Offline Meta-Reinforcement Learning with Online Self-Supervision

Vitchyr H. Pong, Ashvin Nair, Laura Smith, Catherine Huang, Sergey Levine
UC Berkeley
{vitchyr,anair17,smithlaura,thecatherinehuang,svlevine}@berkeley.edu

Abstract

Meta-reinforcement learning (RL) methods can meta-train policies that adapt to new tasks with orders of magnitude less data than standard RL, but meta-training itself is costly and time-consuming. If we can meta-train on offline data, then we can reuse the same static dataset, labeled once with rewards for different tasks, to meta-train policies that adapt to a variety of new tasks at meta-test time. Although this capability would make meta-RL a practical tool for real-world use, offline meta-RL presents additional challenges beyond online meta-RL or standard offline RL settings. Meta-RL learns an exploration strategy that collects data for adapting, and also meta-trains a policy that quickly adapts to data from a new task. Since this policy was meta-trained on a fixed, offline dataset, it might behave unpredictably when adapting to data collected by the learned exploration strategy, which differs systematically from the offline data and thus induces distributional shift. We do not want to remove this distributional shift by simply adopting a conservative exploration strategy, because learning an exploration strategy enables an agent to collect better data for faster adaptation. Instead, we propose a hybrid offline meta-RL algorithm, which uses offline data with rewards to meta-train an adaptive policy, and then collects additional unsupervised online data, without any reward labels, to bridge this distribution shift. By not requiring reward labels for online collection, this data can be much cheaper to collect. We compare our method to prior work on offline meta-RL on simulated robot locomotion and manipulation tasks and find that using additional unsupervised online data collection leads to a dramatic improvement in the adaptive capabilities of the meta-trained policies, matching the performance of fully online meta-RL on a range of challenging domains that require generalization to new tasks.

1 Introduction

Reinforcement learning (RL) agents are often described as learning from reward and punishment analogously to animals: in the same way that a person might train a dog by providing treats, we might train RL agents by providing rewards. However, in reality, modern deep RL agents require so many trials to learn a task that providing rewards by hand is often impractical. Meta-reinforcement learning in principle can mitigate this, by learning to learn using a set of meta-training tasks, and then acquiring new behaviors in just a few trials at meta-test time. Current meta-RL methods are so efficient that it is entirely practical for the meta-test time adaptation to use even human-provided rewards. However, the meta-training phase in these algorithms still requires a large number of online samples, often even more than standard RL, due to the multi-task nature of the meta-learning problem.

Offline reinforcement learning methods, which use only prior experience without active data collection, provide a potential solution to this issue, because a user must only annotate multi-task data with rewards once in the offline dataset, rather than doing so in the inner loop of RL training, and the same

5th Workshop on Meta-Learning at NeurIPS 2021, held virtually.

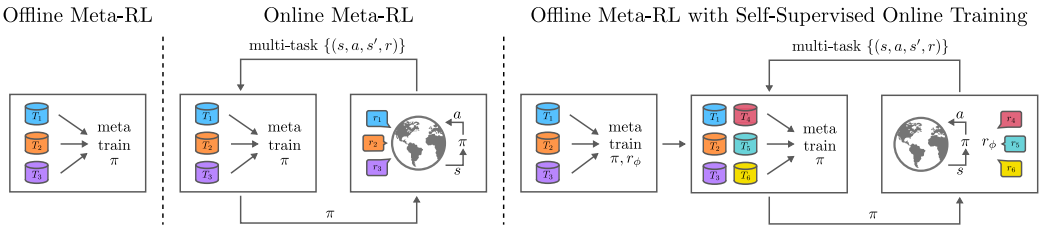

Figure 1: (left) In offline meta-RL, an agent uses offline data from multiple tasks $T_1, T_2, \ldots$, each with reward labels that must only be provided once. (middle) In online meta-RL, new reward supervision must be provided with every environment interaction. (right) In semi-supervised meta-RL, an agent uses an offline dataset collected once to learn to generate its own reward labels for new, online interactions. Similar to offline meta-RL, reward labels must only be provided once for the offline training, and unlike online meta-RL, the additional environment interactions do not require external reward supervision.

offline multi-task data can be reused repeatedly for many training runs. While a few recent works have proposed offline meta-RL algorithms [8, 36], we identify a specific problem when an agent trained with offline meta-RL is tested on a new task: the distributional shift between the behavior policy and the meta-test time exploration policy means that adaptation procedures learned from offline data might not perform well on the (differently distributed) data collected by the exploration policy at meta-test time. In practice, we find that this leads to a large degradation in performance when adapting to new tasks. This mismatch in training distribution occurs because offline meta-RL never trains on data generated by the meta-learned exploration policy.

We propose to address this challenge by collecting additional online data *without* any reward supervision, leading to a semi-supervised offline meta-RL algorithm, as illustrated in Figure 1. Online data can be relatively cheap to collect when it does not require reward labels, but it can still make it possible to bridge the distributional shift issue. To make it feasible to use this data for meta-training, we can generate synthetic reward labels for it based on the labeled offline data.

Based on this principle, we propose semi-supervised meta actor-critic (SMAC), which uses reward-labeled offline data to bootstrap a semi-supervised meta-reinforcement learning procedure, in which an offline meta-RL agent collects additional online experience without any reward labels. The agent uses the reward supervision from the offline dataset to learn to generate new reward functions, which it uses to autonomously annotate rewards in these otherwise rewardless interactions and meta-train on this new data. We evaluate our method and prior offline meta-RL methods on a number of benchmarks [8, 36], as well as a challenging robotic manipulation domain that requires generalization to new tasks, with fewer than 400 time steps of reward labels at meta-test time. We find that, while standard meta-RL methods perform well at adapting to training tasks, they suffer from data-distribution shifts when adapting to new tasks that were not seen during meta-training. In contrast, our method attains significantly better performance, on par with an online meta-RL method that receives fully labeled online interaction data.

2 Related Works

Many prior meta-RL algorithms assume that reward labels are provided with each episode of online interaction [9, 12, 18, 53, 20, 43, 23, 30, 56, 54, 55, 27]. In contrast to these prior methods, our method only requires offline prior data with rewards, and additional online interaction does not require any ground truth reward signal. Prior works have also studied other formulations that combine unlabeled and labeled trials. For example, imitation and inverse reinforcement learning methods use offline demonstrations to either learn a reward function [1, 11, 22, 13] or to directly learn a policy [46, 45, 22, 44, 40]. Semi-supervised and positive-unlabeled reward learning [52, 57, 31] methods use reward labels provided for some interactions to train a reward function for RL. However, all of these methods have been studied in the context of a single task. In contrast, we focus on meta-learning an RL procedure that can adapt to new reward functions. In other words, we do not focus on recovering a single reward function, because there is no single test time reward or task.

SMAC uses a context-based adaptation procedure similar to that proposed by Rakelly et al. [43], which is related to contextual policies, such as goal-conditioned reinforcement learning [26, 47, 2, 42, 6, 50, 41, 38] or successor features [32, 3, 4, 16]. In contrast, our meta-learning procedure applies

to any RL problem, does not assume that the reward is defined by a single goal state or fixed basis function, and only requires reward labels for static offline data.

Our method addresses a similar problem to prior offline meta-RL methods [36, 8], but we show that these approaches perform poorly due to distributional shift. Our method addresses the distribution shift problem by using online interactions without reward supervision. In our experiments, we found that SMAC greatly improves performance on both training and held-out tasks. Lastly, SMAC is also related to unsupervised meta-learning methods [17, 25], which annotate data with their own rewards. In contrast to these methods, we assume that there exists an offline dataset with reward labels that we can use to learn to generate similar rewards.

3 Preliminaries

Meta-reinforcement learning. In meta-RL, we assume there is a distribution of tasks $p_{\mathcal{T}}(\cdot)$. A task $\mathcal{T}$ is a Markov decision process (MDP), defined by a tuple $\mathcal{T} = (\mathcal{S}, \mathcal{A}, r, \gamma, p_0, p_d)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, r is a reward function, γ is a discount factor, $p_0(\mathbf{s}_0)$ is the initial state distribution, and $p_d(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the environment dynamics distribution. A replay buffer $\mathcal{D}$ is a set of state, action, reward, next-states tuples, $\mathcal{D} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^{N_{\text{size}}}$, where all the rewards come from the same task. We will use the letter $\mathbf{h}$ to denote a small replay buffer or "history" and the notation $\mathbf{h} \sim \mathcal{D}$ to denote that a mini-batch $\mathbf{h}$ is sampled from a replay buffer $\mathcal{D}$. We will use the letter τ to represent a trajectory $\tau = (\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \dots)$ without reward labels.

A meta-episode consists of sampling a task $\mathcal{T} \sim p_{\mathcal{T}}(\cdot)$, collecting T trajectories with a policy π_θ, adapting the policy to the task between each trajectory, and measuring the performance on the last trajectory. We write the policy's adaptation procedure as A_ϕ, parameterized by meta-parameters ϕ. Between each trajectory, the adaptation procedure transforms the history of interactions $\mathbf{h}$ within the meta-episode into a context $\mathbf{z} = A_\phi(\mathbf{h})$ that summarizes the previous interactions. This context $\mathbf{z} \in \mathcal{Z}$ is then given to the policy $\pi_\theta(\mathbf{a}, \mid \mathbf{s}, \mathbf{z})$. The exact representation of π_θ, A_ϕ, and $\mathcal{Z}$ depends on the specific meta-RL method used. For example, the context $\mathbf{z}$ can be weights of a neural network [12] outputted by a gradient update, hidden activations ouputted by a recurrent neural network [9], or latent variables outputted by a stochastic encoder [43]. Using this notation, the objective in meta-RL is to learn the adaptation parameters ϕ and policy parameters θ to maximize performance on a meta-episode given a new task $\mathcal{T}$ sampled from $p(\mathcal{T})$.

PEARL. Since we require an off-policy meta-RL procedure for offline meta-training, we build on probabilistic embeddings for actor-critic RL (PEARL) [43], an online off-policy meta-RL algorithm. In PEARL, $\mathbf{z}$ is a vector and the adaptation procedure A_ϕ that maps $\mathbf{h}$ to $\mathbf{z}$ consists of sampling $\mathbf{z}$ from a distribution $\mathbf{z} \sim q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$. The distribution q_{ϕ_e} is generated by an encoder network with parameters ϕ. This encoder is a set-based network that processes all of the tuples in $\mathbf{h} = \{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^{N_{\text{enc}}}$ in a permutation-invariant manner to produce the mean and variance of a diagonal multivariate Gaussian. The policy is a contextual policy $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ conditioned on $\mathbf{z}$ by concatenating $\mathbf{z}$ to the state $\mathbf{s}$.

The policy parameter θ is trained using soft-actor critic [19] which involves learning a critic, or Q-function, $Q_w(\mathbf{s}, \mathbf{a}, \mathbf{z})$, with parameter w that estimates the sum of future discounted rewards conditioned on the current state, action, and context. The encoder parameters are trained by back-propagating the critic loss into the encoder. The actor, critic, and encoder losses are minimized via gradient descent with mini-batches sampled from separate replay buffers for each task.

Offline reinforcement learning. In offline reinforcement learning, we assume that we have access to a dataset $\mathcal{D}$ collected by some policy behavior π_β. An RL agent must train on this fixed dataset and cannot interact with the environment. One challenge that offline RL poses is that the distribution of states and actions that an agent will see when deployed will likely be different from those seen in the offline dataset as they are generated by the agent, and a number of recent methods have tackled this distribution shift issue [15, 14, 33, 51, 39, 34]. Moreover, one can combine offline RL with meta-RL by training meta-RL on multiple datasets $\mathcal{D}_1, \dots, \mathcal{D}_{N_{\text{buff}}}$ [8, 36], but in the next section we describe some limitations of this combination.

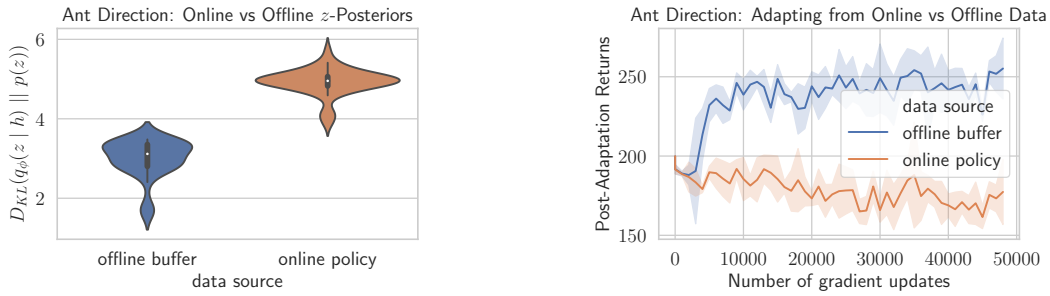

Figure 2: Left: The distribution of the KL-divergence between the posterior $q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$ and a prior $p(\mathbf{z})$ over the course of meta-training, when conditioned on data from the offline dataset (blue) or learned policy (orange), as measured by $D_{\text{KL}}(q_{\phi_e}(\mathbf{z} \mid \mathbf{h})||p_{\mathbf{z}}(\mathbf{z}))$. Note that data from the online policy is *not* available for meta-training here, but only used for measurement. We see that data from the learned policy results in posteriors that are substantially farther from the prior, suggesting a significant difference in distribution over $\mathbf{z}$. **Right:** The performance of the policy after adaptation when adapted using data from the offline dataset (i.e., $\mathbf{z} \sim p(\mathbf{z} \mid \pi_\beta)$) and data generated by the meta-trained policy (i.e., $\mathbf{z} \sim p(\mathbf{z} \mid \pi_\theta)$). During the offline training phase, we see that although the meta-RL policy adapts well when conditioned on $\mathbf{z}$ generated by the offline data, the performance does not increase when $\mathbf{z}$ is generated using data from the meta-trained exploration policy. Since the same policy is evaluated, the change in $\mathbf{z}$-distribution is likely the cause for the drop in performance.

4 The Problem with Naïve Offline Meta-Reinforcement Learning

Offline meta-RL methods can in principle utilize the same constraint-based approaches that standard offline RL algorithms have used to mitigate distributional shift. However, they must also contend with an additional distribution shift challenge that is specific to the meta-RL scenario: distribution shift in $\mathbf{z}$-space. Distribution shift in $\mathbf{z}$-space occurs because meta-learning requires learning an exploration policy π_θ that generates data for adaptation. However, offline meta-learning only trains the adaptation procedure $A_\phi(\mathbf{h})$ using offline data $\mathbf{h}$ generated by a previous behavior policy, which we denote as π_β. After offline training, there will be a mismatch between this learned exploration policy π_θ and the behavior policy π_β, leading to a difference in the data $\mathbf{h}$ and in turn, in the context variables $\mathbf{z} = A_\phi(\mathbf{h})$. In other words, if we write $p(\mathbf{z} \mid \pi)$ to denote the marginal distribution over $\mathbf{z}$ given data generated by policy π, the differences between trajectories from π_θ and π_β will result in differences between $p(\mathbf{z} \mid \pi_\beta)$ during offline training and $p(\mathbf{z} \mid \pi_\theta)$ at meta-test time.

To illustrate this issue, we empirically compare $p(\mathbf{z} \mid \pi_\beta)$ and $p(\mathbf{z} \mid \pi_\theta)$. While computing these distributions in closed form would require strong assumptions about how the policy, adaptation procedure, and environment interact, we approximate the these distributions by using a PEARL-style encoder discussed in Section 3: $p(\mathbf{z} \mid \pi) \approx q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$ where $\mathbf{h} \sim \pi$. We use this approximation to measure the KL-divergence observed during offline training between the posterior $p(\mathbf{z} \mid \pi)$ and a fixed prior $p_{\mathbf{z}}(\mathbf{z})$. If these two distributions were the same, then we would expect the distribution of KL divergences to also be similar. However, we see in Figure 2 that these two distributions are markedly different when analyzing a training run of SMAC on the Ant Direction task (see Section 6).

We also observe that this distribution shift negatively impacts the resulting policy. In Figure 2, we plot the performance of the learned policy when conditioned on $\mathbf{z}$ sampled from $q_{\phi_e}(\mathbf{z} \mid \pi_\beta)$ compared to $q_{\phi_e}(\mathbf{z} \mid \pi_\theta)$. We see that the policy conditioned on $\mathbf{z}$ generated from the behavior policy π_β data leads to improvement, while the same policy conditioned on $\mathbf{z}$ generated from the exploration policy π_θ slightly drops in performance. Since we evaluate the same policy π_θ and only change how $\mathbf{z}$ is sampled, this degradation in performance suggests that the policy suffers from distributional shift between $p(\mathbf{z} \mid \pi_\beta)$ and $p(\mathbf{z} \mid \pi_\theta)$: the encoder produces $\mathbf{z}$ vectors that too unfamiliar to the policy after reading in these exploration trajectories, and therefore actually attains better performance when conditioned on the trajectories from π_β.

We note that this issue arises in any method that trains non-Markovian policies with offline data. For example, recurrent policies for partially observed MDPs [24] depend both on the current observation $\mathbf{o}$ and a history $\mathbf{h}$. When deployed, these policies must also contend with potential distributional shifts between the training and test-time history distributions, in addition to the change in observation distribution $\mathbf{o}$. This additional distribution shift may explain why many memory-based recurrent

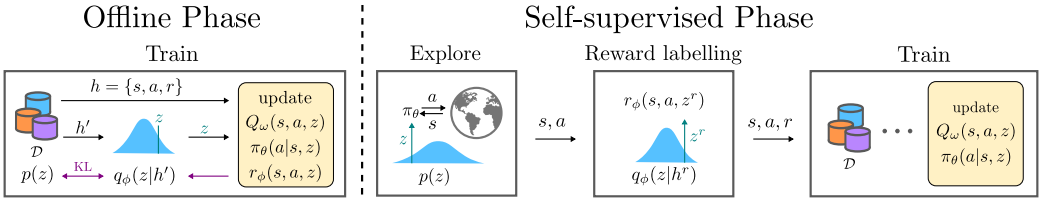

Figure 3: (Left) In the offline phase, we sample a history $\mathbf{h}'$ to compute the posterior $q_{\phi_e}(\mathbf{z} \mid \mathbf{h}')$. We then use a sample from this encoder and another history $\mathbf{h}$ to train the networks. In purple, we update the encoder q_{ϕ_e} with both reconstruction and KL loss. (Right) During the self-supervised phase, we explore by sampling $\mathbf{z} \sim p(z)$ and conditioning our policy on these observations. We label rewards using our learned reward decoder, and append the resulting data to the training data. The training procedure is equivalent to the offline phase, except that we do not train the reward decoder or encoder since no additional ground-truth rewards are observed.

policies are often trained online [9, 21, 10] or have benefited from refreshing the memory states [28]. In this paper, we focus on addressing this issue specifically in the offline meta-RL setting.

Offline meta-RL with self-supervised online training. In complex environments where many behaviors are possible, the distribution shift in z-space will likely be inevitable, since the learned policy is likely to deviate from the behavior policy. To address this issue, we introduce an additional assumption: in addition to the offline dataset, we assume that the agent can autonomously interact with the environment *without observing additional reward supervision*. This problem statement is useful for scenarios where autonomously interacting with the world is relatively easy, but online reward supervision is more expensive to obtain. For instance, it may be cheap to label rewards in an offline dataset for robotics by reward sketching [5], but expensive to have a labeler available online while the robot runs to provide rewards.

Formally, we assume that the agent can generate additional rollouts in an MDP without a reward function, $\mathcal{T} \setminus r = (\mathcal{S}, \mathcal{A}, \gamma, p_0, p_d)$. These additional interactions enable the agent to explore using the learned policy. These exploration trajectories are from the same distribution that will be observed at meta-test time, and therefore can be included into the meta-training process to mitigate the distributional shift issue described above. However, meta-training requires not just states and actions, but also rewards. In the next section, we describe a method for autonomously labeling these rollouts with *synthetic* reward labels to enable an agent to meta-train on this additional data.

5 Semi-Supervised Meta Actor-Critic

In this section, we present semi-supervised meta actor-critic (SMAC), a method that performs offline meta-training followed by self-supervised online meta-training. For the offline meta-training, we assume access to a set of replay buffers, $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N_{\text{buff}}}$, where each buffer corresponds to data for one task. For self-supervised online meta-training, we assume that we can sample MDPs without a reward function. The SMAC adaptation procedure consists of passing history through the encoder described in Section 3, resulting in a posterior $q(\mathbf{z} \mid \mathbf{h})$. SMAC then uses this posterior for both meta-RL training and for reward generation. Below, we describe both components of the algorithm.

5.1 Offline Meta-Training

To learn from the user-provided offline data, we adapt the PEARL meta-learning method [43] to the offline setting. We use an actor-critic algorithm to train a contextual policy using a set-based encoder, and update the critic by minimizing the Bellman error:

$$\mathcal{L}_{\text{critic}}(w) = \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}_i, z\sim q_{\phi_e}(\mathbf{z}|\mathbf{h}), \mathbf{a}'\sim\pi_\theta(\mathbf{a}'|\mathbf{s}',\mathbf{z})} \left[(Q_w(\mathbf{s}, \mathbf{a}, \mathbf{z}) - (r + \gamma Q_{\bar{w}}(\mathbf{s}', \mathbf{a}', \mathbf{z})))^2 \right], \quad (1)$$

where $\bar{w}$ are target network weights [37] updated with the soft update [35] of $\bar{w} \leftarrow \eta \cdot \bar{w} + (1 - \eta) \cdot w$.

PEARL uses soft actor critic (SAC) [19] to train their policy and Q-function. SAC has been primarily applied in the online setting, in which a replay buffer is continuously expanded by adding data from the latest policy. However, when naïvely applied to the offline setting, actor-critic methods such as SAC suffer from off-policy bootstrapping error accumulation [15, 33, 51], which occurs when the target Q-function for bootstrapping $Q(\mathbf{s}', \mathbf{a}')$ is evaluated at actions $\mathbf{a}'$ outside of the training data.

To avoid this error accumulation during offline training, we update our actor with a loss that implicitly constrains the policy to stay close to the actions observed in the replay buffer, following the approach in a previously proposed single-task offline RL algorithm called AWAC [39]. AWAC uses the following loss to approximate a constrained optimization problem, where the policy is constrained to stay close to the data observed in $\mathcal{D}$:

$$\mathcal{L}_{\text{actor}}(\theta) = \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}, \mathbf{z} \sim q_{\phi_e}(\mathbf{z}|\mathbf{h})} \left[\log \pi_\theta(\mathbf{a} \mid \mathbf{s}) \exp\left(\frac{Q(\mathbf{s},\mathbf{a},\mathbf{z}) - V(\mathbf{s}',\mathbf{z})}{\lambda} \right) \right]. \tag{2}$$

We estimate the value function $V(\mathbf{s},\mathbf{z}) = \mathbb{E}_{\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s},\mathbf{z})} Q(\mathbf{s},\mathbf{a},\mathbf{z})$ with a single sample, and λ is the resulting Lagrange multiplier for the optimization problem. See Nair et al. [39] for a full derivation.

This modified actor update makes it possible to train the encoder, actor, and critic on the offline data without the overestimation issues that afflict conventional actor-critic algorithms [33]. However, it does not address the $\mathbf{z}$-space distributional shift issue discussed in Section 4, because the exploration policy learned via this offline procedure will still deviate significantly from the behavior policy π_β. As discussed previously, we will aim to address this issue by collecting additional online data *without reward labels* and learning to generate reward labels if self-supervised meta-training.

Learning to generate rewards. To continue meta-training online without ground truth reward labels, we propose to use the offline dataset to learn a generative model over meta-training task reward functions that we can use to label the transitions collected online. Recall that during offline learning, we learn an encoder q_{ϕ_e} which maps experience $\mathbf{h}$ to a latent context $\mathbf{z}$ that encodes the task. In the same way that we train our policy $\pi_\theta(\mathbf{a} \mid \mathbf{s},\mathbf{z})$ that conditionally decodes $\mathbf{z}$ into actions, as well as a Q-function $Q_w(\mathbf{s},\mathbf{a},\mathbf{z})$ that conditionally decodes $\mathbf{z}$ into Q-values, we additionally train a *reward decoder* $r_{\phi_d}(\mathbf{s},\mathbf{a},\mathbf{z})$ [1] that conditionally decodes $\mathbf{z}$ into rewards. We train the reward decoder r_{ϕ_d} to reconstruct the observed reward in the offline dataset through a mean squared error loss.

Because we use the latent space $\mathbf{z}$ for reward-decoding, we back-propagate the reward decoder loss into q_{ϕ_e}. As visualized in Figure 3, we also regularize the posteriors $q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$ against a prior $p_{\mathbf{z}}(\mathbf{z})$ to provide an information bottleneck in that latent space $\mathbf{z}$ and ensure that samples from $p_{\mathbf{z}}(\mathbf{z})$ represent meaningful latent variables. We found it beneficial to not back-propagate the critic loss into the encoder, in contrast to prior work such as PEARL. To summarize, we train the reward encoder and decoder by minimizing the following loss

$$\mathcal{L}_{\text{reward}}(\phi_d,\mathbf{h},\mathbf{z}) = -\sum_{(\mathbf{s},\mathbf{a},r)\in\mathbf{h}} \|r - r_{\phi_d}(\mathbf{s},\mathbf{a},\mathbf{z})\|_2^2 + D_{\text{KL}}\left(q_{\phi_e}(\mathbf{z} \mid \mathbf{h}) \middle\| p_{\mathbf{z}}(\mathbf{z}) \right). \tag{3}$$

In the next section, describe how we use this reward decoder to generate new reward labels.

5.2 Self-Supervised Online Meta-Training

We now describe the self-supervised online training procedure, during which we use the reward decoder to provide supervision. First, we collect a trajectory τ by rolling out our exploration policy π_θ conditioned on a context sampled from the prior $p(\mathbf{z})$. To emulate the offline meta-training supervision, we would like to label τ with rewards that are in the distribution of meta-training tasks. As such, we sample a replay buffer $\mathcal{D}_i$ uniformly from $\mathcal{D}$ to get a history $\mathbf{h} \sim \mathcal{D}_i$ from the offline data. We then sample from the posterior $\mathbf{z} \sim q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$ and label the reward $r_{\text{generated}}$ of a new state and action, $(\mathbf{s},\mathbf{a})$, using the reward decoder

$$r_{\text{generated}} = r_{\phi_d}(\mathbf{s},\mathbf{a},\mathbf{z}), \qquad \text{where } \mathbf{z} \sim q_{\phi_e}(\mathbf{z} \mid \mathbf{h}) \tag{4}$$

We then add the labeled trajectory to the buffer and perform actor and critic updates as in offline meta-training. Lastly, since we do not observe additional ground-truth rewards, we do not update the reward decoder r_{ϕ_d} or encoder q_{ϕ_e}, and instead only train the policy and Q-function during the self-supervised phase. We visualize this procedure in Figure 3.

5.3 Algorithm Summary and Details

We visualize SMAC in Figure 3. For offline training, we assume access to offline datasets $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N_{\text{buff}}}$, where each buffer corresponds to data generated for one task. Each iteration, we sample a

[1] For simplicity, we write ϕ to represent the parameters of both the encoder and decoder.

Algorithm 1 Semi-Supervised Meta Actor-Critic

Input: datasets $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N_{\text{buff}}}$, policy π_θ, Q-function Q_w, encoder q_{ϕ_e}, and decoder r_{ϕ_d}.
for iteration $n = 1, 2, \ldots, N_{\text{offline}}$ **do** ▷ offline phase
    Sample buffer $\mathcal{D}_i \sim \mathcal{D}$ and two histories from buffer $\mathbf{h}, \mathbf{h}' \sim \mathcal{D}_i$.
    Use the first history sample to $\mathbf{h}$ to infer $\mathbf{z}$ encode it $\mathbf{z} \sim q_{\phi_e}(\mathbf{h})$.
    Update π_θ, Q_w, q_{ϕ_e}, r_{ϕ_d} by minimizing $\mathcal{L}_{\text{actor}}$, $\mathcal{L}_{\text{critic}}$, $\mathcal{L}_{\text{reward}}$ with samples $\mathbf{z}, \mathbf{h}'$.
for iteration $n = 1, 2, \ldots, N_{\text{online}}$ **do** ▷ self-supervised phase
    Collect trajectory τ with $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$, with $\mathbf{z}_t \sim p(\mathbf{z})$.
    Label the rewards in τ using Equation (4) and add the resulting data to $\mathcal{D}_i$.
    Sample buffer $\mathcal{D}_i \sim \mathcal{D}$ and two histories from buffer $\mathbf{h}, \mathbf{h}' \sim \mathcal{D}_i$.
    Encode first history $\mathbf{z} = q_{\phi_e}(\mathbf{h})$.
    Update π_θ, Q_w by minimizing $\mathcal{L}_{\text{actor}}$, $\mathcal{L}_{\text{critic}}$ with samples $\mathbf{z}, \mathbf{h}'$.

buffer $\mathcal{D}_i \sim \mathcal{D}$ and a history from this buffer $\mathbf{h} \sim \mathcal{D}_i$. We condition the stochastic encoder q_{ϕ_e} on this history to obtain a sample $\mathbf{z} \sim q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$. We then use this sample $\mathbf{z}$ and a second history sample $\mathbf{h}' \sim \mathcal{D}_i$ to update the Q-function, the policy, encoder, and decoder by minimizing Equation (1), Equation (2), and Equation (3) respectively. During the self-supervised phase, we found it beneficial to train the actor with a combination of the loss in Equation (2) and the original PEARL actor loss, weighted by hyperparameter λ_{pearl}. We provide pseudo-code for SMAC in Algorithm 1 and a complete list of hyperparameters, such as the network architecture and RL hyperparameters, in Appendix B.

6 Experiments

We proposed a method that uses additional online data to mitigate the distribution shift in $\mathbf{z}$-space that occurs in offline meta-RL. In this section, we evaluate how well the self-supervised phase of SMAC mitigates this negative drop in performance, and compare SMAC to other offline meta-RL methods on a range of meta-RL problems that require generalization to unseen tasks at meta-test time, across multiple simulated robotics domains.

Meta-RL Tasks We first evaluate our method on multiple simulated meta-learning tasks that have been used in past online and offline meta-RL papers [49, 12, 43, 8, 36] (see Figure 8). The first task, `Cheetah Velocity`, contains a two-legged "half cheetah" that can move forwards or backwards along the x-axis. The second task, `Ant Direction`, contains a quadruped "ant" robot that can move in a plane. We also evaluated SMAC on a significantly more diverse robot manipulation meta-learning task called `Sawyer Manipulation`, based on the goal-conditioned environment introduced in Khazatsky et al. [29]. `Sawyer Manipulation` is a simulated PyBullet environment [7] which comprises a Sawyer robot arm that can manipulate drawers, pick and place objects, and push buttons. Sampling a task $\mathcal{T} \sim p(\mathcal{T})$ involves sampling both a new configuration of the environment and the desired behavior to achieve. The initial configuration of the objects can vary drastically, with the presence and location of objects randomized as shown in Figure 9 and the agent is tested on one of three possible desired behaviors, such as pushing a button, opening a drawer, or lifting an object. In all of the environments, we test the meta-RL procedure's ability to generalize to new tasks by evaluating the policies on *held-out tasks* sampled from the same distribution as in the offline datasets. We give a complete description of the possible tasks and how the prior data was collected in Appendix B.

Comparisons and ablations. As an upper bound, we include the performance of PEARL with online training using oracle ground-truth rewards rather than self-generated rewards, which we label *Online Oracle*. To understand the impact of using the actor loss in Equation (2), we include an ablation in which we use the actor loss from PEARL but still employ our proposed unsupervised online phase, which we label *SMAC (actor ablation)*. We also include a meta-imitation baseline, which infers the task like PEARL, but then imitates the task data in the dataset. In this baseline, we replace the actor update in Equation (2) with simply maximizing $\log \pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$. We label this baseline *meta behavior cloning*. This baseline illustrates the gap between offline meta-RL and imitation, and helps us understand the gap between the (highly suboptimal) behavior policy and RL.

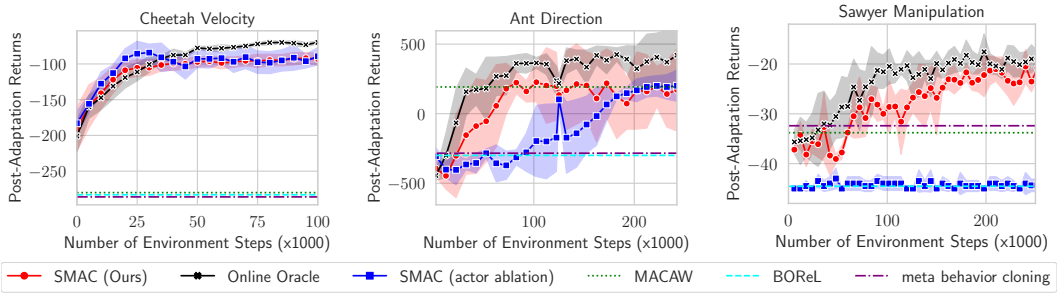

Figure 4: Comparison on self-supervised meta-learning against baseline methods. We report the final return of meta-test adaptation on unseen test tasks, with varying amounts of online meta-training following offline meta-training. Our method SMAC, shown in red, consistently trains to a reasonable performance from offline meta-RL (shown at step 0) and then steadily improves with online self-supervised experience. The offline meta-RL methods, MACAW [36] and BOReL at best match the offline performance of SMAC but have no mechanism to improve via self-supervision. We also compare to SMAC (SAC ablation) which uses SAC instead of AWAC as the underlying RL algorithm. This ablation struggles to pretrain a value function offline, and so struggles to improve on more difficult tasks.

For comparisons to prior work, we include the two previously proposed offline meta-RL methods: meta-actor critic with advantage weighting (labelled *MACAW*) [36] and Bayesian offline RL (labelled *BOReL*) [8]. Since these methods have only been applied to the offline phase, we report their performance only after offline training, since they do not have a self-supervised online stage. For both prior works, we used the code released by the authors. We trained these methods using the same offline dataset and matched hyperparameters when possible, such as batch size and network size.

Comparison results. We plot the mean post-adaptation returns and standard deviation across 4 seeds in Figure 4. We see that across all three environments, SMAC consistently improves during the self-supervised phase, and often achieves a similar performance to the oracle that uses ground-truth reward during the online phase of learning. SMAC also significantly improves over meta behavior cloning, which confirms that the data in the offline dataset is far from optimal.

We found that BOReL and MACAW performed comparatively poorly on all three tasks. A likely cause for this performance is that BOReL and MACAW were both developed assuming several orders of magnitude more data than the regime that we tested. For example, in the BOReL paper [8], the `Cheetah Velocity` was trained with an offline dataset using 400 million transitions and performs additional reward relabeling using ground-truth information about the transitions. In contrast, our offline dataset contains only 240 thousand transitions, roughly *three orders of magnitude* fewer transitions. Similarly, MACAW uses 100M transitions for `Cheetah Velocity`, over 40 times more transitions than used in our experiments. These prior methods also collect offline datasets by training *task-specific policies*, which converge to near-optimal policies within the first million time step [19], meaning that they utilize very high-quality data.

In contrast, our data collection protocol produces more realistic offline datasets that are highly suboptimal, as evidenced by the performance of the meta behavior cloning baseline, leaving plenty of room for improvement with offline RL. We also observed that our method improves over the performance of BOReL and MACAW even *before* the online phase (i.e., at zero new environment steps) on the Cheetah and `Sawyer Manipulation` tasks, and achieves a particularly large improvement on the `Sawyer Manipulation` environments, which are by far the most challenging and exhibit the most variability between tasks. In this domain, we also see the largest gains from the AWAC actor update, in contrast to the actor ablation (in blue), indicating that properly handling the offline phase is also important for good performance. In conclusion, our method (and its ablation) is the only one that is able to attain good generalization performance at meta-test time on these tasks, and actually attains performance that is close to the Online Oracle upper bound baseline, indicating that unsupervised online fine-tuning is highly effective for mitigating distributional shift in meta-RL, whereas without it offline meta-RL generally does not exceed the performance of a meta-imitation learning baseline.

References

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

[2] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., Mcgrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. URL `https://arxiv.org/pdf/1707.01495.pdfhttp://arxiv.org/abs/1707.01495`.

[3] Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pp. 4055–4065, 2017.

[4] Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Žídek, A., and Munos, R. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *arXiv preprint arXiv:1901.10964*, 2019.

[5] Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., Zolna, K., Aytar, Y., Budden, D., Vecerik, M., et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*, 2019.

[6] Colas, C., Sigaud, O., and Oudeyer, P.-Y. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *International Conference on Machine Learning (ICML)*, 2018.

[7] Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[8] Dorfman, R. and Tamar, A. Offline meta reinforcement learning. *arXiv preprint arXiv:2008.02598*, 2020.

[9] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RI^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[10] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pp. 1407–1416. PMLR, 2018.

[11] Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pp. 49–58. PMLR, 2016.

[12] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

[13] Fu, J., Luo, K., and Levine, S. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[14] Fujimoto, S., Conti, E., Ghavamzadeh, M., and Pineau, J. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

[15] Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062. PMLR, 2019.

[16] Grimm, C., Higgins, I., Barreto, A., Teplyashin, D., Wulfmeier, M., Hertweck, T., Hadsell, R., and Singh, S. Disentangled cumulants help successor representations transfer to new tasks. *arXiv preprint arXiv:1911.10866*, 2019.

[17] Gupta, A., Eysenbach, B., Finn, C., and Levine, S. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[18] Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. Meta-reinforcement learning of structured exploration strategies. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.

[19] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[20] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[21] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[22] Ho, J. and Ermon, S. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

[23] Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

[24] Jaakkola, T., Singh, S. P., and Jordan, M. I. Reinforcement learning algorithm for partially observable markov decision problems. *Advances in neural information processing systems*, pp. 345–352, 1995.

[25] Jabri, A., Hsu, K., Eysenbach, B., Gupta, A., Levine, S., and Finn, C. Unsupervised curricula for visual meta-reinforcement learning. *arXiv preprint arXiv:1912.04226*, 2019.

[26] Kaelbling, L. P. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume vol.2, pp. 1094 – 8, 1993.

[27] Kamienny, P.-A., Pirotta, M., Lazaric, A., Lavril, T., Usunier, N., and Denoyer, L. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934*, 2020.

[28] Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[29] Khazatsky, A., Nair, A., Jing, D., and Levine, S. What can i do here? learning new skills by imagining visual affordances. In *International Conference on Robotics and Automation*. IEEE, 2021.

[30] Kirsch, L., van Steenkiste, S., and Schmidhuber, J. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[31] Konyushkova, K., Zolna, K., Aytar, Y., Novikov, A., Reed, S., Cabi, S., and de Freitas, N. Semi-supervised reward learning for offline reinforcement learning. *arXiv preprint arXiv:2012.06899*, 2020.

[32] Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[33] Kumar, A., Fu, J., Tucker, G., and Levine, S. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[34] Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[35] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016. ISSN 10769757. doi: 10.1613/jair.301. URL `https://arxiv.org/pdf/1509.02971.pdf`.

[36] Mitchell, E., Rafailov, R., Peng, X. B., Levine, S., and Finn, C. Offline meta-reinforcement learning with advantage weighting. *arXiv preprint arXiv:2008.06043*, 2020.

[37] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[38] Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual Reinforcement Learning with Imagined Goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[39] Nair, A., Gupta, A., Dalal, M., and Levine, S. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[40] Peng, X. B., Coumans, E., Zhang, T., Lee, T.-W., Tan, J., and Levine, S. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 2020.

[41] Péré, A., Forestier, S., Sigaud, O., and Oudeyer, P.-Y. Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://arxiv.org/pdf/1803.00781.pdf`.

[42] Pong, V., Gu, S., Dalal, M., and Levine, S. Temporal Difference Models: Model-Free Deep RL For Model-Based Control. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://arxiv.org/pdf/1802.09081.pdf`.

[43] Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340. PMLR, 2019.

[44] Reddy, S., Dragan, A. D., and Levine, S. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

[45] Ross, S. and Bagnell, D. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

[46] Schaal, S. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3 (6):233–242, 1999.

[47] Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal Value Function Approximators. In *International Conference on Machine Learning (ICML)*, 2015.

[48] Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A. A., and Hardt, M. Test-Time Training with Self-Supervision for Generalization under Distribution Shifts. In *International Conference on Machine Learning (ICML)*, 2020. URL `https://test-time-training.github.io/`.

[49] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

[50] Warde-Farley, D., de Wiele, T. V., Kulkarni, T., Ionescu, C., Hansen, S., and Mnih, V. Unsupervised control through non-parametric discriminative rewards. *CoRR*, abs/1811.11359, 2018.

[51] Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[52] Xu, D. and Denil, M. Positive-unlabeled reward learning. *arXiv preprint arXiv:1911.00459*, 2019.

[53] Xu, Z., van Hasselt, H. P., and Silver, D. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31:2396–2407, 2018.

[54] Xu, Z., van Hasselt, H., Hessel, M., Oh, J., Singh, S., and Silver, D. Meta-gradient reinforcement learning with an objective discovered online. *arXiv preprint arXiv:2007.08433*, 2020.

[55] Zhao, T. Z., Nagabandi, A., Rakelly, K., Finn, C., and Levine, S. Meld: Meta-reinforcement learning from images via latent state models. *arXiv preprint arXiv:2010.13957*, 2020.

[56] Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: a very good method for bayes-adaptive deep rl via meta-learning. *Proceedings of ICLR 2020*, 2020.

[57] Zolna, K., Novikov, A., Konyushkova, K., Gulcehre, C., Wang, Z., Aytar, Y., Denil, M., de Freitas, N., and Reed, S. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.

Supplementary Material

A Additional Experimental Results

Exploration and offline dataset visualization In Figure 5, we visualize the post-adaption trajectories generated when conditioning the encoder the online exploration trajectories $\mathbf{h}_{\text{online}}$ and the offline trajectories $\mathbf{h}_{\text{offline}}$. Similar to Figure 7, and also visualize the online and offline trajectories themselves. We see that the exploration trajectories $\mathbf{h}_{\text{online}}$ and the offline trajectories $\mathbf{h}_{\text{offline}}$ are very different (green vs red, respectively), but the self-supervised phase mitigates the negative impact that this distribution shift has on offline meta RL. In particular, the post-adaptation trajectories conditioned on these two data sources (blue and orange) are similar after the self-supervised training, whereas before the self-supervised training, only the trajectories conditioned on the offline data (blue) move in multiple directions.

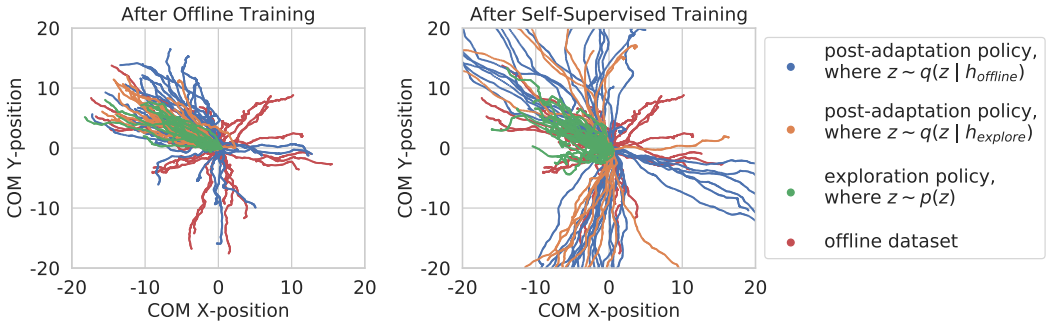

Figure 5: We duplicate Figure 7 but include the exploration trajectories (green) and example trajectories from the offline dataset (red). We see that the exploration policy both before and after self-supervised training primarily moves up and to the left, whereas the offline data moves in all direction. Before the self-supervised phase, we see that conditioning the encoder on online data (orange) rather than offline data (blue) results in very different policies, with the online data resulting in the post-adaptation policy only moving up and to the left. However, the self-supervised phase of SMAC mitigates the impact of this distribution shift and results in qualitatively similar post-adaptation trajectories, despite the large difference between the exploration trajectories and offline dataset trajectories.

Addressing state-space distribution shift by self-supervised meta-training on test tasks. Another source of distribution shift that can negatively impact a meta-policy is a distribution shift in state space. While this distribution shift occurs in standard offline RL, we expect this issue to be more prominent in meta RL, where there is a focus on generalizing to completely novel tasks. In many real-world scenarios, experiencing the state distribution of a novel task is possible, but it is the supervision (ie. reward signal) that is expensive to obtain. Can we mitigate state distribution shift by allow the agent to meta-train in the test task environments, but without rewards?

In this experiment, we evaluate our method, SMAC, when training online on the test tasks instead of on the meta-training tasks as in the experiments in Section 6. Prior work has explored this idea of self-supervision with test tasks in supervised learning [48] and goal-conditioned RL [29]. We use the `Sawyer Manipulation` environment to study how self-supervised training can mitigate state distribution shifts, as these environments contain significant variation between tasks. To further increase the complexity of the environment, we use a version of the environment which samples from a set of eight potential desired behaviors instead of three.

We compare self-supervised training on test tasks to self-supervised training on the set of meta-training tasks, which are also the tasks contained in the offline dataset. A large gap in performance indicates that interacting with the test tasks can mitigate the resulting distribution shift even when no reward labels are provided.

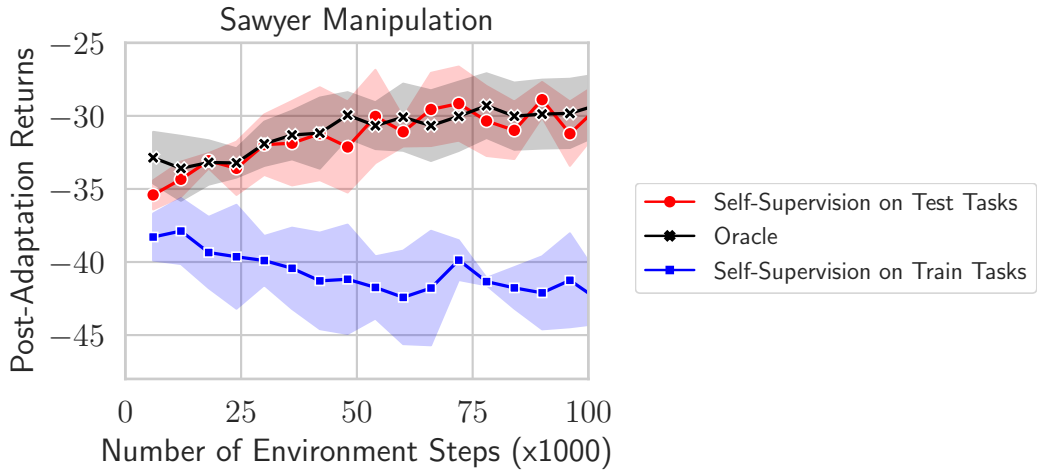

Figure 6: Learning curves when performing self-supervised training on the test environments (red) or the meta-training environments (blue). We also compare to an oracle that trains on test environments in combination with ground-truth rewards (black). We see that interacting with the test environment without rewards allows for steady improvement in post-adaptation test performance and obtains a similar performance to meta-training on those environments with ground-truth rewards.

We show the results in Figure 6 and find that there is indeed a large performance gap between the two training modes, with self-supervision on test tasks improving post-adaptation returns while self-supervision on meta-training tasks does not improve post-adaptation returns. We also compare to an oracle method that performs online training with the test tasks and the ground-truth reward signal. We see that SMAC is competitive with the oracle, demonstrating that we do not need access to rewards in order to improve on test tasks. Instead, the entire performance gain comes from experiencing the new state distribution of test tasks. Overall, these results suggest that SMAC is effective for mitigate distribution shifts in both $\mathbf{z}$-space and state space, even when an agent can interact in the environment without reward supervision.

Visualizing the distribution shift. We also investigate if the self-supervised training helps specifically because it mitigates a distribution shift caused by the exploration policy. To investigate this, we visualize the trajectories of the learned policy both before and after the self-supervised phase for the `Ant Direction` task in Figure 7. For each plot, we show trajectories from the policy $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ when the encoder $q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$ is conditioned on histories from either the offline dataset ($\mathbf{h}_{\text{offline}}$) or from the learned exploration policy ($\mathbf{h}_{\text{online}}$). Since the same policy is evaluated, differences between

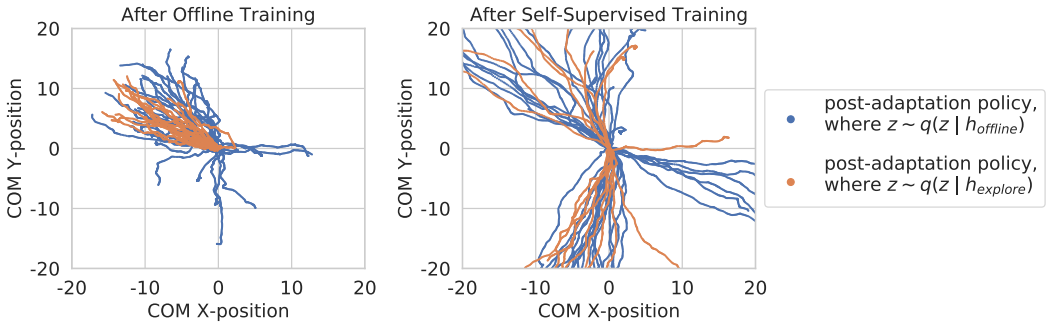

Figure 7: We visualize the visited XY-coordinates of the learned policy on the `Ant Direction` task. **Left:** Trajectories from the post-adaptation policy conditioned on $\mathbf{z} \sim q_{\phi_e}(\mathbf{z}|\mathbf{h})$ when $\mathbf{h}$ is sampled from the offline dataset (blue) or the learned exploration policy (orange) immediately after offline training. When conditioned on offline data, the policy correctly moves in many different directions. However, when conditioned on data from the learned exploration policy, the post-adaptation policy only moves up and to the left, suggesting that the post-adaptation policy is sensitive to data distribution used to collect $\mathbf{h}$. **Right:** After the self-supervised phase, we see that the post-adaptation policy learns to move in many different directions regardless of the data source. These visualization demonstrate that the self-supervised phase mitigates the distribution shift between conditioning on offline and online data.

the resulting trajectories represent the distribution shift caused by using history from the learned exploration policy rather than from the offline dataset.

We see that before the self-supervised phase, there is a large difference between the two modes that can only be attributed to the difference in $\mathbf{h}$. When using $\mathbf{h}_{\text{online}}$, the post-adaptation policy only explores one mode, but when using $\mathbf{h}_{\text{offline}}$, the policy moves in all directions. This qualitative difference explains the large performance gap observed in Figure 2 and highlights that the adaptation procedure is sensitive to the history $\mathbf{h}$ used to adapt. In contrast, after the self-supervised phase, the policy moves in all directions regardless of where the history came from. In Appendix A, we also visualize the exploration trajectories and found that the exploration trajectories are qualitatively similar both before and after the self-supervised phase. Together, these results illustrate the SMAC policy learns to adapt to the exploration trajectories by using the self-supervised phase to mitigate the distribution shift that occurs with naïve offline meta RL.

B Experimental Details

B.1 Environment Details

In this section, we describe the state and action space of each environment. We also describe how reward functions were generated and how the offline data was generated.

Ant Direction The `Ant Direction` task consists of controlling a quadruped "ant" robot that can move in a plane. Following prior work [43, 8], the reward function is the dot product between the agent's velocity and a direction uniformly sampled from the unit circle. The state space is $\mathbb{R}^{20}$, comprising the orientation of the ant (in quaternion) as well as the angle and angular velocity of all 8 joints. The action space is $[-1, 1]^8$, with each dimension corresponding to the torque applied to a respective joint.

The offline data is collected by running PEARL [43] on this meta RL task with 100 pre-sampled [2] target velocities. We terminate PEARL after 100 iterations, with each iteration containing at least 1000 new transitions. In PEARL, there are two replay buffers saved for each task, one for sampling data for training the encoder and another for training the policy and Q-function. We will call the former replay buffer the encoder replay buffer and the latter the RL replay buffer. The encoder replay buffer contains data generated by only the exploration policy, in which $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$. The RL replay buffer contains all data generated, including both exploration and post-adaptation, in which $\mathbf{z} \sim q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$. To make the offline dataset, we load the last 1200 samples of the RL replay buffer and the last 400 transitions from the encoder replay buffer into corresponding RL and encoder replay buffers for SMAC. In the initial submission, we mistakenly stated that we used 1200 samples when in fact we used 1600 samples for each task. During the self-supervised phase, we add all new data to both replay buffers.

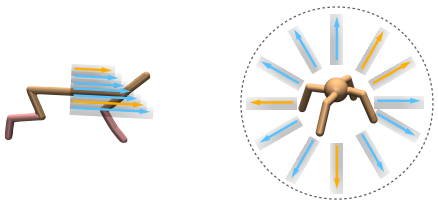

Figure 8: Illustrations of two evaluation domains, each of which has a set of meta-train tasks (examples shown in blue) and held out test tasks (orange). The domains include (left) a half cheetah tasked with running at different speeds and (right) a quadruped ant locomoting to different points on a circle.

Cheetah Velocity The `Cheetah Velocity` task consists of controlling a two-legged "half cheetah" that can move forwards or backwards along the x-axis. Following prior work [43, 8], the reward function is the absolute difference product between the agent's x-velocity and a velocity uniformly sampled from $[0, 3]$. The state space is $\mathbb{R}^{20}$, comprising the z-position; the cheetah's x- and z-velocity; the angle and angular velocity of each joint and the half-cheetah's y-angle; and the XYZ position of the center of mass. The action space is $[-1, 1]^6$, with each dimension corresponding to the torque applied to a respective joint.

[2] To mitigate variance coming from this sampling procedure, we use the same sampled target velocities across all experiments and comparisons. We similarly use a pre-sampled set of tasks for the other environments.

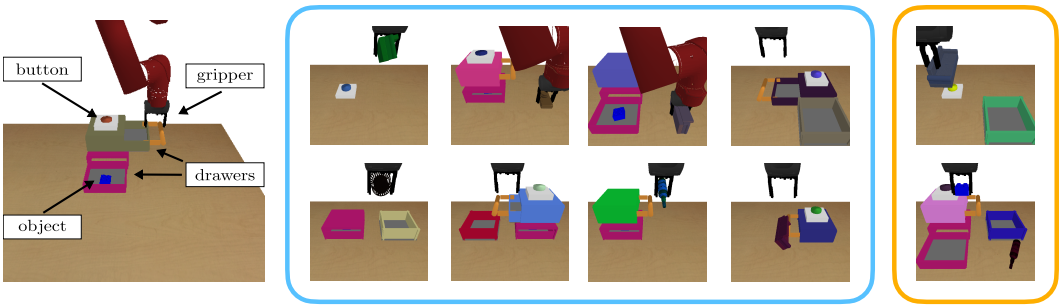

Figure 9: We propose a new meta-learning evaluation domain based on the environment from Khazatsky et al. [29], in which a simulated Sawyer gripper can perform various manipulation tasks such as pushing a button, opening drawers, and picking and placing objects. Each meta-training task contains a unique configuration of the objects (blue), and one meta-episode involves interacting with the given configuration for 3 trajectories. Similarly, we test the agent on held-out tasks (orange) in which different objects may be present and with completely different locations.

The offline data is collected in the same way as in the `Ant Direction` task, using a run from PEARL with 100 pre-sampled target velocities. For the offline dataset, we use the first 1200 samples from the RL replay buffer and last 400 samples from the encoder replay buffer after 50 PEARL iterations, with each iteration containing at least 1000 new transitions. For only this environment, we found that it was beneficial to freeze the encoder buffer during the self-supervised phase.

Sawyer Manipulation The state space, action space, and reward is described in Section 6. Tasks are generated by sampling the initial configuration, and then the desired behavior. There are five objects: a drawer opened by handle, a drawer opened by button, a button, a tray, and a graspable object. If an object is not present, it takes on position 0 in the corresponding element of the state space. First, the presence or absence of each of the five is randomized. Next, the position of the drawers (from 2 sides), initial position of the tray (from 4 positions), and the object (from 4 positions) is randomized. Finally, the desired behavior is randomly chosen from the following list, but only including the ones that are possible in the scene: "move hand", "open top drawer with handle", or "open bottom drawer with button". The offline data is collected using a scripted controller that does not know the desired behavior and randomly performs potential tasks in the scene, choosing another task if it finishes one task before the trajectory ends. This data is loaded into a single replay buffer used for both the encoder and RL.

Offline data collection. For the MuJoCo tasks, we generate data by following a similar procedure as [15], in which we use the replay buffer from a single PEARL run that uses the ground-truth reward. We limit the data collection to 1200 transitions or 6 trajectories per task and terminate the PEARL run early, forcing the meta-RL agent to learn from sub-optimal data. For `Sawyer Manipulation`, we collect data using a scripted policy that randomly performs as many potential tasks in the environment, without knowing what the desired behavior in a sampled task is. We used 50 training tasks and 50 trajectories of length 75 per task. In the offline dataset, the robot succeeds on the task 46% of the transitions.

B.2 Hyperparameters

We list the hyperparameters for training the policy, encoder, decoder, and Q-network in Table 1. If hyperparameters were different across environments, they are listed in Table 2. For pretraining, we use the same hyperparameters and train for 50000 gradient steps. Below, we give details on non-standard hyperparameters and architectures.

Batch sizes. The RL batch size is the batch size per task when sampling $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ tuples to update the policy and Q-network. The encoder batch size is the size of the history $\mathbf{h}$ per task used to conditioned the encoder $q_{\phi_e}(\mathbf{z} \mid \mathbf{h})$. The meta batch size is how many tasks batches were sampled and concatenated for both the RL and encoder batches. In other words, for each gradient update, the policy and Q-network observe (RL batch size) $\times$ (meta batch size) transitions and the encoder observes (RL batch size) $\times$ (encoder batch size) transitions.

Hyperparameter	Value
RL batch size	256
encoder batch size	64
meta batch size	4
Q-network hidden sizes	$[300, 300, 300]$
policy network hidden sizes	$[300, 300, 300]$
decoder network hidden sizes	$[64, 64]$
encoder network hidden sizes	$[200, 200, 200]$
$\mathbf{z}$ dimensionality (d_z)	5
hidden activation (all networks)	ReLU
Q-network, encoder, and decoder output activation	identity
policy output activation	tanh
discount factor γ	0.99
target network soft target η	0.005
policy, Q-network, encoder, and decoder learning rate	3×10^{-4}
policy, Q-network, encoder, and decoder optimizer	Adam
# of gradient steps per environment transition	4

Table 1: SMAC Hyperparameters for Self-Supervised Phase

Hyperparameter	`Cheetah Velocity`	`Ant Direction`	`Sawyer Manipulation`
horizon (max # of transitions per trajectory)	200	200	50
AWR β	100	100	0.3
reward scale	5	5	1
# of training tasks	100	100	50
# of test tasks	30	20	10
# of transitions per training task in offline dataset	1600	1600	3750
λ_{pearl}	1	1	0

Table 2: Environment Specific SMAC Hyperparameters

Encoder architecture. The encoder uses the same architecture as in Rakelly et al. [43]. The posterior is given as the product of independent factors

$$q_{\phi_e}(\mathbf{z} \mid \mathbf{h}) \propto \prod_{\mathbf{s},\mathbf{a},r \in \mathbf{h}} \Phi(\mathbf{z} \mid \mathbf{s}, \mathbf{a}, r),$$

where each factor is a multi-variate Gaussian over $\mathbb{R}^{d_z}$ with learned mean and diagonal variance. In other words,

$$\Phi_{\phi_e}(\mathbf{z} \mid \mathbf{s}, \mathbf{a}, r) = \mathcal{N}(\mu_{\phi_e}(\mathbf{s}, \mathbf{a}, r), \sigma_{\phi_e}(\mathbf{s}, \mathbf{a}, r)).$$

The mean and standard deviation is the output of a single MLP network with output dimensionality $2 \times d_z$. The output of the MLP network is split into two halves. The first half is the mean and the second half is passed through the softplus activation to get the standard deviation.

Self-supervised actor update. The parameter λ_{pearl} controls the actor loss during the self-supervised phase, which is

$$\mathcal{L}_{\text{actor}}^{\text{self-supervised}}(\theta) = \mathcal{L}_{\text{actor}}(\theta) + \lambda_{\text{pearl}} \cdot \mathcal{L}_{\text{actor}}^{\text{PEARL}}(\theta),$$

where $\mathcal{L}_{\text{actor}}^{\text{PEARL}}$ is the actor loss from PEARL [43]. For reference, the PEARL actor loss is

$$\mathcal{L}_{\text{actor}}^{\text{PEARL}}(\theta) = \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_i, \mathbf{z} \sim q_{\phi_e}(\mathbf{z}|\mathbf{h})} \left[D_{\text{KL}} \left(\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z}) \middle|\middle| \frac{\exp Q_w(\mathbf{s}, \mathbf{a}, \mathbf{z})}{Z(\mathbf{s})} \right) \right].$$

When the parameter λ_{pearl} is zero, the actor update is equivalent to the actor update in AWAC [39].