstration trajectories collected using via teleoperation with a 3D mouse, as well as 500 trajectories obtained through executing a policy trained via imitation learning in the environment. This task is challenging because it requires very precise insertion of the hook attachment into the opening, as pictured in Fig 36, before the robot can open the drawer. Due to the sparse reward, making learning progress on this task requires utilizing prior data to construct an initial policy that at least sometimes succeeds.

Dexterous Manipulation with a Robotic Hand. This task requires controlling a 4-fingered robotic hand mounted on a Sawyer robotic arm to reposition an object **gupta2021dexterous**. The task requires careful coordination between the hand and the arm to manipulate the object accurately. The reward for this task is a combination of the negative distance between the hand and the object and the negative distance between the object and the target. The actions are 19-dimensional, consisting of 16-dimensional finger control and 3-dimensional end effector control of the arm. For this task, the prior data of 19 trajectories were collected using kinesthetic teaching and combined with 50 trajectories obtained by executing a policy trained with imitation learning on this data.

The results on these tasks are shown in Figure 36. We first see that AWAC attains performance that is comparable to the best prior method from offline training alone, as indicated by the value at time step 0 (which corresponds to the beginning of online fine-tuning). This means that, during online interaction, AWAC collects data that is of higher quality, and improves more quickly. The prior methods struggle to improve from online training on these tasks, likely because the sparse reward function and challenging dy-

namics make progress very difficult from a bad initialization. These results suggest that AWAC is effectively able to leverage prior data to bootstrap online reinforcement learning in the real world, even on tasks with difficult and uninformative reward functions.

7.7 DISCUSSION AND FUTURE WORK

We have discussed the challenges existing RL methods face when fine-tuning from prior datasets, and proposed an algorithm, AWAC, for this setting. The key insight in AWAC is that an implicitly constrained actor-critic algorithm is able to both train offline and continue to improve with more experience. We provide detailed empirical analysis of the design decisions behind AWAC, showing the importance of off-policy learning, bootstrapping and the particular form of implicit constraint enforcement. To validate these ideas, we evaluate on a variety of simulated and real world robotic problems.

While AWAC is able to effectively leverage prior data for significantly accelerating learning, it does run into some limitations. Firstly, it can be challenging to choose the appropriate threshold for constrained optimization. Resolving this would involve exploring adaptive threshold tuning schemes. Secondly, while AWAC is able to avoid overconservative behavior empirically, in future work, we hope to analyze theoretical factors that go into building a good finetuning algorithm. And lastly, in the future we plan on applying AWAC to more broadly incorporate data across different robots, labs and tasks rather than just on isolated setups. By doing so, we hope to enable an even wider array of robotic applications.

8

OFFLINE REINFORCEMENT LEARNING WITH IMPLICIT Q-LEARNING

Offline reinforcement learning (RL) addresses the problem of learning effective policies entirely from previously collected data, without online interaction (**fujimoto2019off**; **lange2012batch**). This is very appealing in a range of real-world domains, from robotics to logistics and operations research, where real-world exploration with untrained policies is costly or dangerous, but prior data is available. However, this also carries with it major challenges: improving the policy beyond the level of the behavior policy that collected the data requires estimating values for actions other than those that were seen in the dataset, and this, in turn, requires trading off policy improvement against distributional shift, since the values of actions that are too different from those in the data are unlikely to be estimated accurately. Prior methods generally address this by either constraining the policy to limit how far it deviates from the behavior policy (**fujimoto2019off**; **wu2019behavior**; **fujimoto2021minimalist**; **kumar2019stabilizing**; **nair2020awac**; **wang2020critic**), or by regularizing the learned value functions to assign low values to out-of-distribution actions (**kumar2020conservative**; **kostrikov2021offline**). Nevertheless, this imposes a trade-off between how much the policy improves and how vulnerable it is to misestimation due to distributional shift. Can we devise an offline RL method that avoids this issue by never needing to directly query or estimate values for actions that were not seen in the data?

In this work, we start from an observation that *in-distribution* constraints widely used in prior work might not be sufficient to avoid value function extrapolation, and we ask whether it is possible to learn an optimal policy with *in-sample learning*, without *ever* querying the values of any unseen actions. The key idea in our method is to approximate an upper expectile of the distribution over values with respect to the distribution of dataset actions for each state. We alternate between fitting this value function with expectile regression, and then using it to compute Bellman backups for training the Q-

function. We show that we can do this simply by modifying the loss function in a SARSA-style TD backup, without ever using out-of-sample actions in the target value. Once this Q-function has converged, we extract the corresponding policy using advantage-weighted behavioral cloning. This approach does not require explicit constraints or explicit regularization of out-of-distribution actions during value function training, though our policy extraction step does implicitly enforce a constraint, as discussed in prior work on advantage-weighted regression (**peters2007reinforcement**; **peng2019advantage**; **nair2020awac**; **wang2020critic**).

Our main contribution is implicit Q-learning (IQL), a new offline RL algorithm that avoids ever querying values of unseen actions while still being able to perform multi-step dynamic programming updates. Our method is easy to implement by making a small change to the loss function in a simple SARSA-like TD update and is computationally very efficient. Furthermore, our approach demonstrates the state-of-the-art performance on D4RL, a popular benchmark for offline reinforcement learning. In particular, our approach significantly improves over the prior state-of-the-art on challenging Ant Maze tasks that require to "stitch" several sub-optimal trajectories. Finally, we demonstrate that our approach is suitable for finetuning; after initialization from offline RL, IQL is capable of improving policy performance utilizing additional interactions.

8.1 RELATED WORK

A significant portion of recently proposed offline RL methods are based on either constrained or regularized approximate dynamic programming (e.g., Q-learning or actor-critic methods), with the constraint or regularizer serving to limit deviation from the behavior policy. We will refer to these methods as "multi-step dynamic programming" algorithms, since they perform true dynamic programming for multiple iterations, and therefore can in principle recover the optimal policy if provided with high-coverage data. The constraints can be implemented via an explicit density model (**wu2019behavior**; **fujimoto2019off**; **kumar2019stabilizing**; **ghasemipour2021emaq**), implicit divergence constraints (**nair2020awac**; **wang2020critic**; **peters2007reinforcement**; **peng2019advantage**; **siegel2020keep**), or by adding a supervised learning term to the policy improvement objective (**fujimoto2021minimalist**). Several works have also proposed to directly regularize the Q-function to produce low values for out-of-distribution actions (**kostrikov2021offline**; **kumar2020conservative**; **fakoor2021continuous**). Our method is also a multi-step dynamic programming algorithm. However, in contrast to prior works, our method completely avoids directly querying the learned Q-function with unseen actions during training, removing the need for any constraint during this stage, though the subsequent pol-

icy extraction, which is based on advantage-weighted regression (**peng2019advantage**; **nair2020awac**), does apply an implicit constraint. However, this policy does not actually influence value function training.

In contrast to multi-step dynamic programming methods, several recent works have proposed methods that rely either on a single step of policy iteration, fitting the value function or Q-function of the behavior policy and then extracting the corresponding greedy policy (**peng2019advantage**; **brandfonbrener2021offline**; **gulcehre2021regularized**), or else avoid value functions completely and utilize behavioral cloning-style objectives (**chen2021decisi**
We collectively refer to these as "single-step" approaches. These methods avoid needing to query unseen actions as well, since they either use no value function at all, or learn the value function of the behavior policy. Although these methods are simple to implement and effective on the MuJoCo locomotion tasks in D4RL, we show that such single-step methods perform very poorly on more complex datasets in D4RL, which require combining parts of suboptimal trajectories ("stitching"). Prior multi-step dynamic programming methods perform much better in such settings, as does our method. We discuss this distinction in more detail in Section 8.4.1. Our method also shares the simplicity and computational efficiency of single-step approaches, providing an appealing combination of the strengths of both types of methods.

Our method is based on estimating the characteristics of a random variable. Several recent works involve approximating statistical quantities of the value function distribution. In particular, quantile regression (**koenker2001quantile**) has been previously used in reinforcement learning to estimate the quantile function of a state-action value function (**dabney2018distributional**; **dabney2018implicit**; **kuznetsov2020controlling**). Although our method is related, in that we perform expectile regression, our aim is not to estimate the distribution of values that results from stochastic transitions, but rather estimate expectiles of the state value function with respect to random actions. This is a very different statistic: our aim is not to determine how the Q-value can vary with different future outcomes, but how the Q-value can vary with different actions *while averaging together future outcomes due to stochastic dynamics*. While prior work on distributional RL can also be used for offline RL, it would suffer from the same action extrapolation issues as other methods, and would require similar constraints or regularization, while our method does not.

8.2 PRELIMINARIES

The RL problem is formulated in the context of a Markov decision process (MDP) $(\mathbf{S}, , p_0(s), p(s'|s,a), r(s,a), \gamma)$, where $\mathbf{S}$ is a state space, is an action space, $p_0(s)$ is a distribution of initial states, $p(s'|s,a)$ is the environment dynamics, $r(s,a)$ is a reward function, and γ is a discount factor. The agent interacts with the MDP according to a policy $\pi(a|s)$. The goal is to obtain a policy that maximizes the cumulative discounted returns:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 \sim p_0(\cdot), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)\right].$$

Off-policy RL methods based on approximate dynamic programming typically utilize a state-action value function (Q-function), referred to as $Q(s,a)$, which corresponds to the discounted returns obtained by starting from the state s and action a, and then following the policy π.

OFFLINE REINFORCEMENT LEARNING. In contrast to online (on-policy or off-policy) RL methods, offline RL uses previously collected data without any additional data collection. Like many recent offline RL methods, our work builds on approximate dynamic programming methods that minimize temporal difference error, according to the following loss:

$$L_{TD}(\theta) = \mathbb{E}_{(s,a,s')\sim}[(r(s,a) + \gamma \max_{a'} Q_{\hat{\theta}}(s',a') - Q_{\theta}(s,a))^2], \tag{43}$$

where is the dataset, $Q_{\theta}(s,a)$ is a parameterized Q-function, $Q_{\hat{\theta}}(s,a)$ is a target network (e.g., with soft parameters updates defined via Polyak averaging), and the policy is defined as $\pi(s) = \arg\max_a Q_{\theta}(s,a)$. Most recent offline RL methods modify either the value function loss (above) to regularize the value function in a way that keeps the resulting policy close to the data, or constrain the $\arg\max$ policy directly. This is important because out-of-distribution actions a' can produce erroneous values for $Q_{\hat{\theta}}(s',a')$ in the above objective, often leading to *overestimation* as the policy is defined to maximize the (estimated) Q-value.

8.3 IMPLICIT Q-LEARNING

In this work, we aim to entirely avoid querying *out-of-sample* (unseen) actions in our TD loss. Although the goal of this work is to approximate the optimal Q-function, we start by considering fitted Q evaluation with a SARSA-style objective which has been considered in prior work on Offline Reinforcement Learning (**brandfonbrener2021offline**;

gulcehre2021regularized). This objective aims to learn the value of the dataset policy π_β (also called the behavior policy):

$$L(\theta) = \mathbb{E}_{(s,a,s',a')\sim}[(r(s,a) + \gamma Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2]. \tag{44}$$

This objective never queries values for out-of-sample actions, in contrast to Equation (43). One specific property of this objective that is important for this work is that it uses mean squared error (MSE) that fits $Q_\theta(s,a)$ to predict the mean statistics of the TD targets. Thus, if we assume unlimited capacity and no sampling error, the optimal parameters should satisfy

$$Q_{\theta^*}(s,a) \approx r(s,a) + \gamma \mathbb{E}_{\substack{s'\sim p(\cdot|s,a) \\ a'\sim\pi_\beta(\cdot|s)}}[Q_{\hat{\theta}}(s',a')]. \tag{45}$$

Prior work (**brandfonbrener2021offline**; **gulcehre2021regularized**; **peng2019advantage**) has proposed directly using this objective to learn Q^{π_β}, and then train the policy π_ψ to maximize Q^{π_β}. This avoids any issues with out-of-distribution actions, since the TD loss only uses dataset actions. However, while this procedure works well empirically on simple MuJoCo locomotion tasks in D4RL, we will show that it performs very poorly on more complex tasks that benefit from multi-step dynamic programming. In our method, which we derive next, we retain the benefits of using this SARSA-like objective, but modify it so that it allows us to perform multi-step dynamic programming and learn a near-optimal Q-function.

Our method will perform a Q-function update similar to Equation (44), but we will aim to estimate the maximum Q-value over actions that are in the support of the data distribution. Crucially, we will show that it is possible to do this *without ever querying the learned Q-function on out-of-sample actions* by utilizing expectile regression. Formally, the value function we aim to learn is given by:

$$L(\theta) = \mathbb{E}_{(s,a,s')\sim}[(r(s,a) + \gamma \max_{\substack{a' \in \\ \text{s.t. } \pi_\beta(a'|s')>0}} Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2]. \tag{46}$$

Our algorithm, implicit Q-Learning (IQL), aims to estimate this objective while evaluating the Q-function only on the state-action pairs in the dataset. To this end, we propose to fit $Q_\theta(s,a)$ to estimate state-conditional expectiles of the target values, and show that specific expectiles approximate the maximization defined above. In Section 8.3.4 we show that this approach performs multi-step dynamic programming in theory, and in Section 8.4.1 we show that it does so in practice.

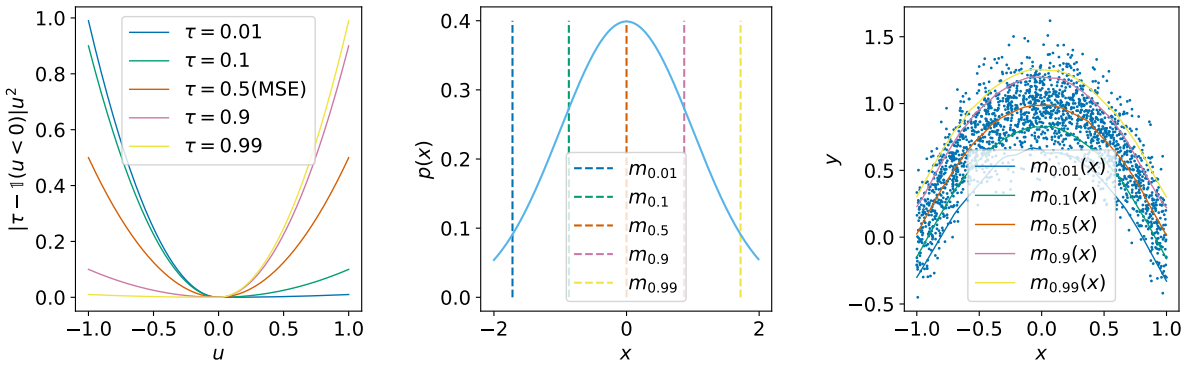


Figure 38: **Left:** The asymmetric squared loss used for expectile regression. $\tau = 0.5$ corresponds to the standard mean squared error loss, while $\tau = 0.9$ gives more weight to positives differences. **Center:** Expectiles of a normal distribution. **Right:** an example of estimating state conditional expectiles of a two-dimensional random variable. Each x corresponds to a distribution over y. We can approximate a maximum of this random variable with expectile regression: $\tau = 0.5$ correspond to the conditional mean statistics of the distribution, while $\tau \approx 1$ approximates the maximum operator over in-support values of y.

8.3.1 *Expectile Regression*

Practical methods for estimating various statistics of a random variable have been thoroughly studies in applied statistics and econometrics. The $\tau \in (0, 1)$ expectile of some random variable X is defined as a solution to the asymmetric least squares problem:

$$\arg\min_{m_\tau} \mathbb{E}_{x\sim X}[L_2^\tau(x - m_\tau)], \text{ where } L_2^\tau(u) = |\tau - 1(u < 0)|u^2.$$

That is, for $\tau > 0.5$, this asymmetric loss function downweights the contributions of x values smaller than m_τ while giving more weights to larger values (see Figure 38, left). Expectile regression is closely related to quantile regression (**koenker2001quantile**), which is a popular technique for estimating quantiles of a distribution widely used in reinforcement learning (**dabney2018distributional**; **dabney2018implicit**) [1]. The quantile regression loss is defined as an asymmetric ℓ_1 loss.

1 Our method could also be derived with quantiles, but since we are not interested in learning all of the expectiles/quantiles, unlike prior work (**dabney2018distributional**; **dabney2018implicit**), it is more convenient to estimate a single expectile because this involves a simple modification to the MSE loss that is already used in standard RL methods. We found it to work somewhat better than quantile regression with its corresponding ℓ_1 loss.

We can also use this formulation to predict expectiles of a conditional distribution:

$$\arg\min_{m_\tau(x)} \mathbb{E}_{(x,y)\sim}[L_2^\tau(y - m_\tau(x))].$$

Figure 38 (right) illustrates conditional expectile regression on a simple two-dimensional distribution. Note that we can optimize this objective with stochastic gradient descent. It provides unbiased gradients and is easy to implement with standard machine learning libraries.

8.3.2 *Learning the Value Function with Expectile Regression*

Expectile regression provides us with a powerful framework to estimate statistics of a random variable beyond mean regression. We can use expectile regression to modify the policy evaluation objective in Equation (44) to predict an upper expectile of the TD targets that approximates the maximum of $r(s, a) + \gamma Q_{\hat{\theta}}(s', a')$ over actions a' constrained to the dataset actions, as in Equation (46). This leads to the following expectile regression objective:

$$L(\theta) = \mathbb{E}_{(s,a,s',a')\sim}[L_2^\tau(r(s, a) + \gamma Q_{\hat{\theta}}(s', a') - Q_\theta(s, a))].$$

However, this formulation has a significant drawback. Instead of estimating expectiles just with respect to the actions in the support of the data, it also incorporates stochasticity that comes from the environment dynamics $s' \sim p(\cdot|s, a)$. Therefore, a large target value might not necessarily reflect the existence of a single action that achieves that value, but rather a "lucky" sample that happened to have transitioned into a good state. We resolve this by introducing a separate value function that approximates an expectile only with respect to the action distribution, leading to the following loss:

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim}[L_2^\tau(Q_{\hat{\theta}}(s, a) - V_\psi(s))]. \tag{47}$$

We can then use this estimate to update the Q-functions with the MSE loss, which averages over the stochasticity from the transitions and avoids the "lucky" sample issue mentioned above:

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim}[(r(s, a) + \gamma V_\psi(s') - Q_\theta(s, a))^2]. \tag{48}$$

Note that these losses do not use any explicit policy, and only utilize actions from the dataset for both objectives, similarly to SARSA-style policy evaluation. In Section 8.3.4, we will show that this procedure recovers the optimal Q-function under some assumptions. Also, even though only one action is available for every state in the dataset for continuous action spaces, due to neural network generalization, the expectile regression does not result in SARSA-style policy evaluation as shown in Section 8.4.2.

8.3.3 *Policy Extraction and Algorithm Summary*

While our modified TD learning procedure learns an approximation to the optimal Q-function, it does not explicitly represent the corresponding policy, and therefore requires a separate policy extraction step. While one can consider any technique for policy extraction that constrains the learned policy to stay close to the dataset actions, we aim for a simple method for policy extraction. As before, we aim to avoid using out-of-samples actions. Therefore, we extract the policy with advantage-weighted regression (**peters2007reinforcement**; **peng2019advantage**) previously successfully used for policy extraction in Offline RL (**wang2018exponentially**; **nair2020awac**; **brandfonbrener20210ffline**):

Algorithm 6 Implicit Q-learning

Initialize parameters ψ, θ, $\hat{\theta}$, ϕ.
TD learning (IQL):
for each gradient step **do**
  $\psi \leftarrow \psi - \lambda_V \nabla_\psi L_V(\psi)$
  $\theta \leftarrow \theta - \lambda_Q \nabla_\theta L_Q(\theta)$
  $\hat{\theta} \leftarrow (1-\alpha)\hat{\theta} + \alpha\theta$
Policy extraction (AWR):
for each gradient step **do**
  $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi L_\pi(\phi)$

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim}[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s)))\log \pi_\phi(a|s)], \tag{49}$$

where $\beta \in [0, \infty)$ is an inverse temperature. Note that this objective does not clone all actions from the dataset but, as shown in prior work, this objective learns a policy that maximizes the Q-values subject to a distribution constraint (**peters2007reinforcement**; **peng2019advantage**; **nair2020awac**). This step can be seen as selecting and cloning the most optimal actions in the dataset.

Our final algorithm consists of two stages. First, we fit the value function and Q, performing a number of gradient updates alternating between Eqn. (47) and (48). Second, we perform stochastic gradient descent on Equation (49). For both steps, we use a version of clipped double Q-learning (**fujimoto2018addressing**), taking a minimum of two Q-functions for V-function and policy updates. We summarize our final method in Al-

gorithm 6. Note that the policy does not influence the value function in any way, and therefore extraction could be performed either concurrently or after TD learning. Concurrent learning provides a way to use IQL with online finetuning, as we discuss in Section 8.4.3.

8.3.4 *Analysis*

In this section, we will show that IQL can recover the optimal value function under the dataset support constraints. First, we prove a simple lemma that we will then use to show how our approach can enable learning the optimal value function.

Lemma 1. *Let X be a real-valued random variable with a bounded support and supremum of the support is x^*. Then,*

$$\lim_{\tau \to 1} m_\tau = x^*$$

One can show that expectiles of a random variable have the same supremum x^*. Moreover, for all τ_1 and τ_2 such that $\tau_1 < \tau_2$, we get $m_{\tau_1} \leqslant m_{\tau_2}$. Therefore, the limit follows from the properties of bounded monotonically non-decreasing functions.

In the following theorems, we show that under certain assumptions, our method indeed approximates the optimal state-action value Q^* and performs multi-step dynamical programming. We first prove a technical lemma relating different expectiles of the Q-function, and then derive our main result regarding the optimality of our method.

For the sake of simplicity, we introduce the following notation for our analysis. Let $\mathbb{E}^\tau_{x \sim X}[x]$ be a τ^{th} expectile of X (e.g., $\mathbb{E}^{0.5}$ corresponds to the standard expectation). Then, we define $V_\tau(s)$ and $Q_\tau(s, a)$, which correspond to optimal solutions of Eqn. 47 and 48 correspondingly, recursively as:

$$V_\tau(s) = \mathbb{E}^\tau_{a \sim \pi_\beta(\cdot|s)}[Q_\tau(s, a)],$$
$$Q_\tau(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)}[V_\tau(s')].$$

Lemma 2. *For all s, τ_1 and τ_2 such that $\tau_1 < \tau_2$ we get $V_{\tau_1}(s) \leqslant V_{\tau_2}(s)$.*

Proof. The proof follows the policy improvement proof (**sutton2018reinforcement**). See **??**. □

Corollary 1. *For any τ and s we have $V_\tau(s) \leqslant \max_{\substack{a \in \\ s.t.\ \pi_\beta(a|s)>0}} Q^*(s,a)$ where $V_\tau(s)$ is defined as above and $Q^*(s,a)$ is an optimal state-action value function constrained to the dataset and defined as*

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{\substack{a' \in \\ s.t.\ \pi_\beta(a'|s')>0}} Q^*(s',a') \right].$$

Proof. The proof follows from the observation that convex combination is smaller than maximum. □

Theorem 1.

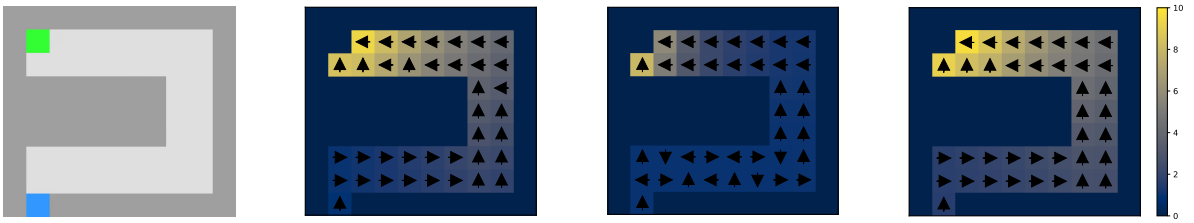
$$\lim_{\tau \to 1} V_\tau(s) = \max_{\substack{a \in \\ s.t.\ \pi_\beta(a|s)>0}} Q^*(s,a).$$

Proof. Follows from combining Lemma 1 and Corollary 1. □

Therefore, for a larger value of $\tau < 1$, we get a better approximation of the maximum. On the other hand, it also becomes a more challenging optimization problem. Thus, we treat τ as a hyperparameter. Due to the property discussed in Theorem 1 we dub our method implicit Q-learning (IQL). We also emphasize that our value learning method defines the entire spectrum of methods between SARSA ($\tau = 0.5$) and Q-Learning ($\tau \to 1$). Note that in contrast to other multi-step methods, IQL absorbs the policy improvement step into value learning. Therefore, fitting Q-function corresponds to the policy evaluation step, while fitting the value function with IQL corresponds to *implicit* policy improvement.

8.4 Experimental evaluation

Our experiments aim to evaluate our method comparatively, in contrast to prior offline RL methods, and in particular to understand how our approach compares both to single-step methods and multi-step dynamic programming approaches. We will first demonstrate the benefits of multi-step dynamic programming methods, such as ours, in contrast to single-step methods, showing that on some problems this difference can be extremely large. We will then compare IQL with state-of-the-art single-step and multi-step algorithms on the D4RL (**fu2020d4rl**) benchmark tasks, studying the degree to which we can learn effective policies using only the actions in the dataset. We examine domains that contain near-optimal trajectories, where single-step methods perform well, as well as



(a) toy maze MDP (b) true optimal $V^\star$ (c) One-step Policy Eval. (d) IQL

Figure 39: Evaluation of our algorithm on a toy umaze environment (a). When the static dataset is heavily corrupted by suboptimal actions, one-step policy evaluation results in a value function that degrades to zero far from the rewarding states too quickly (c). Our algorithm aims to learn a near-optimal value function, combining the best properties of SARSA-style evaluation with the ability to perform multi-step dynamic programming, leading to value functions that are much closer to optimality (shown in (b)) and producing a much better policy (d).

domains with no optimal trajectories at all, which require multi-step dynamic programming. Finally, we will study how IQL compares to prior methods when finetuning with online RL starting from an offline RL initialization.

8.4.1 *The Difference Between One-Step Policy Improvement and IQL*

We first use a simple maze environment to illustrate the importance of multi-step dynamic programming for offline RL. The maze has a u-shape, a single start state, and a single goal state (see Figure 39a). The agent receives a reward of 10 for entering the goal state and zero reward for all other transitions. With a probability of 0.25, the agent transitions to a random state, and otherwise to the commanded state. The dataset consists of 1 optimal trajectory and 99 trajectories with uniform random actions. Due to a short horizon of the problem, we use $\gamma = 0.9$.

Figure 39 (c, d) illustrates the difference between single-step methods which fit $Q^\pi(s, a)$ via SARSA-style objective, in this case represented by Onepstep RL (**brandfonbrener2021offline**; **wang2018exponentially**; **gulcehre2021regularized**) and IQL with $\tau = 0.95$. Note that these methods represent a special case of our method with $\tau = 0.5$. Although states closer to the high reward state will still have higher values, these values decay much faster as we move further away than they would for the optimal value function, and the resulting policy is highly suboptimal. Since IQL (d) performs iterative dynamic programming, it correctly propagates the signal, and the values are no longer dominated by noise. The resulting value function closely matches the true optimal value function (b).

8.4.2 *Comparisons on Offline RL Benchmarks*

Next, we evaluate our approach on the D4RL benchmark in comparison to prior methods (see Table 1). The MuJoCo tasks in D4RL consist of the Gym locomotion tasks, the Ant Maze tasks, and the Adroit and Kitchen robotic manipulation environments. Some prior works, particularly those proposing one-step methods, focus entirely on the Gym locomotion tasks. However, these tasks include a significant fraction of near-optimal trajectories in the dataset. In contrast, the Ant Maze tasks, especially the medium and large ones, contain very few or no near-optimal trajectories, making them very challenging for one-step methods. These domains require "stitching" parts of suboptimal trajectories that travel between different states to find a path from the start to the goal of the maze (**fu2020d4rl**). As we will show, multi-step dynamic programming is essential in these domains. The Adroit and Kitchen tasks are comparatively less discriminating, and we found that most RL methods perform similarly to imitation learning in these domains (**florence2021implicit**). We therefore focus our analysis on the Gym locomotion and Ant Maze domains, but include full Adroit and Kitchen results in **??** for completeness.

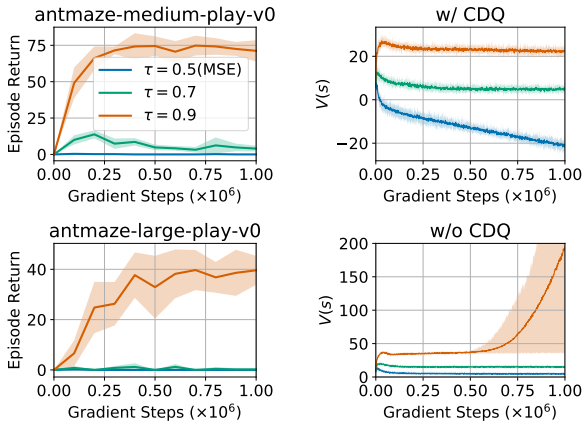


Figure 40: **Left**: Estimating a larger expectile τ is crucial for antmaze tasks that require dynamical programming ('stitching'). **Right:** Clipped double Q-Learning (CDQ) is crucial for learning values for $\tau = 0.9$.

COMPARISONS AND BASELINES. We compare to methods that are representative of both multi-step dynamic programming and one-step approaches. In the former category, we compare to CQL (**kumar2020conservative**), TD3+BC (**fujimoto2021minimalist**), and AWAC (**nair2020awac**). In the latter category, we compare to Onestep RL (**brandfonbrener2021offline**) and Decision Transformers (**chen2021decision**). We obtained the Decision Transformers results on Ant Maze subsets of D4RL tasks using the author-provided implementation[2] and following authors instructions communicated over email. We obtained results for TD3+BC and Onestep RL (Exp. Weight) directly from the authors. Note that

2 https://github.com/kzl/decision-transformer

chen2021decision and **brandfonbrener20210ffline** incorrectly report results for some prior methods, such as CQL, using the "-v0" environments. These generally produce lower scores than the "-v2" environments that these papers use for their own methods. We use the "-v2" environments for all methods to ensure a fair comparison, resulting in higher values for CQL. Because of this fix, our reported CQL scores are higher than all other prior methods. We obtained results for "-v2" datasets using an author-suggested implementation.[3] On the Gym locomotion tasks (halfcheetah, hopper, walker2d), we find that IQL performs comparably to the best performing prior method, CQL. On the more challenging Ant Maze task, IQL outperforms CQL, and outperforms the one-step methods by a very large margin.

RUNTIME. Our approach is also computationally faster than the baselines (see Table 1). For the baselines, we measure runtime for our reimplementations of the methods in JAX (**jax2018github**) built on top of JAXRL (**jaxrl**), which are typically faster than the original implementations. For example, the original implementation of CQL takes more than 4 hours to perform 1M updates, while ours takes only 80 minutes. Even so, IQL still requires about 4x less time than our reimplementation of CQL on average, and is comparable to the fastest prior one-step methods. We did not reimplement Decision Transformers due to their complexity and report runtime of the original implementation.

EFFECT OF τ HYPERPARAMETER. We also demonstrate that it is crucial to compute a larger expectile on tasks that require "stitching" (see Figure 40). We provide complete results in **??**. With larger values of τ, our method approximates Q-learning better, leading to better performance on the Ant Maze tasks. Moreover, due to neural network generalization, values learned with expectile regression increase with a larger τ and do not degrade to behavior policy values ($\tau = 0.5$). Finally, clipped double Q-Learning is crucial for estimating values for a larger $\tau = 0.9$.

8.4.3 *Online Fine-tuning after Offline RL*

The policies obtained by offline RL can often be improved with a small amount of online interaction. IQL is well-suited for online fine-tuning for two reasons. First, IQL has strong offline per-

Dataset	AWAC	CQL	IQL (Ours)
antmaze-umaze-v0	56.7 → 59.0	70.1 → **99.4**	**88.0** → **96.3**
antmaze-umaze-diverse-v0	49.3 → 49.0	31.1 → **99.4**	**67.0** → 49.0
antmaze-medium-play-v0	0.0 → 0.0	23.0 → 0.0	**69.0** → **89.2**
antmaze-medium-diverse-v0	0.7 → 0.3	23.0 → 32.3	**71.8** → **91.4**
antmaze-large-play-v0	0.0 → 0.0	1.0 → 0.0	**36.8** → **51.8**
antmaze-large-diverse-v0	1.0 → 0.0	1.0 → 0.0	**42.2** → **59.8**
antmaze-v0 total	107.7 → 108.3	151.5 → 231.1	**374.8** → **437.5**

3 https://github.com/young-geng/CQL

formance, as shown in the previous section, which provides a good initialization. Second, IQL implements a weighted behavioral cloning policy extraction step, which has previously been shown to allow for better online policy improvement compared to other types of offline constraints (**nair2020awac**). To evaluate the finetuning capability of various RL algorithms, we first run offline RL on each dataset, then run 1M steps of online RL, and then report the final performance. We compare to AWAC (**nair2020awac**), which has been proposed specifically for online finetuning, and CQL (**kumar2020conservative**), which showed the best performance among prior methods in our experiments in the previous section. Exact experimental details are provided in Appendix **??**. We use the challenging Ant Maze D4RL domains (**fu2020d4rl**), as well as the high-dimensional dexterous manipulation environments from **rajeswaran2018dextrous**, which **nair2020awac** propose to use to study online adaptation with AWAC. Results are shown in Table 2. On the Ant Maze domains, IQL significantly outperforms both prior methods after online finetuning. CQL attains the second best score, while AWAC performs comparatively worse due to much weaker offline initialization. On the dexterous hand tasks, IQL performs significantly better than AWAC on relocate-binary-v0, comparably on door-binary-v0, and slightly worse on pen-binary-v0, with the best overall score.

8.5 conclusion

We presented implicit Q-Learning (IQL), a general algorithm for offline RL that completely avoids any queries to values of out-of-sample actions during training while still enabling multi-step dynamic programming. To our knowledge, this is the first method that combines both of these features. This has a number of important benefits. First, our algorithm is computationally efficient: we can perform 1M updates on one GTX1080 GPU in less than 20 minutes. Second, it is simple to implement, requiring only minor modifications over a standard SARSA-like TD algorithm, and performing policy extraction with a simple weighted behavioral cloning procedure resembling supervised

learning. Finally, despite the simplicity and efficiency of this method, we show that it attains excellent performance across all of the tasks in the D4RL benchmark, matching the best prior methods on the MuJoCo locomotion tasks, and exceeding the state-of-the-art performance on the challenging ant maze environments, where multi-step dynamic programming is essential for good performance.

Table 1: Averaged normalized scores on MuJoCo locomotion and Ant Maze tasks. Our method outperforms prior methods on the challenging Ant Maze tasks, which require dynamic programming, and is competitive with the best prior methods on the locomotion tasks.

Dataset	BC	10%BC	BCQ	DT	ABM	AWAC	Onestep RL	TD3+BC	CQL	IQL (Ours)
halfcheetah-m-v2	42.6	42.5	**47.0**	42.6±0.1	53.6	43.5	**48.4±0.1**	**48.3±0.3**	44.0±5.4	**47.4±0.2**
hopper-m-v2	52.9	56.9	56.7	**67.6±1.0**	0.7	57.0	59.6±2.5	59.3±4.2	58.5±2.1	**66.2±5.7**
walker2d-m-v2	75.3	75.0	72.6	74.0±1.4	0.5	72.4	**81.8±2.2**	83.7±2.1	72.5±0.8	78.3± 8.7
halfcheetah-m-r-v2	36.6	40.6	40.4	36.6±0.8	**50.5**	40.5	38.1±1.3	**44.6±0.5**	**45.5±0.5**	**44.2±1.2**
hopper-m-r-v2	18.1	75.9	53.3	82.7±7.0	49.6	37.2	**97.5±0.7**	60.9±18.8	**95.0±6.4**	**94.7±8.6**
walker2d-m-r-v2	26.0	62.5	52.1	66.6±3.0	53.8	27.0	49.5±12.0	**81.8±5.5**	77.2±5.5	73.8±7.1
halfcheetah-m-e-v2	55.2	**92.9**	**89.1**	86.8±1.3	18.5	42.8	**93.4±1.6**	**90.7±4.3**	**91.6±2.8**	86.7±5.3
hopper-m-e-v2	52.5	**110.9**	81.8	**107.6±1.8**	0.7	55.8	103.3±1.9	98.0±9.4	**105.4±6.8**	91.5±14.3
walker2d-m-e-v2	**107.5**	**109.0**	**109.5**	**108.1±0.2**	3.5	74.5	**113.0±0.4**	**110.1±0.5**	**108.8±0.7**	**109.6±1.0**
locomotion-v2 total	466.7	**666.2**	602.5	**672.6±16.6**	231.4	450.7	**684.6±22.7**	**677.4±44.5**	**698.5±31.0**	**692.4±52.1**
antmaze-u-v0	54.6	62.8	**89.8**	59.2	59.9	56.7	64.3	78.6	74.0	**87.5 ± 2.6**
antmaze-u-d-v0	45.6	50.2	**83.0**	53.0	48.7	49.3	60.7	71.4	**84.0**	62.2 ± 13.8
antmaze-m-p-v0	0.0	5.4	15.0	0.0	0.0	0.0	0.3	10.6	61.2	**71.2 ± 7.3**
antmaze-m-d-v0	0.0	9.8	0.0	0.0	0.5	0.7	0.0	3.0	53.7	**70.0 ± 10.9**
antmaze-l-p-v0	0.0	0.0	0.0	0.0	0.	0.0	0.0	0.2	15.8	**39.6±5.8**
antmaze-l-d-v0	0.0	6.0	0.0	0.0	0.0	1.0	0.0	0.0	14.9	**47.5±9.5**
antmaze-v0 total	100.2	134.2	187.8	112.2	109.1	107.7	125.3	163.8	303.6	**378.0±49.9**
total	566.9	800.4	790.3	784.8	340.5	558.4	809.9	841.2	1002.1	**1070.4±102.0**
kitchen-v0 total	**154.5**	-	-	-	-	-	-	-	144.6	**159.8±22.6**
adroit-v0 total	104.5	-	-	-	-	-	-	-	93.6	**118.1±30.7**
total+kitchen+adroit	825.9	-	-	-	-	-	-	-	1240.3	**1348.3±155.3**
runtime	10m	10m		960m		20m	20m*	20m	80m	20m

*: Note that it is challenging to compare one-step and multi-step methods directly. Also, **brandfonbrener2021offline** reports results for a set of hyperparameters, such as batch and network size, that is significantly different from other methods. We report results for the original hyperparameters and runtime for a comparable set of hyperparameters.

Part III

COMBINED

9

LEARNING NEW SKILLS BY IMAGINING VISUAL AFFORDANCES

9.1 INTRODUCTION

Suppose that you need to learn to open a new kind of drawer in a kitchen. While this new skill might demand some amount of trial and error, you would likely be able to use your mental model and past experience to *imagine* the drawer in the open position, and perhaps even imagine likely intermediate steps, such as grasping the handle, even if you do not yet know precisely how to perform the task. Borrowing the terminology put forward by Gibson **gibson1979ecologicalapproach**, the drawer presents the *affordance* of being "openable," and you are aware of this affordance from your past experience with other similar objects. In fact, infants learn about affordances such as movability, suckability, graspability, and digestibility through interaction and sensory feedback **berger2014development**. Learning and utilizing affordances through interaction allows an agent to acquire diverse, meaningful experiences even in unfamiliar situations. However, this way of learning new skills differs markedly from the approach taken by most robotic learning algorithms: the most widely used exploration methods are generally *undirected*, and focus more on seeking out novelty and surprise **houthooft2016vime**; **tang2017hashtag**; Pathak et al., 2017, rather than familiar and previously seen outcomes. In this paper, we study how robots operating entirely from pixel input can learn about affordances and, when faced with a new and unfamiliar environment, can utilize a previously trained model of possible outcomes to propose potential *goals* that they can practice in this new environment, so as to explore and update their policy efficiently.

We study affordance learning through the framework of self-supervised goal-conditioned reinforcement learning (RL). Learning a new skill in this framework requires generalization in terms of goal setting (*affordances*) and generalization in terms of goal reaching (*behavior*). Prior methods for goal-conditioned RL learn a policy to reach a goal state without the need for an externally provided reward function, and are able to master skills such

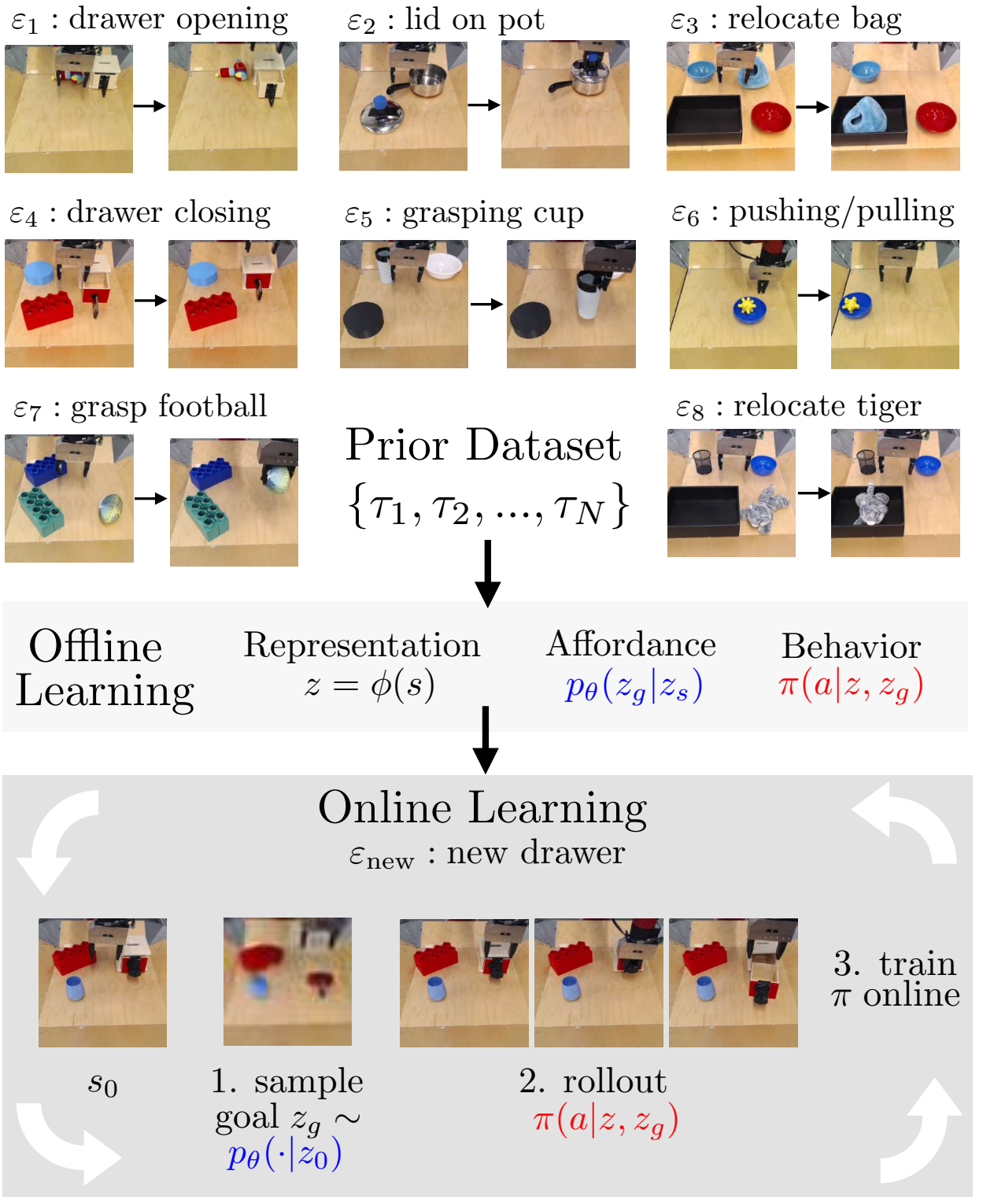


Figure 41: We propose a system for efficient self-supervised robot learning in an unseen environment $\mathcal{E}_{\text{new}}$ by utilizing prior data $\mathcal{D}$ of trajectories from related similar environments $\mathcal{E}_i \sim p(\mathcal{E})$. Tasks are specified via a target goal image. From prior data, we learn an encoder $\mathbf{z} = \phi(\mathbf{s})$ of images (representation) for compressing observations and self-generating rewards, a model of what tasks might be tested in the new environment (affordance), and goal-conditioned policy to accomplish a given task (behavior). While this provides reasonable performance in some test environments, perfecting the policy in test environments may require additional interaction in the test environment

as pushing and grasping objects from image observations **nair2018rig**; **lynch2019play**; **nair2019ccrig**; Agrawal et al., 2016. They learn skills by setting goals to explore, and learning a policy to reach them. However, while these prior works have studied how to learn goal-conditioned policies in individual environments, they do not consider what happens when the robot enters a *new* environment where the policy does not simply generalize zero-shot and needs to be trained further. Our approach to solving tasks in this setting is to learn affordances, represented by a generative model of possible outcomes, and then sample possible affordances in a new environment to explore the new environment efficiently.

Our method, visuomotor affordance learning (VAL), uses expressive conditional models for learning generalizable affordances, along with off-policy RL to learn generalizable behaviors. First, we propose to use more expressive generative models for RL to learn compressed representations of images that can reconstruct unseen objects and help the policy generalize to them. Second, we propose to learn an expressive conditional generative model that generalizes past data to set meaningful goals for novel environments, enabling an understanding of object affordances. Third, we demonstrate how we can use off-policy RL with all prior data to initialize a general-purpose goal-conditioned policy, then fine-tune the policy with additional data to master a skill in a new environment. Combining RL and representation learning, we show that we are able to learn skills with a small amount of online exploration on novel objects in real-world robotic scenarios such as object grasping, relocation, and drawer opening and closing.

The main contribution of this work is to present a learning system that can learn robotic skills entirely from image pixels in novel environments by utilizing prior data to generate goals and generalize behavior in new settings. We demonstrate that our method can learn complex manipulation skills including grasping, drawer opening, pushing, and object re-positioning for a diverse set of objects in simulation. We also demonstrate our method in the real world on a Sawyer robot, where our method is able to learn tasks such as grasping and placing unseen objects and opening and closing unseen drawers after only five minutes of online interaction.

9.2 RELATED WORK

RL has been applied previously to robotic manipulation **kober2008mp**; **peters2010reps**; Levine et al., 2016, and also various other applications from playing games **mnih2013atari**; **silver2016alphago** to locomotion **benbrahim1997biped**; **kohl2004quadruped**; **deisenroth2011pilco**; **williams2017mpc**. Such approaches require an external reward function, but obtaining

this reward function itself poses a challenge to exploring in novel uninstrumented environments as we consider in this paper. Thus, in this work we focus on self-supervised RL methods that that do not assume externally provided reward functions.

When learning without an externally provided reward function, one common idea is to use novelty-based intrinsic reward functions **chentanez2005intrinsically**; **lopes2012exploration**; **houthooft2016vime**; **stadie2016exploration**; Bellemare et al., 2016; Pathak et al., 2017. State novelty-based methods eventually visit all possible states, but do not necessarily learn a useful policy from purely optimizing the exploration objective. An alternative exploration framework that learns a useful policy even solely from the intrinsic objective is goal reaching **Baranes2012**; **nair2018rig**; **nachum2018hiro**; **held2018goalgan**; **Pere2018**; **wadefarley2019discern**; **pong2020skewfit**; Kaelbling, 1993; Schaul et al., 2015; Andrychowicz et al., 2017: by picking a distance measure between states, setting goals, and attempting to reach them, an agent can discover all potential goals in its environment. We refer the reader to the survey of Colas et al. **colas2021gepsurvey** for a full classification and discussion of these methods. However, these prior methods do not study the question of how to set goals in new environments, which is vital for collecting coherent experience when faced with a new task. Our method utilizes representation learning and off-policy RL to generalize prior experience to set exploration goals in new settings.

Most similar to our work, context-conditioned reinforcement learning with imagined goals (CCRIG) learns a conditional variational auto-encoder (CVAE) **sohn2015cvae** that generates goals conditional on the current scene **nair2019ccrig**. CCRIG was able to learn pushing skills that generalized mainly to object color and partially to object geometry. Our work differs from CCRIG in a number of ways. First, we learn expressive generative models that are able to generate goals in scenes with significantly more visual diversity. Second, we learn a diverse set of skills (e.g., grasping, drawer opening, object placing) that require the goal generation to understand affordances of the environment. Finally, we show that we can use off-policy RL on prior experience, in addition to fine-tuning further on a single specific task to learn new skills. These differences allow our method to better operate in real-world scenarios, as borne out in our experiments.

Another line of work explores the use of affordances in RL, robotics, and control, historically through the lens of perception **zech2017affordances**; **hassanin2018affordances**; **yamanobe2018affordances**. Affordances have also been discussed previously in reinforcement learning in order to accelerate planning in model-based RL by planning over only a subset of relevant actions **abel2014affordances**; **khetarpal2020affordances**; **xu2021affordances**. Our work is more related to the view of affordances in develop-

mental robotics, where affordances were hypothesized to be useful for learning general manipulation skills **hart2010affordances**; **min2016affordances**. In our work, we show how goal-conditioned RL utilizing from prior data can put these ideas into practice on real-world robotics systems.

9.3 PRELIMINARIES

In this section, we cover preliminaries on RL, goal-conditioned RL, and self-supervised visual RL.

Goal-conditioned reinforcement learning. In goal-conditioned RL, we augment the standard Markov decision process (MDP), which is defined in terms of states $\mathbf{s}_t \in \mathcal{S}$, actions $\mathbf{a}_t \in \mathcal{A}$, and environment dynamics $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$, with goals $\mathbf{g} \in \mathcal{G}$ that represent the agent's intention to perform one of a variety of tasks drawn from the task family $p(\mathbf{g})$. The reward function is also goal-conditioned, and given by some function $r(\mathbf{s}, \mathbf{g})$. The discounted return is defined as $R_t = \sum_{i=0}^{H} \gamma^i r(\mathbf{s}_i, \mathbf{g})$, where γ is a discount factor and H is the horizon, which may be infinite. The aim of the agent is to optimize a policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g})$ to maximize the expected discounted return $J(\pi) = \mathbb{E}_{\mathbf{g}}[R_0]$. Efficient off-policy RL algorithms have been proposed to learn goal-conditioned policies Schaul et al., 2015; Andrychowicz et al., 2017.

Self-supervised visual reinforcement learning. For scalable robot learning, we cannot always assume known shaped reward functions for tasks. In the absence of such reward functions, Andrychowicz et al. propose goal-state reaching as a natural objective Andrychowicz et al., 2017: tasks are defined by state outcomes, where the goal space $\mathcal{G} = \mathcal{S}$, the state space, and the reward is a goal-reaching objective $r(\mathbf{s}, \mathbf{g}) = -\mathbb{1}_{\|\mathbf{s}-\mathbf{g}\|>\epsilon}$. The task distribution $p(\mathbf{g})$ is chosen to be the feasible states of the robot and objects it interacts with.

But when states are high-dimensional (e.g., images), two issues arise: we do not know the task distribution $p(\mathbf{g})$, and exact reaching of a target goal state is impractical. Reinforcement learning with imagined goals (RIG) **nair2018rig** addresses these issues with a generative model, which is used to learn a latent space of observations and similarity metric on images. Specifically, a variational auto-encoder Kingma and Welling, 2014 with encoder $\phi(\mathbf{z}_t|\mathbf{s}_t)$ and prior $p(\mathbf{z})$ is learned. At training time, the robot sets goals for itself in latent space by sampling a goal latent $\mathbf{z}_g \sim p(\mathbf{z})$ and learns a policy $\pi(\mathbf{z}_t, \mathbf{z}_g)$ to reach latent goals. At test time, the robot can be tasked with a goal image $\mathbf{g}$ and execute the learned policy with goal latent $\mathbf{z}_g \sim \phi(\cdot|\mathbf{g})$ to match the goal image. In this way, goal-conditioned RL with generative models enables self-supervised learning in a single

environment $\mathcal{E}$ where the task distribution $p(\mathbf{g})$ is not known apriori.

9.4 PROBLEM SETTING

In this paper, we now consider acting in a distribution of environments $p(\mathcal{E})$ with shared structure.[1] Each environment $\mathcal{E}_i$ has its own task distribution $p_i(\mathbf{g})$. As before, tasks are defined by goal states, so $p_i(\mathbf{g})$ represents the potential outcomes of interest in that environment. We assume that the outcomes of interest depends only on the appearance of the environment. When $\mathcal{E}$ is fully observed, this means $p(\mathbf{g}|s_0)$ is shared across environments. As prior data, the robot has access to a training dataset $\mathcal{D} = \{\tau_1, \tau_2, \ldots \tau_N\}$ of trajectories from prior environments, where the trajectories achieve a final outcome $\mathbf{s}_T \sim p_i(\mathbf{g})$ – that is, they succeed on tasks from the underlying task distribution for that environment. Now, the robot is placed in a new environment $\mathcal{E}_{\text{new}} \sim p(\mathcal{E})$, and must learn to solve tasks in the new environment through self-supervised practice at training time, such that it can accomplish tasks sampled from $p_i(\mathbf{g})$ at test time. The environments we consider vary visually and dynamically in terms of the objects that are present and potential tasks they afford, such as being able to lift various objects and open different drawers; such real-world variation is presented in Figure 41.

Thus, the agent in this new setting must generalize its prior experience $\mathcal{D}$ to practice potential skills it may be asked to perform at test time in the new environment efficiently, even when it encounters novel objects. At test time, the robot is evaluated in terms of its ability to accomplish a task in $\mathcal{E}_{\text{new}}$ specified by a goal image. To perform well in this setting, a method must attempt to infer $p_{\text{new}}(\mathbf{g})$, which should be possible since $p(\mathbf{g}|s_0)$ is common across environments and $\mathcal{D}$ contains trajectories that achieve goals from the same distribution. Note that given observations from $\mathcal{E}_{\text{new}}$, the agent may still have to practice the various behaviors the environment affords if there are multiple, since the test task distribution is unknown at training time.

9.5 VISUOMOTOR AFFORDANCE LEARNING

In this section, we present visuomotor affordance learning (VAL), our method for self-supervised learning in novel environments utilizing prior data from related environ-

1 Meta-learning approaches have also studied learning a new task quickly, given experience on a set of related MDPs **duan2016rl2**; **finn2017maml**; **rakelly2019pearl**. The aims of our method are related to meta-learning, in that we also aim to learn in new environments more quickly, but we do not assume being given user-specified tasks or rewards to solve in the new environment.

Algorithm 7 Visual Affordance Learning

Require: Dataset $\mathcal{D}$, policy $\pi(\mathbf{a}|\mathbf{z}, \mathbf{z}_g)$, Q-function $Q(\mathbf{z}, \mathbf{a}, \mathbf{z}_g)$, RL algorithm $\mathcal{A}$, replay buffer $\mathcal{R}$, relabeling strategy $p_{RS}(\mathbf{z})$, environment family $p(\mathcal{E})$.

Learn encoder $\phi(\mathbf{z}|\mathbf{s})$ by generative model of $\mathcal{D}$
Learn affordances $p(\mathbf{z}_t|\mathbf{z}_0)$ by generative model of $\mathcal{D}$
Add latent encoding of $\mathcal{D}$ to the replay buffer
Initialize π and Q by running $\mathcal{A}$ offline
Sample $\mathcal{E}_{new} \sim p(\mathcal{E})$, $\mathcal{E}_{new} = (p_{new}(\mathbf{s}_0), \pi_{new}(\mathbf{s}_{t+1}|\mathbf{s}, \mathbf{a}))$
for $1, \ldots, N_{episodes}$ **do**
  Sample initial state $\mathbf{s}_0 \sim p_{new}(\mathbf{s}_0)$.
  Sample goal $\mathbf{z}_g \sim p(\mathbf{z}_t|\mathbf{z}_0)$
  for $t = 0, \ldots, H$ **do**
    Sample $\mathbf{a}_t \sim \pi(\cdot|\mathbf{z}_t, \mathbf{z}_g)$
    Sample $\mathbf{s}_{t+1} \sim p_{new}(\cdot|\mathbf{s}_t, \mathbf{a}_t)$
  end for
  Store trajectory $(\mathbf{z}_1, \mathbf{a}_1, \ldots, \mathbf{z}_H)$ in replay buffer $\mathcal{R}$.
  for $1, \ldots, N_{train\_steps}$ **do**
    Sample transition $(\mathbf{z}_t, \mathbf{a}_t, \mathbf{z}_{t+1}, \mathbf{z}_g)$
    Relabel with $\mathbf{z}'_g \sim p_{RS}(\mathbf{z}_g)$ and recompute reward
    Update π and Q with relabeled transition using $\mathcal{A}$
  end for
end for=0

ments. VAL consists of three learning phases: (A) an affordance learning phase to learn affordances from the prior data, (B) an offline behavior learning phase to learn behaviors from the prior data, and (C) an online behavior learning phase where the agent actively interacts with the test environment using affordances and learns potential behaviors in the new environment. The overall method is summarized in Algorithm 7.

9.5.1 *Affordance Learning*

The affordance learning system must generate goals that induce coherent exploration trajectories even in new environments. We would like to learn a model that, given an observation s_0 in a new environment, generates a potential goal state the agent might be tasked to reach. With high-dimensional image observations, attempting to sample such goal states directly is a difficult generative modeling problem. Instead, we first learn a lower-dimensional latent space $p(\mathbf{z}_t|\mathbf{s}_t)$ by training a generative model through image reconstruction. With sufficiently expressive models and enough data, the gener-

ative model represents images in a manner that it can reconstruct even unseen objects. Given such a latent space, we can then learn affordances by training a conditional model $p(\mathbf{z}_t|\mathbf{z}_0)$ to generate goals that are plausible outcomes of an initial state, even for unseen environments.

To instantiate these two models, we must choose a class of latent variable generative models for $p(\mathbf{s}_t|\mathbf{z}_t)$ and conditional affordance models $p(\mathbf{z}_t|\mathbf{z}_0)$. For the first, while many choices would be suitable, including models such as variational auto-encoders (VAEs) Kingma and Welling, 2014 and generative adversarial networks (GANs) **goodfellow2014gan**; **donahue2017bigan**, in our implementation we use the VQVAE model **oord2017vqvae**. The VQVAE is expressive enough to represent very diverse datasets, and be able to reconstruct even unseen objects with a high level of detail. We do not require image reconstructions for our method, but the ability to partially reconstruct unseen objects suggests that model expressively represents geometry and color information that may be important for learning affordances and behaviors. In the VQVAE case, ϕ is deterministic, so we will use the shorthand for the latent embedding $\mathbf{z}_t = \phi(\mathbf{s}_t)$, where $\mathbf{z}_t$ is the continuous latent resulting after quantization. In our experiments, we compare this choice of model to other expressive models: VAE Kingma and Welling, 2014, CVAE **sohn2015cvae**, and BiGAN **donahue2017bigan**.

Next, given a latent space, we need to sample potential goals from this space that is predictive of which goals might be tasked at test time in the new environment $\mathcal{E}_{\text{new}}$. We use a conditional PixelCNN model **oord2016pixelcnn** in the latent space to do so, conditioned on the initial state s_0. The PixelCNN model is trained to maximize $\mathbb{E}_{\tau\sim\mathcal{D},(\mathbf{s}_0,\mathbf{s}_t)\sim\tau,z\sim\phi(\cdot|\mathbf{s})}[\log p_\theta(\mathbf{z}_t|\mathbf{z}_0)]$, where θ is the parameters of the PixelCNN density model. To generate exploration goals, we sample $z_g \sim p_\theta(\cdot|\mathbf{z}_0)$, where $\mathbf{z}_0 = \phi(\mathbf{s}_0)$ is the encoding of an image of the current setting. The PixelCNN model in VAL being conditional allows us to generate meaningful goals that might be achievable with a novel object. As we show in Section 9.6.1, the conditional model allows us to sample affordances such as opening drawers and lifting objects, even for new environments that are visually complex with variation in the identity, color, geometry, and functionality of objects.

9.5.2 *Offline Behavior Learning*

Given learned affordances of what behaviors the agent may be tasked with, the agent needs to learn how to actually accomplish those behaviors. In this phase, we wish to learn a reasonable goal-conditioned policy with offline RL. While trained on a limited

Figure 42: Samples on unseen objects in the real world. In each column, the top image is the conditioning image s_0 and the images below are conditionally sampled images from the corresponding generative model. Our model, CVQVAE, generates clear and diverse samples.

fixed offline dataset, the hope is that the policy can generalize learning from offline data to accomplish desired behaviors in a new environment, allowing us to either perform tasks successfully zero-shot without further learning, or collect meaningful exploration data in the next phase (Section 9.5.3).

To learn with offline RL while allowing the possibility of quickly fine-tuning in a new environment, we use advantage weighted actor critic (AWAC) **nair2020awac** as the underlying RL algorithm. AWAC is an off-policy RL algorithm that has shown strong performance in utilizing prior data for offline pretraining while still being amenable to online fine-tuning. The aim of behavior learning is to optimize a goal-conditioned policy $\pi(\mathbf{a}|\mathbf{z}, \mathbf{z}_g)$ which is able to solve any task in the task distribution $p(\mathbf{z}_g)$. We do not assume an external reward function, so we require a reward function to optimize. Following prior work **nair2018rig**; Andrychowicz et al., 2017, we optimize a goal reaching objective: to maximize the density $p_{\mathbf{z}_g}(\mathbf{z} = \mathbf{z}_g)$; in practice we use the sparse reward function $r(\mathbf{z}, \mathbf{z}_g) = -\mathbb{1}_{\|\mathbf{z}-\mathbf{z}_g\|>\epsilon}$, where ϵ is a fixed threshold as it encourages fully solving tasks in a binary fashion. For a particular transition in the replay buffer $(\mathbf{z}_t, \mathbf{a}_t, \mathbf{z}_{t+1}, \mathbf{z}_g, r)$, we can also relabel the goal with a new goal $\mathbf{z}'_g$ and the recompute the reward. In practice we keep $\mathbf{z}_g$ with 20% probability, future hindsight experience replay with 40% probability, and sample $\mathbf{z}'_g \sim p(z_t|z_0)$ with 40% probability. Importantly, due to using a compressed representation, we can expect that z_t in any new environment will be semantically similar to past experience. Thus, in this phase, we can obtain a policy that generalizes partially to a new environment.

9.5.3 *Online Behavior Learning*

Guaranteeing zero-shot generalization for every possible new environment is in general impossible. Instead, we will discuss how to finetune in a specific environment $\mathcal{E}_{\text{new}}$ using

affordances. In this phase, we utilize the learned affordances which inform *what* tasks to perform, and the offline learned behaviors that inform *how* to perform those tasks.

At online training time, the new task distribution $p_{new}(\mathbf{g})$ is unknown, so we use the affordance model to sample potential tasks. Thus, to collect coherent exploration data, we sample goals from the affordance module $z_g \sim p_\theta(\cdot|z_0)$ and roll out the goal-conditioned policy $\pi(\mathbf{a}|\mathbf{z}, \mathbf{z}_g)$. We then iterate between improving the policy with off-policy RL, and collecting exploration data and appending it to the replay buffer. To learn a new task, exploration data in the new environment for fine-tuning the policy is extremely valuable, so this iteration allows us to quickly fine-tune. The online learning process is illustrated in the bottom box in Figure 41.

In summary, VAL enables self-supervised learning in novel environments utilizing prior data. We first learn a generative model for learning a latent space and affordances, and learn how to accomplish tasks with off-policy RL. Then, both are used in an online behavior learning phase to perfect potential behaviors in a new environment. The overall method is summarized in Algorithm 7.

9.6 Real-world experimental evaluation

We first evaluate our method in a real-world manipulation setting with a Sawyer robot and diverse objects and tasks. This setting is very challenging due to the variety of objects, scenes, and tasks that the robot interacts with, and being limited to using image inputs.

Real-world setting. The real-world setting is shown in Figure 41. A Sawyer robot is controlled at 5Hz with 4 degrees of control: 3 dimensions of end-effector velocity control in Euclidean space and one dimension to open and close the gripper. The environments span 10 drawer handles, 10 pot handles, 40 toys, and 60 distractor objects. To create a new environment $\mathcal{E}$ in the real world, we randomly sample one interaction object as well as 2-3 distractor objects. The behaviors that the environments afford therefore span grasping and relocating toys, opening and closing drawers, as well as covering and uncovering pots. Each test environment contains an unseen interaction object and a random set of 2-3 distractor objects, with all positions randomized.

Our experiments aim to answer the following questions:

1. Does the learned generative model sample plausible affordances in new scenes?
2. Is VAL able to accelerate learning of real-world manipulation skills online in new settings?

9.6.1 *Generative Models for Affordance Learning*

Expressive generative models enable us to propose desired outcomes in new environments even when the current policy cannot yet achieve them. We can preview this capability by inspecting the training procedure of the models, which also gives insights into which models will tend to perform well when used for interactive learning. Specifically, we inspect model samples to see if the model can actually output plausible candidate affordances. We evaluate our model, which combines a VQVAE with a conditional pixel CNN, and prior models that have been used in self-supervised RL as well as other expressive generative models. We compare (1) VAE Kingma and Welling, 2014, (2) CVAE **sohn2015cvae**, (3) BiGAN **donahue2017bigan**, (4) Conditional BiGAN, (5) Ours, a VQVAE with a conditional PixelCNN.

Sampled affordances are shown in Figure 42. Our model is able to sample coherent tasks to perform where other methods produce indistinct or uninterruptible samples. For example, the VQVAE model turns an unseen drawer (with a wide handle) into one with a thin handle, which exists in the prior data. This illustrates its ability to utilize prior data for generating conducive goals for online RL. CCVAE samples are usually less coherent, often missing the object completely and not capturing the geometry of unseen objects. The CBiGAN also struggles to produce realistic images, while the VAE fails completely as it is not a conditional model cannot represent the wide diversity of potential images.

9.6.2 *Real-World Visuomotor Affordance Learning*

Next, we investigate whether VAL can handle real-world visual complexity and object diversity to learn control policies for varied tasks on a Sawyer robot. The setup is shown in Figure 41: the robot is tasked with fine-tuning in a particular environment, beginning with about 1,000 trajectories of prior data for affordance learning and offline behavior learning. The protocol for collecting prior data, as well as examples of images from the data are shown in Appendix **??**.

After pretraining affordances, a policy, and a Q function on the prior data, the robot is dropped in a new test environment. We evaluate five test environments which each contain distractor objects as well as objects that may be interacted with such a shoe that can be picked up, drawers than can be opened or closed, and a lid which may be placed on a pot. These behaviors are demonstrated on similar objects in the prior dataset, but the objects during test time are previously unseen - for instance, the drawer has a

different handle. The agent can interact with the environment without supervision to collect more data and improve the policy; then to evaluate, the agent is tasked with a specific goal image that corresponds to a task such as opening a closed drawer.

Results running VAL in the real world are shown in the table in Figure 43, reporting the success rates on five test tasks for our method and CCRIG both offline and after one epoch of online training, which includes 10 interactive trials, amounting to less than five minutes of real-world interaction time. On all five tasks, VAL shows nontrivial offline performance followed by strong online improvement. Qualitatively, our method is able to learn behaviors such as recovering from a missed grasp by returning to the grasp. Film strips of our method are shown on the right side of Figure 43. In contrast, CCRIG struggles to make progress on all five tasks. CCRIG fails for two reasons. First, the quality of the sampled affordances are significantly poorer. Second, as can be seen in videos, CCRIG sometimes comes close to solving the task, but does not fully solve it, preventing improvement after subsequent training.

9.7 EXPERIMENTAL EVALUATION IN SIMULATION

In order to further study and understand VAL, we carry out more experiments in simulation, where we can better control the quantity of data and ablate parts of the method. These experiments aim to answer the following questions:

1. Does VAL outperform prior self-supervised RL methods in accelerating learning a new task from prior data?
2. Can VAL scale with additional data for affordance and behavior learning?

Simulated setting. Randomly generated workspaces in our simulated multi-task environment are shown in Figure 47. In this PyBullet simulation **coumans2021**, the robot is faced with multiple potential tasks in each environment based on which objects are present: opening and closing a drawer by the handle, opening and closing a different drawer by pressing a button, grasping objects, and moving objects into drawers or a box. To sample a new environment $\mathcal{E}$, we randomize the existence, position, color, and orientation of the following: two drawers, a box, a button, and an object. If an object is present it is chosen from a set of 84 object geometries. To successfully learn in a test environment, the method must be able to explore in the environment based on the behaviors that environment affords.

9.7.1 *Self-Supervised Online Fine-Tuning from Prior Data*

We first compare VAL against prior methods in this simulated setting. The robot receives a prior dataset $\mathcal{D}$ of trajectories in the pre-training environments, which is utilized for learning affordances and offline RL. The details of this dataset are explained further in Appendix **??**. After the pre-training phase, the robot is placed in a test setting and begins online fine-tuning. We evaluate the policy on goal images in the new environment, sampled from expert trajectories, and report whether the final state of the policy's trajectory matches the state of the goal image within a chosen threshold.

Learning curves of the online training phase for various objects are shown in Figure 44. We see that for each task, VAL outperforms prior methods which do not make progress on learning these tasks at all. On the drawer opening task, offline RL achieves nontrivial performance of about 60%, but online interaction allows fine-tuning to over 90% success rate.

How is VAL able to outperform prior methods so significantly? One major reason is the quality of sampled goals. Samples for the affordance model for the simulated domain are shown in Figure 47. We compare (1) VAE Kingma and Welling, 2014, (2) CVAE **sohn2015cvae**, (3) BiGAN **donahue2017bigan**, (4) Conditional BiGAN, (5) Ours, a VQVAE with a conditional PixelCNN. Our model produces diverse, coherent samples of possible outcomes (i.e., affordances) in new scenes with novel objects. In comparison to our model, conditional VAE samples tend to be blurry and do not capture the geometry of unseen objects well.

9.7.2 *Scalable Robot Learning with VAL*

For general-purpose robot learning, we would ideally like to learn diverse skills continuously, using prior experience to perpetually improve at learning new skills. To study this possibility, we examine whether solving tasks in a new environment can be sped up by collecting larger amounts of prior experience. In this experiment conducted in simulation in order to carefully control the data quantity, the robot receives a prior dataset $\mathcal{D}$ of K trajectories for running VAL to grasp an unseen object in a new environment. We vary K to observe whether the method can benefit from larger amounts of prior data.

Learning curves of the online training phase, averaged over five test objects, are shown in Figure 48. First, we see from the starting point of each curve that the offline policy already generalizes to some level for grasping objects, but the average success rate is only around 35%. Importantly, note that training on more data only slightly improves gener-

alization after offline training (at timestep 0). Then, fine-tuning results in rapid policy improvement up to around 65% success rate after only 150,000 timesteps when utilizing the most prior data, compared to 40% with the least prior data. Thus, more prior data significantly accelerates learning in new environments even when the initial performance is comparable. This suggests that VAL can be deployed in a continual learning setting, with each new task being learned faster as it benefits from the increasing dataset size.

9.8 CONCLUSION

We present visuomotor affordance learning (VAL), a method for learning tasks online in a new environment without supervision, utilizing trajectories from other related environments. VAL uses expressive generative models to learn visual affordances, combines these affordances with off-policy goal-conditioned RL to learn skills offline, and then fine-tunes in a new environment online. Like deep learning in domains such as computer vision Krizhevsky et al., 2012 and natural language processing **devlin2019bert** which have been driven by large datasets and generalization, robotics will likely require learning from a similar scale of data. In future work, VAL could enable such systems by allowing autonomous collection of coherent exploration data in diverse real-world settings.

Task	VAL (Ours) Offline → Online	CCRIG Offline → Online
(1) Pickup shoe	12.5% → **50%**	0% → 16.6%
(2) Drawer closing	25% → **100%**	0% → 12.5%
(3) Drawer opening	62.5% → **100%**	0% → 0%
(4) Place object in tray	25% → **75%**	0% → 0%
(5) Lid on pot	37.5% → **87.5%**	0% → 0%

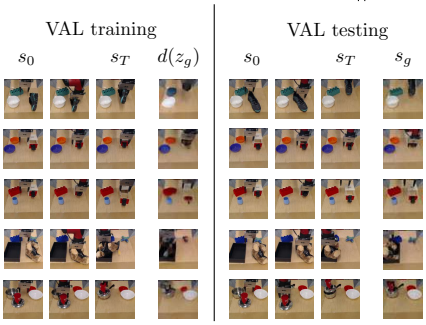


Figure 43: Real-world results. Left, success rates per method for the five tasks tested. We report the offline performance, followed by the performance after five minutes of online fine-tuning. With VAL, we see reasonable initial offline performance followed by significant improvement on all tasks. Meanwhile CCRIG fails to succeed any of the tasks. Right, film strips of VAL during training (left, with decoded affordance proposals) and testing (right, with goal images) are visualized. Videos are available at https://sites.google.com/view/val-rl

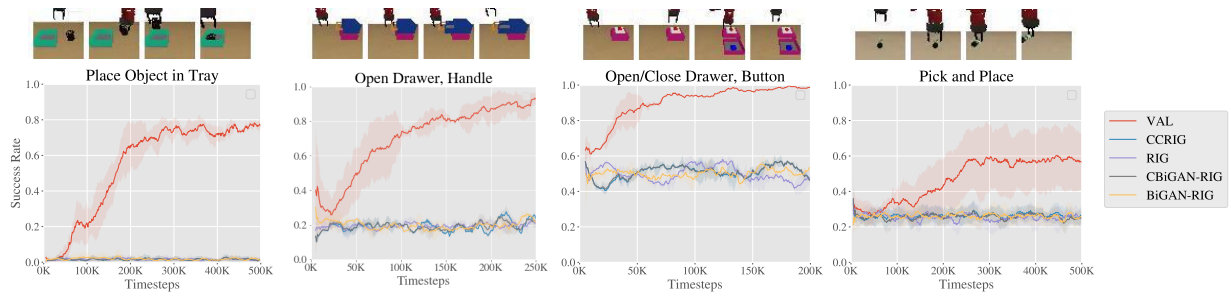


Figure 44: Learning curves for simulation experiments, fine-tuning on an unseen environment. Our method is able to learn these tasks online, while none of the baselines or prior methods are able to make meaningful learning progress in this setting. A successful rollout of each task in a test environment is shown above the corresponding plots.

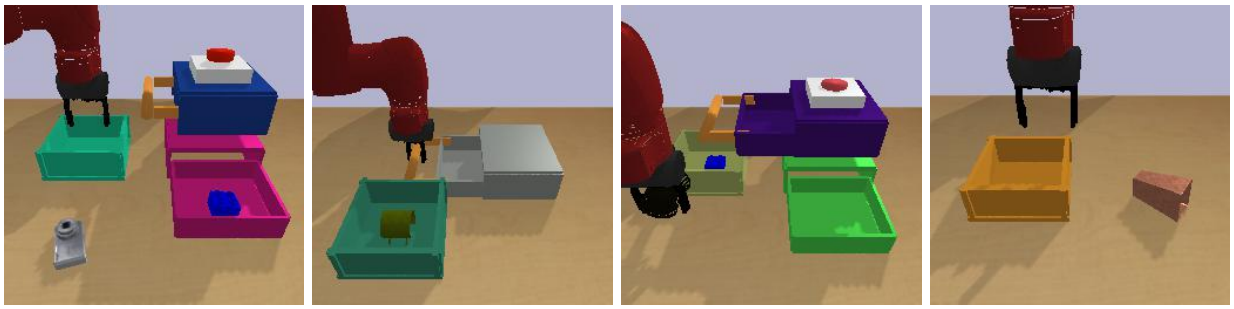


Figure 45: Randomly sampled scenes from our simulated multi-task environment. To practice in a sampled scene, the agent must infer the potential behaviors that the scene affords.

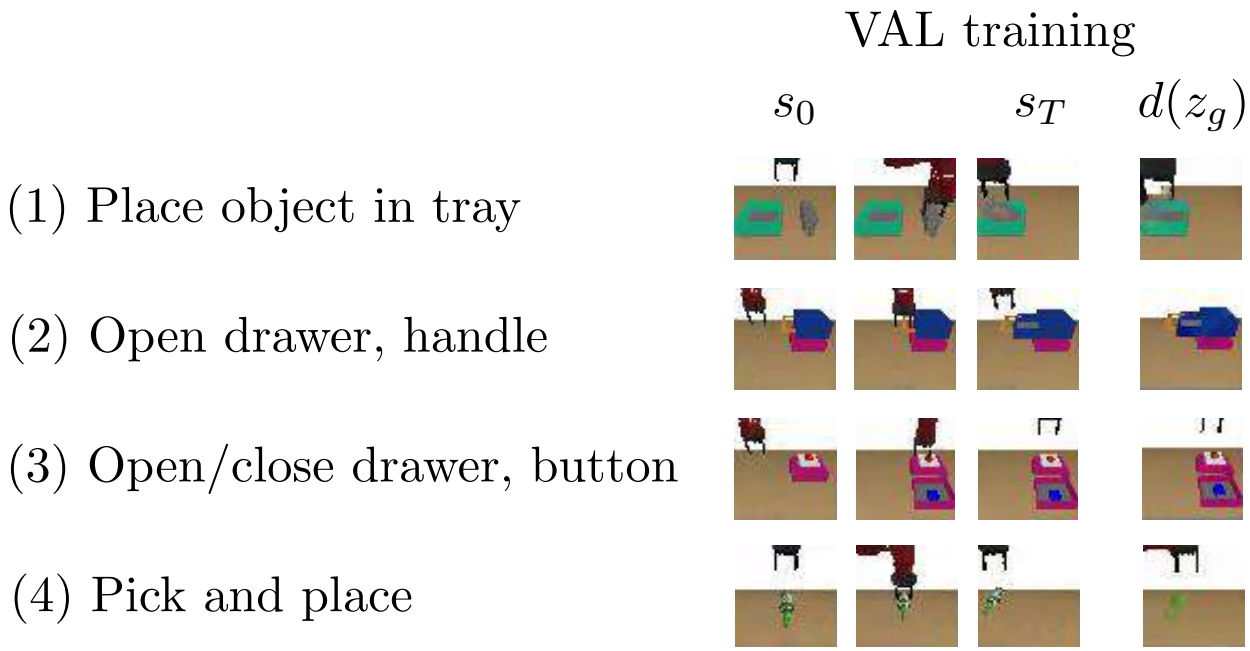


Figure 46: Training rollouts from VAL. For each environment, the reconstruction of a sampled affordance is shown on the rightmost column, and frames from a trajectory attempting to achieve that affordance is shown on the left.

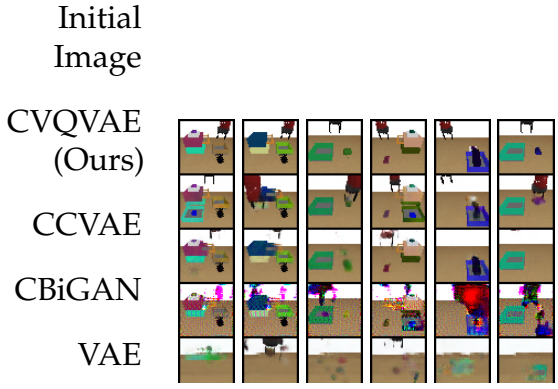


Figure 47: Samples on test environments in simulation. In each column, the top image is the conditioning image s_0 and the images below are conditionally sampled images. The left four columns are relatively successful samples for our affordance model, each showing the potential outcome of a behavior. The right two columns are failure modes, lacking diversity or altering an object's geometry or color.

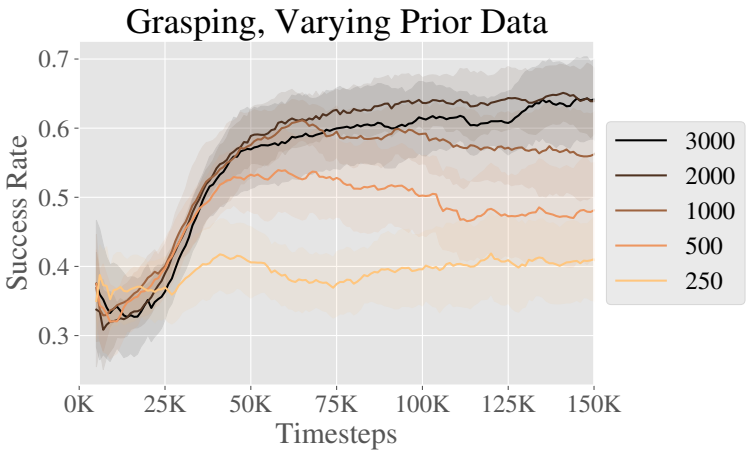


Figure 48: Learning curves for simulated grasping of novel objects with VAL, using data from an increasing number of training objects for offline RL. We collect 50 trajectories per training object, and each line is labeled with the total number of training trajectories.

10

PLANNING TO PRACTICE: EFFICIENT ONLINE FINE-TUNING BY COMPOSING GOALS IN LATENT SPACE

10.1 INTRODUCTION

Developing controllers that can solve a range of tasks and generalize broadly in real-world settings is a long-standing challenge in robotics. Such generalization in other domains such as computer vision and natural language processing has been attributed to training large machine learning models on massive datasets Krizhevsky et al., 2012. Consequently, one promising approach to robustly handle a wide variety of potential situations that a robot might encounter is to train on a large dataset of robot behaviors. Prior work in robotics has demonstrated effectively generalization from training on large datasets for individual tasks, such as grasping **yu2021conservative**; Levine et al., 2017. However, a general-purpose robot should be able to perform a wide range of skills, and should also be *taskable*. That is, it must be able to complete a specific task when specified by a human, including temporally extended tasks that require sequencing many skills together to complete. This topic has been studied in prior work on goal-conditioned reinforcement learning, where a robot aims to perform a task given a desired end state **Khazatsky2021WhatCI**; **chebotar2021actionable**; **kalashnikov2021mtopt**. What remains to make these methods widely applicable for real-world robotics?

While goal-conditioned policies can be trained effectively for relatively short-horizon tasks, temporally extended multi-stage can pose a significant challenge for current methods. These tasks present a major exploration challenge during online learning, and a major challenge for credit assignment during offline learning. In this paper, we aim to address these challenges by combining two ideas. The first is that long-horizon goal-reaching tasks can be decomposed into shorter-horizon tasks consisting of subgoals. The second is that these subgoals can be used to *fine-tune* a goal-conditioned policy online, even if its performance from offline data is poor. The first idea enables us to address

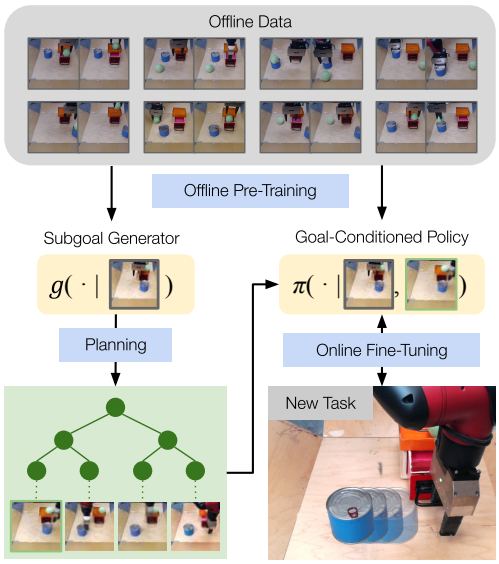


Figure 49: Our method, Plan to Practice (PTP), solves long-horizon goal-conditioned tasks by combining planning and fine-tuning. We begin with an offline dataset containing a variety of behaviors, and train a subgoal generator and goal-conditioned policy on this data. Then, to learn a more complex multi-stage tasks, we optimize over subgoals using the subgoal generator, which corresponds to a planning procedure over (visual) subgoals, and fine-tune the policy with online RL by practicing these subgoals. This enables the robot to solve multi-stage tasks directly from images.

the exploration challenge, by automatically generating intermediate subgoals that can be "practiced" on the way to a longer-horizon final goal. In the framework of goal-conditioned RL, solving long-horizon tasks can be reduced to the problem of optimization over a sequence of subgoals for the goal-conditioned policy, and this optimization over subgoals can be regarded as a kind of high-level planning, where the optimizer selects waypoints for achieving a distant goal. The high-level planner itself can use a learned high-level model.

However, if we rely entirely on offline data, credit assignment challenges make it difficult to perform longer-horizon tasks even with subgoal planning. Even if the offline RL policy performs well on each individual skill, there may be errors from stitching skills together because the initial states of each stage diverge from the offline data when they are composed together. In practice, this leads to poor performance when using only offline training. Therefore, the second key idea in our work is to utilize subgoal planning not merely to *perform* a multi-stage task, but also to make it possible to *practice* that task to finetune it online. While online training for temporally extended tasks is ordinarily difficult, by addressing the exploration challenge with subgoal planning, we make it possible for the robot to practice a series of relatively short-horizon tasks, which

makes this kind of finetuning feasible. Thus, the planner acts both as a higher level policy when performing the task, and as a scaffolding curriculum for finetuning the lower-level goal-conditioned policy By collecting data actively in a specific environment, we can directly experience the distribution shift and can use reinforcement learning to improve performance under this shift.

To this end, we propose Planning to Practice (PTP), an approach that efficiently trains a goal-conditioned policy to solve multi-step tasks by setting subgoals to exploit the compositional structure of the offline data. An outline is shown in Fig. 49. Our approach is based around a planner that composes generated subgoals to guide the goal-conditioned policy during an online fine-tuning phase. To propose diverse and reachable subgoals to form the candidate plans, we design a conditional subgoal generator based on conditional variational autoencoder (CVAE) **sohn2015cvae**. Through training on the offline dataset, the conditional subgoal generator captures the distribution of reachable subgoals from a given state and generates sequences of subgoals from the learned latent space in a recursive manner. Our subgoal planning algorithm hierarchically searches for subgoals in a coarse-to-fine manner using multiple conditional subgoal generators that trained to generate goals at different temporal resolutions. Both the goal-conditioned policy and the conditional subgoal generators are pre-trained on the offline data, and the policy is fine-tuned on the novel target task.

Our main contribution is a system for learning to solve long-horizon goal-reaching tasks by fine-tuning the goal-conditioned policy with subgoal planning in a learned latent space. We evaluate our approach on multi-stage robotic manipulation tasks with raw image observations and image goals in both simulation and the real world. After being pre-trained on short demonstrations of primitive interactions, our approach is able to find feasible subgoal sequences as plans for unseen final goals by recursively generating subgoals with the learned conditional subgoal generators. By comparing our approach with both model-free methods and prior approaches that optimize over subgoals, we demonstrate that the produced plans significantly improve the learning efficiency and the resultant success rates during the online fine-tuning.

10.2 RELATED WORK

We propose to use a combination of optimization-based planning and fine-tuning with goal-conditioned reinforcement learning from prior data in order to allow robots to learn temporally extended skills. In this section, we cover prior methods in offline RL, planning, goal-conditioned RL, and how they relate to our method.

Learning from prior data. Offline reinforcement learning methods learn from prior data **lange2012batch**; **fujimoto2019off**; **kumar2019stabilizing**; **zhang2021brac**; **kumar2020conservativ** **fujimoto2021minimalist**; **singh2020cog**, and can also finetune through online interaction **nair2020awac**; **villaflor2020finetuning**; **lu2021awopt**; **Khazatsky2021WhatCI**; **lee2021finetuning**; **meng2021starcraft**. Such methods have been used in a variety of robotic settings **kalashnikov2018scala** **cabi2019scaling**; **kalashnikov2021mtopt**; **lu2021awopt**. Our focus is not on introducing new offline RL methods. Rather, our work shows that planning over subgoals for a goal-conditioned policy that is pretrained offline can enable finetuning for temporally extended skills that would otherwise be very difficult to learn.

Goal-conditioned reinforcement learning. The aim of goal-conditioned reinforcement learning (GCRL) is to control the agent to efficiently reach specified goal states **Kaelbling1993Learning** **Schaul2015UniversalVF**; **Eysenbach2021CLearningLT**. Compared to policies that are trained to solve a fixed task, the same goal-conditioned policy can perform a variety of tasks when it is commanded with different goals. Such flexibility allows GCRL to better share knowledge across different tasks and make use of goal relabeling techniques to improve the sample efficiency without meticulous reward engineering **Andrychowicz2017HindsightER**; **Pong2020SkewFitSS**; **Fang2019CurriculumguidedHE**; **Ding2019GoalconditionedIL**; **Gupta2019Relay** **Sun2019PolicyCW**; **Eysenbach2020RewritingHW**; **Ghosh2021LearningTR**. Prior has explored various strategies for proposing goals for exploration **nair2018rig**; **Nair2019ContextualIG**; **Khazatsky2021WhatCI**; **ChaneSane2021GoalConditionedRL**, and studied goal-conditioned RL from offline data **chebotar2021actionable**. However, such works generally aim to learn short-horizon behaviors, and learning to reach goals that require multiple stages (e.g., several manipulation primitives) is very difficult, as shown in our experiments. Our work aims to extend model-free goal-conditioned RL methods by incorporating elements of planning to enable effective finetuning for multi-stage tasks.

Planning. A wide range of methods have been developed for planning in robotics. At the most abstract level, symbolic task planning searches over discrete logical formulas to accomplish abstract goals **fikes1971strips**. Motion planning methods solve the geometric problem of reaching a goal configuration with dynamics and collision constraints **Kavraki1996**; **koenig2002dstarlite**; **karaman2011rrtstar**; **zucker2013chomp**; **kalakrishnan2011stomp**. Prior methods have also considered task and motion planning as a combined problem **srivastava14tamp**. These methods generally assume high-level structured representations of environments and tasks, which can be difficult to actualize in real-world environments. Since in our setting we only have image inputs and not structured scene representations, we focus on methods that can handle raw images for observations and task specification.

Combining goal-conditioned RL and planning. A number of recent works have sought to integrate concepts from planning with goal-conditioned policies in order to plan sequences of subgoals for longer-horizon tasks **Nasiriany2019PlanningWG**; **Eysenbach2019Searc** **fang2019cavin**; **Charlesworth2020PlanGANMP**; **Pertsch2020LongHorizonVP**; **Sharma2021Autonomo** **Zhang2021CPlanningAA**. These prior methods either propose subgoals from the set of previously seen states, or directly optimize over subgoals, often by utilizing a latent variable model to obtain a concise representation of image-based states **nair2018rig**; **ichter2018learning**; **nair2019hierarchical**; **Nasiriany2019PlanningWG**; **Pertsch2020LongHorizonVP**; **Khazatsky2021WhatCI**; **ChaneSane2021GoalConditionedRL**. The method we employ is most closely related conceptually to the method proposed by Pertsch et al. **Pertsch2020LongHorizonVI** which also employs a hierarchical subgoal optimization, and the method proposed by Nasiriany et al. **Nasiriany2019PlanningWG**, which also optimizes over sequences of latent vectors from a generative model. Our approach makes a number of low-level improvements, including the use of a conditional generative model **Nair2019ContextualIG**, which we show leads to significantly better performance. More importantly, our method differs conceptually from these prior works in that our focus is specifically on utilizing subgoal optimization as a way to enable finetuning goal-conditioned policies for longer-horizon tasks. We show that it is in fact this capacity to enable effective finetuning that enables our method to solve more complex multi-stage tasks in our experiments.

10.3 PROBLEM STATEMENT

In this paper, we consider the problem of learning to complete a long-horizon task specified by a goal image. The robot learns over a variety of initial configurations and goal distributions, which cover a range of behaviors such as opening or closing a drawer, and picking, placing, or pushing an object. As prior data, the robot has access to an offline dataset of trajectories $\mathcal{D}_{\text{offline}} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ for offline pre-training. In each trajectory, the robot is controlled by a human tele-operator or a scripted policy to achieve one of the goals the environment affords. A goal-conditioned policy is pre-trained on this dataset using offline RL algorithms.

After offline pre-training, the robot is placed in a particular environment that it has online access to interact in. Even though the initial configuration of this environment may have been included in the set of training environments, the goal distribution for this environment at test time requires sequencing multiple skills together, which is not present in the offline data. For instance, as shown in Fig. 49, the robot would need to first slide away the can that blocks the drawer, then reaches the handle of the drawer,

and finally opens the drawer.

Naïvely running offline RL may not solve the long-horizon test tasks for two reasons. First, the robot is given a test goal distribution that is long horizon but offline dataset consists of individual skills. The method needs to somehow compose these individual skills autonomously in order to succeed at goals drawn from the test distribution. Second, offline RL may not solve the task due to distribution shift. Distribution shift appears in two forms: distribution shift between transitions in the prior data and transitions obtained by the actively rolling out the policy, and the distribution shift introduced when performing tasks sequentially. If the robot may actively interact in the new environment to improve its policy, how can the robot further practice and improve its performance?

10.4 PRELIMINARIES

We consider a goal-conditioned Markov Decision Process (MDP) denoted by a tuple $M = (\mathcal{S}, \mathcal{A}, \rho, P, \mathcal{G}, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, initial state probability ρ, transition probability P, a goal space $\mathcal{G}$, and discount factor γ. In each episode, a desired goal $s_g \in \mathcal{G}$ is sampled for the robot to reach. At each time step t, a goal-conditioned policy $\pi(a_t|s_t, s_g)$ selects an action $a_t \in \mathcal{A}$ conditioned on the current state s_t and goal s_g. After each step, the robot receives the goal-reaching reward $r_t(s_{t+1}, s_g)$. The robot aims to reach the goal by maximizing the average cumulative reward $\mathbb{E}[\Sigma_t \gamma^t r_t]$. Our approach learns a goal-conditioned policy π for solving the target task specified by a desired final goal s_g. The goal-conditioned policy is pre-trained on a previously collected offline dataset $\mathcal{D}_{\text{offline}}$ and then fine-tuned to reach s_g by accumulating data into an online replay buffer $\mathcal{D}_{\text{online}}$. $\mathcal{D}_{\text{offline}}$ contains diverse short-horizon interactions with objects in the environment. During online fine-tuning, we would like the policy to learn to improve and compose these short-horizon behavior for multi-stage tasks specified by s_g.

Defining informative goal-reaching rewards and extracting useful state representations from high-dimensional raw observations such as images can be challenging. Following the practice in prior work **nair2018rig**; **Khazatsky2021WhatCI**, we pre-train a state encoder $h = \phi(s)$ to extract the latent state representation h. By encoding the states and goals to the latent space, we can obtain an informative goal-reaching reward function $r_t = R(h_{t+1}, h_g)$ by computing $h_{t+1} = \phi(s_{t+1})$ and $h_g = \phi(s_g)$. Specifically, $R(h_{t+1}, h_g)$ returns 0 when $\|h_{t+1}, h_g\| < \epsilon$ and -1 otherwise, where ϵ is a selected threshold. In addition, we also use $\phi(s_{t+1})$ as the backbone feature extractor in all of our models that take s as an input. For simplicity, we directly use s to denote h in the rest of the paper. The details of the state encoder are explained in Sec. 10.6.2.

We propose Planning to Practice (PTP), an approach that efficiently fine-tunes a goal-conditioned policy to solve novel tasks. To enable the robot to efficiently learn to solve the target task, we propose to use subgoals to facilitate the online fine-tuning of the goal-conditioned policy. Given the initial state s_0 and the goal state s_g, we search for a sequence of K subgoals $\hat{s}_1 : K = \hat{s}_1, ..., \hat{s}_K$ to guide the robot to reach s_g. Such subgoals will inform the goal-conditioned policy $\pi(a|s, s_g)$ what is the immediate next step on the path to s_g and provide the policy more dense reward signals compared to directly using the final goal. We choose the sequence of subgoals at the beginning of each episode and feed the first subgoal in the sequence to the goal-conditioned policy. The policy will switch to the next subgoal in the sequence when the current subgoal is reached or the time budget assigned for the current subgoal runs out.

The main challenge is to search for a sequence of subgoals that can lead to the desired final goals while ensuring each subgoal is a valid state that can be reached from the previous subgoal. Particularly when the states correspond to full images, most vectors will not actually represent valid states, and indeed naïvely optimizing over image pixels may simply result in out-of-distribution inputs that lead to erroneous results when input into the goal-conditioned policy.

As outlined in Fig. 49, we devise a method to effectively propose and select valid subgoal sequences to guide online fine-tuning by means of a generative model. At the heart of our approach is a conditional subgoal generator $g(\cdot|s_0)$ that recursively produces candidate subgoals in a hierarchical manner conditioned on the initial state s_0. To find the optimal sequence of subgoals $\hat{s}^*_{1:K}$, we first sample N candidate sequences $\hat{s}^1_{1:K}, ..., \hat{s}^N_{1:K}$ from the state space using the conditional subgoal generator. Then we rank the candidate sequences using a cost function $c(s_0, \hat{s}_{1:K}, s_g)$. The sequence that corresponds to the lowest cost will be selected as $\hat{s}^*_{1:K}$ for the goal-conditioned policy. Through this sampling-based planning procedure, we choose the subgoal for guiding the goal-conditioned policy π during online fine-tuning. The overall algorithm is summarized in Algorithm 8. Next we describe the design of each module in details.

10.5.1 *Conditional Subgoal Generation*

The effectiveness of our planner relies on the generation of diverse and feasible sequences of subgoals as candidates. Specifically, we would like to generate the candidates by sampling from the distribution of suitable subgoal sequences $p(\hat{s}_1, ..., \hat{s}_K|s_0)$ condi-

Algorithm 8 Planning To Practice (PTP)

Require: set of final goals $\mathcal{G}$, time horizon T, offline data $\mathcal{D}_{\text{offline}}$, number of subgoals K.

Train $\pi(a|s, s_g)$ and $g(s, z)$ on $\mathcal{D}_{\text{offline}}$.
Initialize the online replay buffer $\mathcal{D}_{\text{online}} \leftarrow \varnothing$.
while not converged **do**
  Reset the environment and observe s_0.
  Sample s_g from $\mathcal{G}$.
  Plan for the subgoals $\hat{s}_{1:K}$.
  $k \leftarrow 1$
  for $t = 1, ..., T$ **do**
    Compute the action $a_t \leftarrow \pi(a_t|s_t, \hat{s}_K)$
    Observe the state s_{t+1} and the reward r_t
    $\mathcal{D}_{\text{online}} \leftarrow \mathcal{D}_{\text{online}} \cup (s_t, a_t, r_t, s_{t+1})$.
    if $t \pmod{\Delta t} == 0$ **or** $\|s_{t+1} - \hat{s}_K\| < \epsilon$ **then**
      $k \leftarrow \min(k + 1, K)$
  Train π on batches sampled from $\mathcal{D}_{\text{offline}}$ and $\mathcal{D}_{\text{online}}$.

=o

tioned on the initial state s_0. Most existing methods independently sample the subgoal at each step from a learned prior distribution **Pertsch2020LongHorizonVP** or a replay buffer **Eysenbach2019SearchOT**, which is unlikely to propose useful plans for tasks with large, combinatorial state spaces (i.e., with multiple objects).

We propose to break down $p(\hat{s}_1, ..., \hat{s}_K|s_0)$ into $p(\hat{s}_1|s_0)\Pi_{i=1}^{k} p(\hat{s}_i|\hat{s}_{i-1})$ through modeling the conditional distribution $p(s'|s)$ of the reachable next subgoal s'. By utilizing temporal compositionality, the conditional subgoal generation paradigm improves generalization and enables generation of sequences of arbitrary lengths.

We use a conditional variational encoder (CVAE) **sohn2015cvae** to capture the distribution of reachable goals $p(s'|s)$. In the CVAE, we define the decoder as $g(s, z)$ and the encoder as $q(z|s, s')$, where z is the learned latent representation of the transitions and it is sampled from a prior probability $p(z)$. To propose a sequence of subgoals, we use $g(s, z)$ as the conditional subgoal generator. Conditioned on the initial state s_0, the first subgoal $\hat{s}_1$ can be generated as $\hat{s}_1 = g(s_0, z_1)$ given the sampled z_1. Then the i_{th} subgoal can be recursively generated by sampling $z_i \sim p(z)$ and computing $\hat{s}_i = g(\hat{s}_{i-1}, z_i)$ given the previous subgoal $\hat{s}_{i-1}$. In this way, we could sample a sequence of i.i.d. latent representations $z_1, ..., z_K$ and recursively generate $\hat{s}_1, ..., \hat{s}_K$ conditioned on the initial state s_0 using the conditional subgoal generator.

The CVAE is trained to minimize the evidence lower bound (ELBO) Kingma and

Algorithm 9 $\texttt{Plan}(s_0, s_g, L, K, M, N)$

Require: the initial state s_0, the goal state s_g, number of subgoals K, number of levels L, multiplier M, number of samples N.

Sample N latent action sequences $\{z^i_{1:K}\}^N_{i=1}$.
Recursively generate subgoals $\{\hat{s}^i_{1:K}\}^N_{i=1}$ using $g(s, z)$.
Select $z^*_{1:K}$ and $\hat{s}^*_{1:K}$ of the lowest cost.
Update $z^*_{1:K}$ and $\hat{s}^*_{1:K}$ using MPPI.
if $L = 1$ **then**
  return $\hat{s}^*_{1:K}$
else
  Denote $\hat{s}^*_0 \leftarrow s_0$.
  Initialize the plan $\hat{\mathcal{S}}$ as an empty list
  for $i = 1, ..., K$ **do**
    Append $\texttt{Plan}(\hat{s}^*_{i-1}, \hat{s}^*_i, L-1, M, M, N)$ to $\mathcal{S}$
  return $\hat{\mathcal{S}}$

=0

Welling, 2014 of $p(s'|s)$ given the offline dataset $\mathcal{D}$. During training, we sample transitions (s_t, s_τ) from the offline dataset to form the minibatches, where $\tau = t + \Delta t$ is a future step that is Δt steps ahead. Instead of using a fixed Δt, we sample Δt from a range for each transition to provide richer data. To encourage the trained model to be robust to compounding errors, we sample sequences composed of multiple states and use the subgoal reconstructed at the previous step as the context in the next step. Therefore, the objective for training the conditional subgoal generator is:

$$\mathbb{E}_{q(z|s_t, s_\tau)} \|s_\tau - g(s_t, z)\|^2 + D_{KL}[q(z|s_t, s_\tau) \| p(z)] \tag{50}$$

where $D_{KL}[\cdot\|\cdot]$ indicates the KL-Divergence.

10.5.2 *Efficient Planning in the Latent Space*

We build a planner that efficiently searches for sequences of subgoals in the latent space as shown in Algorithm 9. To tackle the large search space of candidate subgoal sequences, we design a hierarchical planning algorithm that searches for subgoals in a coarse-to-fine manner and re-use the previously selected subgoals as candidates in new episodes.

The hierarchical planning is conducted at L levels with different temporal resolutions

Δt_1, ..., Δt_L. The temporal resolution of each level is an integral multiple of that of the previous level, i. e., $\Delta t_i = M \Delta t_{i-1}$, where M is a scaling factor and is set to 2 in our experiments. We first plan for the subgoals $\hat{s}_1^l, \hat{s}_2^l, ...$ on the first level. Then the subgoals $\hat{s}_{1:K}^l$ of finer temporal resolution are planned on each level l to connect the subgoals planned on the previous level $l-1$. Specifically, given the adjacent subgoals $\hat{s}_i^{l-1}$ and $\hat{s}_{i+1}^{l-1}$ produced on the previous level, we plan for a segment of M subgoals $\hat{s}_{i*M+1}^l, ..., \hat{s}_{(i+1)*M}^l$ on the level l, by treating $\hat{s}_i^{l-1}$ and $\hat{s}_{i+1}^{l-1}$ as the initial state and final goal state in Eqn. 52. The planned segments are returned to the previous level and concatenated as a more fine-grained plan. For this purpose, we train L conditional subgoal generators to propose subgoals that are Δt_1, ..., Δt_L steps away, respectively. In contrast to the prior work **Pertsch2020LongHorizonVP**, the conditional subgoal generators enable us to plan for unseen goals that are beyond the temporal horizon of the demonstrations in the offline dataset by exploiting the compositional structure of the demonstrations. By recursively generating the subgoals across time at each level, we only need to enforce that the temporal resolution of the top level Δt_L is smaller than since the the conditional subgoal generator $f^1(s, z)$ needs to be trained on trajectories at least $\Delta t_L + 1$ steps in length.

We maintain a latent plan buffer for each level to further facilitate the planning with the conditional subgoal generator. After each episode, the selected latent representations on each level are appended to the corresponding latent plan buffer. In each target task, the subgoals are supposed to have the same semantic meaning. In spite of the variations of the initial and goals state in each episode, the optimal plans in the latent space can often be similar to each other. Therefore, we sample half of the latent representations from the prior distribution $p(z)$ and the other half from the latent plan buffer among the initial samples to enhance the chance of finding a close initial guess.

We build our planner upon the model predictive path integral (MPPI) **Gandhi2021RobustMP**, which iteratively optimizes the plan through importance sampling. In each interaction, we perturb the chosen plan in the latent space with a small Gaussian noise as new candidates.

10.5.3 *Cost Function For Feasible Subgoals*

To provide informative guidance to the policy $\pi(a|s, s_g)$, we would like that the final goal s_g can be reached at the end of the episode while encouraging the transition between each pair of subgoals to be feasible within a limited time budget. As explained in Sec. 10.4, the goal state is considered to be reached when the Euclidean distance between the last subgoal in the plan and the desired goal is less than a threshold δ in

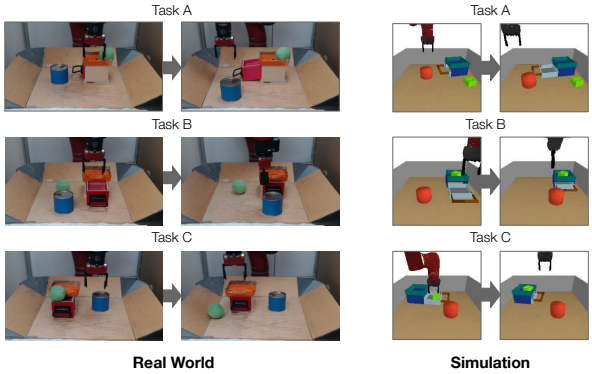


Figure 50: **Target tasks.** Three multi-stage tasks are designed for our experiments in the simulation and the real world respectively. In each target task, the robot needs to strategically interacts with the environment (e. g.first takes out an object in the drawer then closes

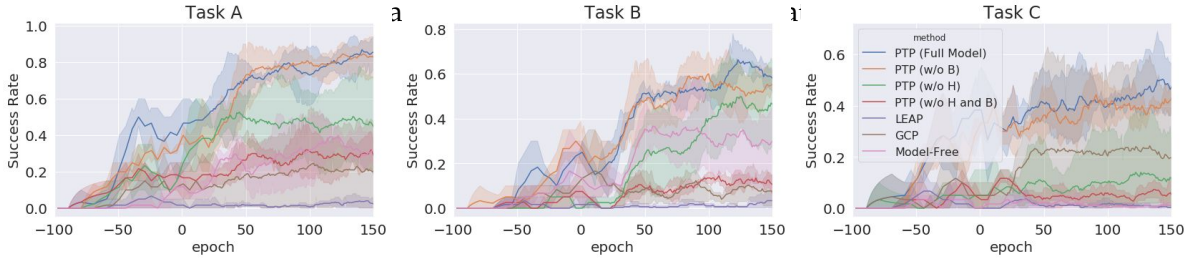


Figure 51: **Quantitative comparison in simulation.** The average success rate across 3 runs is shown with the shaded region indicating the standard deviation. The negative x-axis indicates the epochs of offline pre-training and positive x-axis indicates epochs of online fine-tuning. Using offline learning and planning, our method PTP is able to solve these tasks partially (at 0 epochs). Then with online finetuning the performance improves further. In contrast, prior methods have lower offline performance and do not fine-tune successfully in most cases, as they do not collect coherent online data.

the learned latent space. The feasibility of each transition between adjacent subgoals can be measured using the goal-conditioned value function $V(s, s')$ trained by the reinforcement learning algorithm. Therefore, finding the subgoals $\hat{s}^*_{1:K}$ can be formulated as a constrained optimization problem:

$$\begin{aligned} &\text{minimize} && \|s_g - \hat{s}_K\| \\ &\text{subject to} && V(\hat{s}_i, \hat{s}_{i+1}) \geqslant \delta, \text{for } i = 0, ..., K-1 \end{aligned} \tag{51}$$

where we use $\hat{s}_0 = s_0$ to denote the initial state for convenience. By re-writing Eq. 51 as a Lagrangian, we obtain the cost function with a weight η:

$$c(s_0, \hat{s}_{1:K}, s_g) = \|s_g - \hat{s}_K\| + \eta \sum_{i=0}^{K-1} V(\hat{s}_i, \hat{s}_{i+1}) \tag{52}$$

The details of our method are explained in Sec. 10.6.2.

10.6 EXPERIMENTS

In our experiments, we aim to answer the following questions: 1) Can PTP propose and select feasible subgoals as plans for real-world robotic manipulation tasks? 2) Can the subgoals planned by PTP facilitate online fine-tuning of the goal-conditioned policies to solve target tasks unseen in the offline dataset? 3) How does each design option affect the performance of PTP? Videos of our experimental results are available on the project website: sites.google.com/view/planning-to-practice

10.6.1 *Experimental Setup*

Environment. As shown in Fig. 50, our experiments are conducted in a table-top manipulation environment with a Sawyer robot. At the beginning of each episode, a fixed drawer and two movable objects are randomly placed on the table. The robot can change the state of the environment by opening/closing the drawer, sliding the objects, and picking and placing objects into different destinations, . At each time step, the robot receives a 48 x 48 RGB image via a Logitech C920 camera as the observation and takes a 5-dimensional continuous action to change the gripper status through position control. The action dictates the change of the coordinates along the three axes, the change of the rotation, and the status of the fingers. We use PyBullet **coumans2021** for our simulated experiments.

Prior data. The prior data consists of varied demonstrations for different primitive tasks. In each demonstrated trajectory, we randomly initialize the environment and perform primitive interactions such as opening the drawer and poking the object. These trajectories are collected using teleoperation in the real world, and a scripted policy that uses privileged information of the environment (e.g., the object pose and the status of the drawer) in simulation. The trajectories vary in length from 5 to 150 time steps, with 2,344 trajectories in the real world and 4,000 in simulation.

Target tasks. In each target task, a desired goal state is specified by a 48 x 48 RGB image (same dimension with the observation). The robot is tasked to reach the goal state by interacting with the objects on the table. Task success for our evaluation is determined based on the object positions at the end of each episode (this metric is not used for learning). As shown in Fig. 50, we design three target tasks that require multi-stage interactions with the environment to complete. These target tasks are designed with temporal dependencies between stages (e. g.the robot needs to first move away a can that blocks the drawer before opening the drawer). The transitions from the initial state to the goal state are unseen in the offline data. The episode length is 400 steps in simulation and 125 steps in the real world, which are much longer than the time horizon of the demonstrations.

Baselines and ablations. We compare PTP with 3 baselines and 3 ablations. **Model-Free** uses a policy directly conditioned on the final goal and conducts online fine-tuning without using any subgoals. **LEAP Nasiriany2019PlanningWG** learns a variational auto-encoder (VAE) Kingma and Welling, 2014 to capture the prior distribution of states and plans for subgoals without conditioning on any context. **GCP Pertsch2020LongHorizonVP** learns a goal predictor that hierarchically generates intermediate subgoals between the initial state and the goal state. To analyze the design options in PTP, we also compare with variations of our method by removing the latent plan buffer (**PTP (w/o B)**), the hierarchical planning algorithm (**PTP (w/o H)**), and both of these two designs (**PTP (w/o H and B)**). All methods use the same neural network architecture in the goal-conditioned policy and are pre-trained on the same offline dataset.

10.6.2 *Implementation Details*

Following **Khazatsky2021WhatCI**, we use a vector quantized variational autoencoder (VQ-VAE) **Oord2017NeuralDR** as the state encoder, which encodes a $48 \times 48 \times 3$ image to a 720-dimensional encoding. The conditional subgoal generator is implemented with a U-Net architecture **Ronneberger2015UNetCN** and decodes the subgoal from a 8-dimensional latent representations conditioned on the encoding of the current state. In our planner, we use $L = 3$, $K = 8$, $M = 2$, $N = 1024$, and we run MPPI for 5 iteration on each level. g is trained to predict subgoals that are 15, 30, and 60 steps away. Implicit Q-Learning (IQL) **kostrikov2021iql** is used as the underlying RL algorithm for offline pre-training and online fine-tuning with default hyperparameters. We use the same network architectures for the policy and the value functions from **Khazatsky2021WhatCI** for simulation experiments. For real-world experiments, we use a convolutional neural

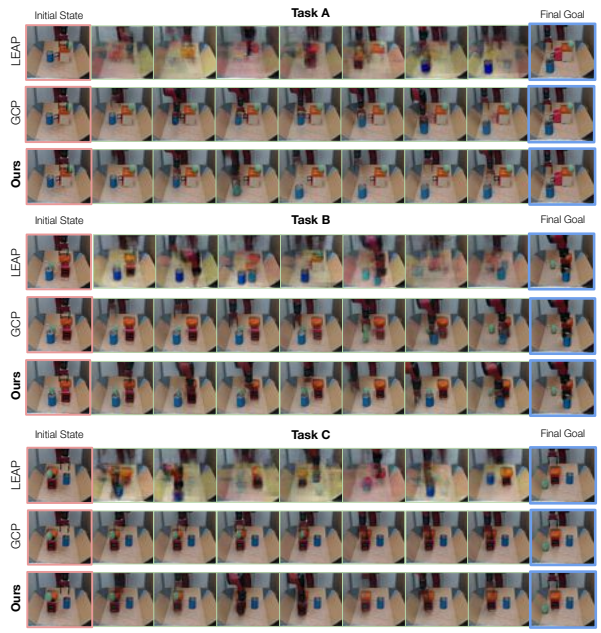


Figure 52: **Planned subgoal sequences.** Each row shows the sequence of subgoals produced by each method. The initial state and the final goal are shown at the two ends.

network instead. We use Adam optimizer with a learning rate of $3 \cdot 10^{-4}$ and a batch size of 1024. During training, we relabel the goal with future hindsight experience replay **Andrychowicz2017HindsightER** with 70% probability. We use $\epsilon = 3$ for the reward function defined in Sec. 10.4, $\eta = 0.01$ in Eqn. 52.

10.6.3 *Quantitative Comparisons*

We evaluate PTP and baselines on three unseen target tasks. We use simulated versions of these tasks for comparisons and ablations, and real-world tasks, where all pretraining and finetuning uses only real-world data, to evaluate the practical effectiveness of the method.

Simulation. We first pre-train the goal-conditioned policy on the offline dataset for 100 epochs and the run online fine-tuning for the target task for 150 epochs. Each epoch takes 2,000 simulation steps (only during fine-tuning) and 2,000 training iterations. We run online fine-tuning using each method with 3 different random seeds. After each epoch, we test the policy in the target task for 5 episodes. We report the average success rate across 3 runs in Fig. 51 where the negative x-axis indicates the offline pre-training epochs and positive x-axis indicates the online fine-tuning epochs.

As shown in Fig. 51, our full model consistently outperforms baselines with a large

performance gap. The generated subgoals not only enables the pre-trained policy to achieve higher success rate by breaking down the hard problems into easier pieces, but also introduces larger performance improvements during online fine-tuning. After fine-tuning for 150 epochs, the policy achieves the success rates of 84.9%, 59.9%, 49.3% in the three target tasks respectively. Compared to the policy pre-trained on the offline dataset, the performance is significantly improved (+31.6%, +37.8%, and +13.8%). When directly using the final goal or subgoals generated by baseline methods, the policy's performance plateaus at around 0.0% to 30.0% and does not improve much during online fine-tuning.

We found that the hierarchical planner and the latent plan buffer are crucial for PTP's performance. Without these two design options, the planner often suffers from the large search space of possible subgoal sequences and the resultant success rates decrease. The latent plan buffer significantly improves the performance of non-hierarchical PTP while it has a minor effect on hierarchical PTP.

Real-world evaluation. We pre-train the policy for 200 epochs and fine-tune it for 10 epochs. In each epoch, we run 10,000 training iterations and collect 1,000 steps in the real world. We train on three target tasks which are shown in Figure 50, and report the success rate of the goal-conditioned policy before and after online fine-tuning in Table 3. Planning enables the robot to succeed partially with just the offline initialized policy, achieving success rates of 12.5%, 75.0% and 25.0% on the three tasks. (When the offline policy is conditioned on only the final goal image without planning, the success rate is 0%.) Then in each task, we fine-tune to a significantly higher success rate.

Qualitatively, at the beginning of fine-tuning, the robot often fails, deviating from the planned subgoals or colliding with the environment. With the planned subgoals, the original long-horizon task is broken down to short snippets that are easier to complete. Even if a subgoal is not reached successfully at first, the data is useful to collect additional experience and fine-tune the policy. After fine-tuning for 4-5 epochs, we already observe that the robot's performance reaching subgoals during training time significantly improves, collecting even more coherent and useful data. After 10 epochs, we achieve success rates of 62.5%, 100.0% and 50.0%. In comparison, GCP cannot provide useful guidance to the policy when the generated goals are noisy.

10.6.4 *Generated Subgoals*

In Fig. 52, we present qualitative results of the generated subgoals for each task in the real world. Each row shows a sequence of generated subgoals produced by the planner in each method. In all the three target tasks, PTP successfully plans for a sequence of

Table 3: The real-world success rates before and after online fine-tuning. The tasks are described in Sec. 10.6.1.

Task	PTP (Ours)	GCP
	Offline → Online	Offline → Online
Task A	12.5% → **62.5%**	12.5% → 0.0%
Task B	75.0% → **100.0%**	50.0% → 75.0%
Task C	25.0% → **50.0%**	25.0% → 12.5%

subgoals that can lead to the desired final goal. The transition between adjacent subgoals are feasible within a short period of time. By comparison, both of the baseline methods fail to generate reasonable plans. Without conditioning on the current state, LEAP **Nasiriany2019PlanningWG** can hardly produce any realistic images of the environment. Most of the generated subgoals are highly noisy images with duplicated robot arms and objects. The quality of the subgoals produced by GCP **Pertsch2020LongHorizonVP** is higher than that of LEAP but still much worse than ours. GCP cannot generalize well for the initial state and the goal state that are out of the distribution of the offline dataset, which contains only short snippets of demonstrations.

10.7 CONCLUSION AND DISCUSSION

We presented PTP, a method for real-world learning of temporally extended skills by utilizing planning and fine-tuning to stitch together skills from prior data. First, planning is used to convert a long-horizon task into achievable subgoals for a lower level goal-conditioned policy trained from prior data. Then, the goal-conditioned policy is further fine-tuned with active online interaction, mitigating the distribution shift between the offline data and actual states seen during rollouts. This procedure allows robots to extend their capabilities autonomously, composing previously seen data into more complicated and useful skills.

bibtex/references