Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

Pre-Training for Robots: Offline RL Enables Learning New Tasks in a Handful of Trials

Aviral Kumar[*,1], Anikait Singh[*,1], Frederik Ebert[*,1], Mitsuhiko Nakamoto[1], Yanlai Yang[3], Chelsea Finn[2], Sergey Levine[1]
[1]UC Berkeley, [2]Stanford University, [3]New York University (*Equal contribution)

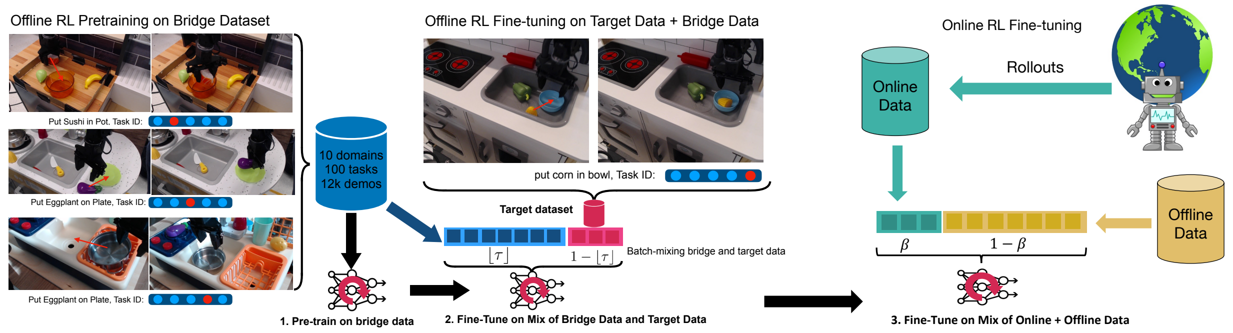
Offline RL Pretraining on Bridge Dataset
Put Sushi in Pot, Task ID:
Put Eggplant on Plate, Task ID:
Put Eggplant on Plate, Task ID:
10 domains
100 tasks
12k demos
1. Pre-train on bridge data
Offline RL Fine-tuning on Target Data + Bridge Data
put corn in bowl, Task ID:
Target dataset
$\lfloor \tau \rfloor$
$1-\lfloor \tau \rfloor$
Batch-mixing bridge and target data
2. Fine-Tune on Mix of Bridge Data and Target Data
Online RL Fine-tuning
Rollouts
Online Data
Offline Data
β
$1-\beta$
3. Fine-Tune on Mix of Online + Offline Data

Fig. 1: **Overview of PTR:** We first perform general offline pre-training on diverse multi-task robot data and subsequently fine-tune on one or several target tasks while mixing batches between the prior data and the target dataset using a batch mixing ratio of τ. Additionally, a separate online fine-tuning phase can be done, where offline pre-training is done on a static dataset and an online replay buffer is collected using rollouts in the environment. The offline and online buffers are mixed per batch with a ratio of β.

***Abstract*—Progress in deep learning highlights the tremendous potential of utilizing diverse robotic datasets for attaining effective generalization and makes it enticing to consider leveraging broad datasets for attaining robust generalization in robotic learning as well. However, in practice, we often want to learn a new skill in a new environment that is unlikely to be contained in the prior data. Therefore we ask: how can we leverage existing diverse offline datasets in combination with small amounts of task-specific data to solve new tasks, while still enjoying the generalization benefits of training on large amounts of data? In this paper, we demonstrate that end-to-end offline RL can be an effective approach for doing this, without the need for any representation learning or vision-based pre-training. We present pre-training for robots (PTR), a framework based on offline RL that attempts to effectively learn new tasks by combining pre-training on existing robotic datasets with rapid fine-tuning on a new task, with as few as 10 demonstrations. PTR utilizes an existing offline RL method, conservative Q-learning (CQL), but extends it to include several crucial design decisions that enable PTR to actually work and outperform a variety of prior methods. To our knowledge, PTR is the first RL method that succeeds at learning new tasks in a new domain on a real WidowX robot with as few as 10 task demonstrations, by effectively leveraging an existing dataset of diverse multi-task robot data collected in a variety of toy kitchens. We also demonstrate that PTR can enable effective autonomous fine-tuning and improvement in a handful of trials, without needing any demonstrations. An accompanying overview video can be found in the supplementary material and at this anonymous URL: https://sites.google.com/view/ptr-rss**

I. Introduction

Robotic learning methods based on reinforcement learning (RL) or imitation learning (IL) have led to impressive results [32, 21, 60, 22, 1], but the generalization abilities of policies learned this way are typically limited by the quantity and breadth of training data available. In practice, the cost of real-world data collection for each task means that such methods often use smaller datasets, which leads to more limited generalization. A natural way to circumvent this limitation is to incorporate existing diverse robotic datasets into the training pipeline of a robot learning algorithm, analogously to how pre-training on diverse prior datasets has enabled rapid fine-tuning in supervised learning. How can we devise methods that enable effective pre-training for robotic RL?

In most cases, answering this question requires devising a method that can pre-train on existing data from a wide range of tasks and domains, and then provide a good starting point for efficiently learning a *new* task in a *new* domain. Prior approaches utilize such existing data by running imitation learning (IL) [60, 9, 47] or by using representation learning [41] methods for pre-training and then fine-tuning with imitation learning. However, this may not necessarily lead to representations that can reason about the consequences of their actions. In contrast, end-to-end RL can offer a more general paradigm, that can be effective for both pre-training and fine-tuning, and is applicable even when assumptions in prior work are violated. Hence we ask, can we devise a simple and unified framework where *both* the pre-training and fine-tuning process uses RL? This presents significant challenges pertaining to leveraging large amounts of offline multi-task datasets, which would require high capacity models and this can be very challenging [4].

In this paper, we show that multi-task offline RL pre-training on diverse multi-task demonstration data followed by offline RL fine-tuning on a very small number of trajectories (as few as 10 trials, maximum 15) or online fine-tuning on autonomously collected data, can indeed be made into an effective robotic learning strategy that can significantly outperform methods based on imitation learning as well as RL-based methods that do not employ pre-training. This is surprising and significant, since prior work [37] has suggested that IL methods are superior to offline RL when provided with human demonstrations. Our framework, which we call PTR (pre-training for robots), is based on the CQL algorithm [27], but introduces a number of design decisions, that we show are critical for good performance and enable large-scale pre-training. These choices include a specific choice of architecture for providing high capacity while preserving spatial information, the use of group normalization, and an approach for feeding actions into the model that ensures that actions are used properly for value prediction. We experimentally validate these design decisions and show that PTR benefits from increasing the network capacity, even with large ResNet-50 architectures, which have never been previously shown to work with offline RL. Our experiments utilize the Bridge Dataset [9], which is an extensive dataset consisting of thousands of trials for a very large number of robotic manipulation tasks in multiple environments. A schematic of PTR is shown in Figure 1.

The main contribution of this work is a demonstration that PTR can enable offline RL pre-training on diverse real-world robotic data, and that these pre-trained policies can be fine-tuned to learn new tasks with just 10-15 demonstrations or with autonomously collected online interaction data in the real world. This is a significant improvement over prior RL-based pre-training and fine-tuning methods, which typically require thousands of trials [50, 22, 20, 6, 29]. We present a detailed analysis of the design decisions that enable offline RL to provide an effective pre-training framework, and show empirically that these design decisions are crucial for good performance. Although these decisions are based on prior work, we show that the novel combination of these components in PTR is important to make offline RL into a viable pre-training tool that can outperform other approaches.

II. Related Work

A number of prior works have proposed algorithms for offline RL [14, 26, 27, 24, 25, 54, 19, 13, 48]. In particular, many prior works study offline RL with multi-task data and devise techniques that perform parameter sharing[53, 43, 51, 11, 18], or perform data sharing or relabeling [61, 2, 62, 22, 57]. In this paper, our goal is not to develop new offline RL algorithms, but to show that these offline RL algorithms can be an effective tool to pre-train from prior data and then fine-tune on new tasks. We show that a few simple but important design decisions are essential for making offline RL pre-training scalable, and provide detailed experiments on fine-tuning these pre-trained models to new tasks.

Going beyond methods that only perform fine-tuning from a learned initialization with online interaction [40, 25, 31], we consider two independent fine-tuning settings: (1) the setting where we do not use any online interaction and fine-tune the pre-trained policy entirely offline, (2) the setting where a limited amount of online interaction is allowed to autonomously acquire the skills to solve the task from a challenging initial condition. This resembles the problem setting considered by offline meta-RL methods [33, 8, 39, 45, 34]. However, our approach is simpler as we fine-tune the very same offline RL algorithm that we use for pre-training. In our experiments, we observe that our method, PTR, outperforms the meta-RL method of Mitchell et al. [39].

Some other prior approaches that attempt to leverage large, diverse datasets via representation learning [36, 58, 59, 41, 16, 56, 35], as well as other methods for learning from human demonstrations, such as behavioral cloning methods with expressive policy architectures [47]. We compare to some of these methods [56, 41] in our experiments and find that PTR outperforms these methods. We also perform an empirical study to identify the design decisions behind the improved performance of RL-based PTR on demonstration data compared to BC, and find that the gains largely come from the ability of the value function in identifying the most "critical" decisions in a trajectory. While some prior works [37] shows results that suggest that offline RL underperforms imitation learning when provided with human demonstration data, our results show that offline RL can perform better than BC even with demonstrations, supporting the analysis in Kumar et al. [28].

The most closely related to our work are prior methods that run model-free offline RL on diverse real-world data and then fine-tune on new tasks [50, 22, 20, 6, 29]. These prior methods typically only consider the setting of *online* fine-tuning, whereas in our experiments, we demonstrate the efficacy of PTR for offline fine-tuning (where we must acquire a good policy for the downstream task using 10-15 demonstrations) *as well as* online fine-tuning considered in these prior works, where we must acquire a new task entirely via autonomous interaction in the real world.

III. Preliminaries and Problem Statement

An RL algorithm aims to learn a policy in a Markov decision process (MDP), which is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \mu_0, \gamma)$, where $\mathcal{S}, \mathcal{A}$ denote the state and action spaces, and $T(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, $r(\mathbf{s}, \mathbf{a})$ represent the dynamics and reward function respectively. $\mu_0(\mathbf{s})$ denotes the initial state distribution, and $\gamma \in (0, 1)$ denotes the discount factor. The policy $\pi(\mathbf{a}|\mathbf{s})$ learned by RL agents must optimize the long-term cumulative reward, $\max_\pi J(\pi) := \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \pi}[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$.

Problem statement. Our goal is to learn general-purpose initializations from a broad, multi-task offline dataset and then fine-tune these initializations to specific downstream tasks. We denote the general-purpose offline dataset by $\mathcal{D}$, which is partitioned into k chunks. Each chunk contains data for a given robotic task (e.g., picking and placing a given object) collected in a given domain (e.g., a particular kitchen). See Figure 1 for an illustration. Denoting the task/domain abstractly using an identifier i, the dataset can be formally represented as $\mathcal{D} = \cup_{i=1}^k (i, \mathcal{D}_i)$, where we denote the set of training tasks concisely as $\mathcal{T}_{\text{train}} = [k]$. Chunk $\mathcal{D}_i$ consists of data for a given task identifier i, and consists of a collection of transition tuples, $\mathcal{D}_i = \{(\mathbf{s}^i_j, \mathbf{a}^i_j, r^i_j, \mathbf{s}'^i_j)\}_{j=1}^n$ collected by a demonstrator on task i. Each task has a different reward function. Our goal is to utilize this multi-task dataset to help train a policy for one or multiple target tasks (denoted without loss of generality as task $\mathcal{T}_{\text{target}} = \{k+1, \cdots, n\}$).

While the diverse prior dataset $\mathcal{D}$ does not contain any experience for the target tasks, in the offline fine-tuning setting, we are provided with a very small dataset of demonstrations $\mathcal{D}^* := \{\mathcal{D}^*_{k+1}, \mathcal{D}^*_{k+1}, \cdots, \mathcal{D}^*_n\}$ corresponding to each of the target tasks. In our experiments, we use only 10 to 15 demonstrations for each target task, making it impossible to learn the target task from this data alone, such that a method that effectively maximizes performance for the target tasks $\mathcal{T}_{\text{target}}$ must leverage the prior data $\mathcal{D}$. We also study the setting where we aim to quickly fine-tune the policy learned via offline pre-training and offline fine-tuning using limited amounts of autonomously collected data via online real-world interaction. More details about this setup are provided in Section V-F.

Background and preliminaries. The Q-value of a given state-action tuple $Q^\pi(\mathbf{s}, \mathbf{a})$ for a policy π is the long-term discounted reward attained by executing action $\mathbf{a}$ at state $\mathbf{s}$ and following policy π thereafter. The Q-function satisfies the Bellman equation $Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}'}[Q^\pi(\mathbf{s}', \mathbf{a}')]$. Typical model-free offline RL methods [14, 26, 27] alternate between estimating the Q-function of a fixed policy π using the offline dataset $\mathcal{D}$ and then improving the policy π to maximize the learned Q-function. Our system, PTR, utilizes one such model-free offline-RL method, conservative Q-learning (CQL) [27]. We discuss how we adapt CQL for pre-training on diverse data followed by single-task fine-tuning in Section IV.

Tasks and domains. We use the Bridge Dataset [9] as the source of our pre-training tasks, which we augment with a few additional tasks as discussed in Section V. Our terminology for "task" and "domain" follows Ebert et al. [9]: a task is a skill-object pair, such as "put potato in pot" and a domain corresponds to an environment, which in the case of the Bridge Dataset consists of different toy kitchens, potentially with different viewpoints and robot placements. We assume the new tasks and environments come from the same training distribution, but are not seen in the prior data.

IV. Learning Policies for New Tasks from Offline RL Pre-training

To effectively solve new tasks from diverse offline datasets, a robotic learning framework must: **(1)** extract useful skills out of the diverse robotic dataset, and **(2)** rapidly specialize the learned skills towards an unseen target task, given only a minimal amount of experience from this target task in the form of demonstrations, or collected autonomously by interaction. In this section, we present our framework, PTR, that provides these benefits by training a single, highly expressive deep network via offline RL, and then specializes it on the target task with a small amount of data. We will first present the key components of our robotic framework in Section IV-A and then discuss our novel technical contributions, the practical design choices that are crucial, in Section IV-B.

A. The Components of PTR

To satisfy both requirements **(1)** and **(2)** from above, our framework uses a multi-task offline RL approach, where the policy and Q-function are conditioned on a task identifier. This allows us to share a single set of weights for all possible tasks in the diverse offline dataset, providing a general-purpose pre-training procedure that can use diverse data. Once a policy is obtained via this multi-task pre-training process, we adapt this policy for solving a new target task by utilizing a very small amount of target task data or autonomously collected data. We describe the two phases, pre-training and fine-tuning, below:

Phase 1: Multi-task offline RL pre-training. In the first phase, PTR learns a single Q-function and policy for all tasks $i \in \mathcal{T}_{\text{train}}$ conditioned on the task identifier i, i.e., $Q_\phi(\mathbf{s}, \mathbf{a}; i)$ and $\pi_\theta(\mathbf{a}|\mathbf{s}, i)$, via multi-task offline RL. We use a one-hot task identifier that imposes minimal assumptions on the task structure. For multi-task offline RL, we use the conservative Q-learning (CQL) [27] algorithm, extending it to the multi-task setting. This amounts to training the multi-task Q-function against a temporal difference error objective along with a regularizer that explicitly minimizes the expected Q-value under the learned policy $\pi_\theta(\mathbf{a}|\mathbf{s}; i)$, to prevent overestimation of Q-values for unseen actions, which can lead to poor offline RL performance [26]. Formally, the training objective for our multi-task Q-function, as prescribed by CQL, is given by:

$$\min_\phi \;\; \alpha \left(\mathop{\mathbb{E}}_{\substack{i \sim \mathcal{T}_{\text{train}}, \\ \mathbf{s} \sim \mathcal{D}_i, \mathbf{a} \sim \pi}} [Q_\phi(\mathbf{s}, \mathbf{a}; i)] - \mathop{\mathbb{E}}_{\substack{i \sim \mathcal{T}_{\text{train}}, \\ \mathbf{s}, \mathbf{a} \sim \mathcal{D}}} [Q_\phi(\mathbf{s}, \mathbf{a}; i)] \right) + \frac{1}{2} \mathop{\mathbb{E}}_{\substack{i \sim \mathcal{T}_{\text{train}}, \\ \mathbf{s}, \mathbf{a}, \mathbf{s}' \sim \mathcal{D} \\ \mathbf{a}' \sim \pi}} \left[\left(Q_\phi(\mathbf{s}, \mathbf{a}; i) - r - \gamma Q_{\bar{\phi}}(\mathbf{s}', \mathbf{a}') \right)^2 \right].$$

$Q_{\bar{\phi}}$ denotes the target Q-network, which is a delayed copy of the current Q-network. We train ϕ by running gradient descent on the above objective, and then optimize the learned policy to maximize the learned Q-values, along with an additional entropy regularizer as shown below:

$$\max_\theta \quad \mathbb{E}_{i \sim \mathcal{T}_{\text{train}}, \mathbf{s} \sim \mathcal{D}_i} \left[\mathbb{E}_{\mathbf{a} \sim \pi_\theta(\cdot|\mathbf{s}; i)} [Q_\phi(\mathbf{s}, \mathbf{a}; i)] \right] + \beta \mathcal{H}(\pi_\theta).$$

At the end of this multi-task offline training phase, we obtain a policy π_θ^{off} and Q-function Q_ϕ^{off}, that are ready to be fine-tuned to a new downstream task.

Phase 2: Offline or online fine-tuning of π_θ^{off} and Q_ϕ^{off} to a target task $\mathcal{T}_{\text{target}}$. In the second phase, PTR attempts to learn a policy to solve one or more downstream tasks by adapting π_θ^{off}, using a limited set of user-provided demonstrations that we denote $\mathcal{D}^*$, or using a combination of target demonstration data and autonomously collected online data. Our method for the offline fine-tuning setting is simple yet effective: we incorporate the new target task data into the replay buffer of the very same offline multi-task CQL algorithm from the previous phase and resume training from Phase 1. However, naïvely incorporating the target task data into the replay buffer might still not be effective since this scheme would hardly ever train on the target task data during adaptation due to the large imbalance between the sizes of the few target demonstrations and the large pre-training dataset. To address this imbalance, each minibatch passed to multi-task CQL during offline fine-tuning consists of a τ fraction of transitions from bridge demonstration data and $1 - \tau$ fraction of transitions from the target dataset. By setting τ to be small, we are able to prioritize multi-task CQL to look at target task data frequently, enabling it to make progress on the downstream task without overfitting.

For the autonomous online fine-tuning setting, we utilize a similar technique and have each mini-batch consist of β fraction of transitions from the bridge data and the target demonstration data, and $1 - \beta$ fraction of transitions from the newly collected online data. We alternate between collecting one trajectory and making 10 gradient steps for every single transition collected in the environment. Utilizing a high update to the data ratio allowed us to efficiently train the agent on newly collected online samples from rollouts.

Handling task identifiers for new tasks. The description of our system so far has assumed that the downstream test tasks are identified via a task identifier. In practice, we utilize a one-hot vector to indicate the index of a task. While such a scheme is simple to implement, it is not quite obvious how we should incorporate new tasks with one-hot task identifiers. In our experiments, we use two approaches for solving this problem: first, we can utilize a larger one-hot encoding that incorporates tasks in both $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{target}}$, but not use the indices for $\mathcal{T}_{\text{target}}$ during pre-training. The Q-function and the policy are trained on these *placeholder* task identifiers only during fine-tuning in Phase 2. Another approach for handling new tasks is to not use unique task identifiers for every new task, but rather "*re-target*" or re-purpose existing task identifiers for new target tasks in the fine-tuning phase. PTR provides this option: we can simply assign an already existing task identifier to the target demonstration data before fine-tuning the learned Q-function and the policy. For example, in our experiments in Section V we re-target the put sushi in pot task which uses orange transparent pots to instead put the sushi into a metal pot, which was never seen during training.

A complete overview of our approach is shown in Figure 1. We use a value of $\alpha = 10.0$ in multi-task CQL and $\tau = 0.8$ for mixing the pre-training dataset and the target task dataset in most of our experiments in the real-world, without requiring any domain-specific tuning. For online fine-tuning, we utilized $\alpha = 0.5$ to evenly mix between the online and offline datasets.

B. Important Design Choices and Practical Considerations

Even though the components discussed in Section IV-A are sufficient to give rise to an offline pre-training and fine-tuning approach, as we show in Section 5, this approach does not lead very good results on its own. Instead, we must make some crucial design decisions, including designing neural network architectures that can learn from diverse data with offline RL, cross-validation metrics to identify policies we expect to be effective after fine-tuning, and the design of the reward functions that can be used to label the pre-training dataset. We show that making the right choices for these components leads

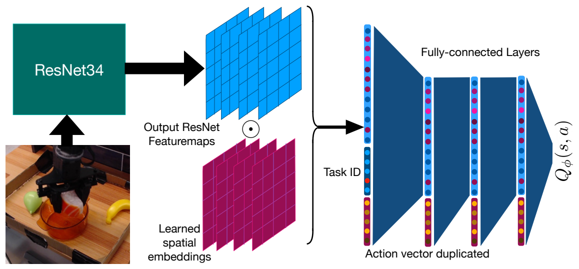

Fig. 2: **Q-function architecture for PTR.** The encoder is a ResNet34 with group normalization along with learned spatial embeddings (left). The decoder (right) is a MLP with the action vector duplicated and passed in at each layer. A one-hot task identifier is also passed into the input of the decoder.

to significant improvement (more than **3.5x** in final real-world performance; see Appendix E). Thus, describing, analyzing, and evaluating these choices is a crucial part of this work that we hope will facilitate applications of offline RL pre-training.

Policy and Q-function architectures. Perhaps the most crucial design decision for our approach is the neural network architecture for representing π^{off} and Q^{off}. Since we wish to fine-tune the policy for different tasks, we must use high-capacity neural network models for representing the policy and the Q-function. We experimented with a variety of standard (high-capacity) architectures for vision-based robotic RL. This includes standard convolutional architectures [50] and IMPALA architectures [11]. However, we observed in Figure 7 that these standard models were unable to effectively handle the diversity of the pre-training data and performed poorly. Then, we attempted to utilize standard ResNets [17] (ResNet-18, Resnet-34, and their adaptations to imitation problems from Ebert et al. [9]) to represent Q_ϕ, but faced divergence challenges similar to prior efforts that use batch normalization [4, 3] in the Q-network. Batch normalization layers are known to be hard to train with TD-learning [3] and, therefore, by replacing batch normalization layers with **group normalization** layers [55], we were able to address such divergence issues. See Appendix E for quantitative studies comparing these choices. Unlike prior work [30], we observed that with group normalization, we attain favorable scaling properties of PTR: the more the parameters, the better the performance as shown in Figure 7. We also observed that choosing an appropriate method for converting the three-dimensional feature-map tensor produced by the ResNet into a one-dimensional embedding plays a crucial role for learning accurate Q-functions and obtaining functioning policies. Unlike standard ResNet architectures for supervised learning, simply utilizing global average pooling (as used in many classification architectures) performs poorly. Instead we point-wise multiply the learned feature-map with a 3-dimensional parameter tensor before computing sums over the spatial dimensions which allows the network to explicitly encode spatial information. We refer to this technique as "**learned spatial embeddings**". An illustration of this architecture is provided in Figure 2. As detailed in Appendix E, Table XIII, we find that utilizing this technique leads to improved performance.

Next, we found that a Q-function $Q_\phi(\mathbf{s}, \mathbf{a})$ obtained by running naïve multi-task CQL on the demonstration data tends to not use the action input $\mathbf{a}$ effectively, due to strong correlations between $\mathbf{s}$ and $\mathbf{a}$ in the data, which is almost always the case for narrow, human demonstrations. As a result, policy improvement against such a Q-function overfits to these correlations, producing poor policies. To resolve this issue, we modified the architecture of Q-network to **pass the action a as input to every fully-connected layer** which, as shown in Figure 2 and Appendix E, Table XIV), greatly alleviates the issue and significantly improves over naïve CQL.

Cross-validation during offline fine-tuning. As we wish to learn task-specific policies that do not overfit to small amounts of data, we must apply the right number of gradient steps during fine-tuning: too few gradient steps will produce policies that do not succeed at the target tasks, while too many gradient steps will give policies that have likely lose the generalization ability of the pre-trained policy. To handle this trade-off, we adopt the following heuristic as a loose guideline: we run fine-tuning for many iterations while also plotting the learned Q-values over a held-out dataset of trajectories from the target task as seen in Figure 3. Then for evaluation, we pick the checkpoints that presented a Q-function with the Q-values appearing closest to having a monotonically increasing trend in a trajectory. This is a *relative* guideline and must be performed within the checkpoints observed within a run. The reason for this heuristic choice is that a valid Q-function must be a valid estimator for discounted return, and hence, it must increase over time-steps of a trajectory for a given task. Of course, this heuristic does not hold for arbitrary sub-optimal offline data, but all of our data comes from human-collected demonstrations. In principle, this heuristic can be wrapped into a metric quantifying degree of monotonicity of the Q-value curve in Figure 3, but in our experiments, we felt this was not necessary: as we show below, we were able to narrow down the checkpoints to essentially one or at most, two checkpoints by just visual inspection. Of course, designing an accurate metric would be helpful for future work. We present two worked-out examples of our checkpoint selection strategy for two tasks from Scenario 1 and Scenario 3 in Figure 3. Observe that checkpoints early in training exhibit Q-values that fluctuate arbitrarily at the beginning of training, which is clearly non-monotonic. This is because of the lack of sufficient gradient steps for fine-tuning the target task. Once sufficient gradient steps are performed, the Q-values visibly improve on the monotonicity property. Training further leads to much flatter Q-values, that are visibly less monotonic.

To validate our checkpoint selection mechanism, in Figure 4 we present a film-strip of a sample evaluation of a good and a poor checkpoint as identified by the cross-validation strategy mentioned above. We observe that the checkpoint with more flat Q-values fails to solve the door opening task, whereas the one with a visibly increasing Q-value trend solves the task.

Reward specification. In this paper, we aim to pre-train on existing robotic datasets, such as the Bridge Dataset [9], which consists of human-teleoperated demonstration data. Although the demonstrations are all successful, they are not annotated with any reward function. Perhaps an obvious choice is to label the last transition of each trajectory as success, and give it a +1 binary reward. However, in several of the datasets we use, there can be a 0.5-1.0 second lag between task completion and when the episode is terminated by the data collection. To ensure that a successful transition is not incorrectly labeled as 0, we utilized the practical heuristic of annotating the last $n = 3$ transitions of every trajectory with a reward of $+1$ and and annotated other states with a 0 reward. We show in Appendix D that this provided the best results. In principle, more complicated methods of reward labeling [12] could be used. However, we found the presented rule to be simple and yet effective to learn good policies.

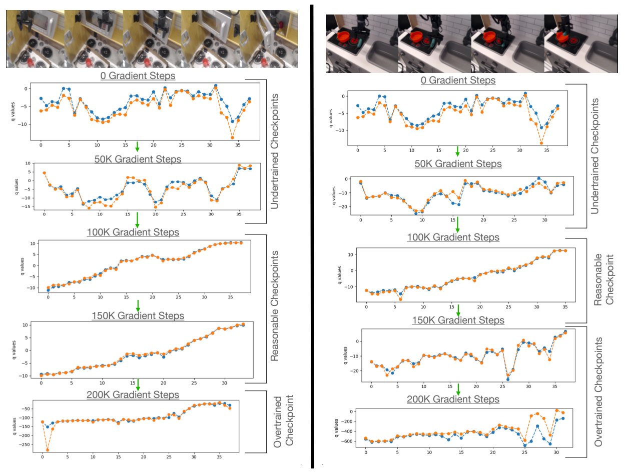

Fig. 3: **Evolution of Q-values on the target task over the process of fine-tuning with PTR.** Observe that while the learned Q-values on *held-out* trajectories from the dataset just at the beginning of Phase 2 (fine-tuning) do not exhibit a roughly increasing trend, we choose to evaluate those checkpoints of PTR that exhibit a visible more increasing trend in the Q-values despite having access to only 10 demonstrations for these target tasks.

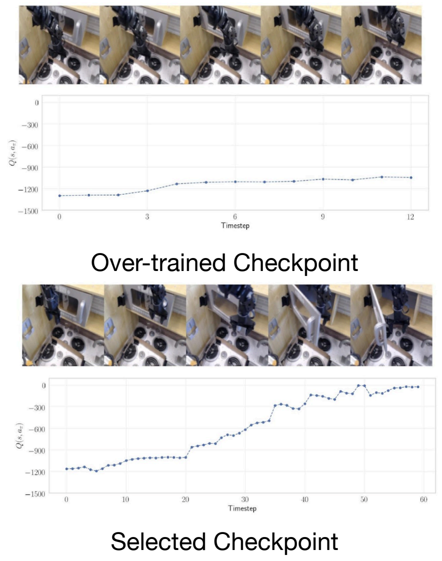

Fig. 4: **Performance evaluation of a selected and over-trained checkpoint of PTR.** We validate our checkpoint selection mechanism on the door opening task. An over-trained checkpoint with nearly flat Q-values fails to solve the task, whereas a checkpoint with visibly increasing Q-values solves the task.

V. Experimental Evaluation of PTR and Takeaways for Robotic RL

The goal of our experiments is to validate if PTR can learn effective policies from only a handful of user-provided demon-

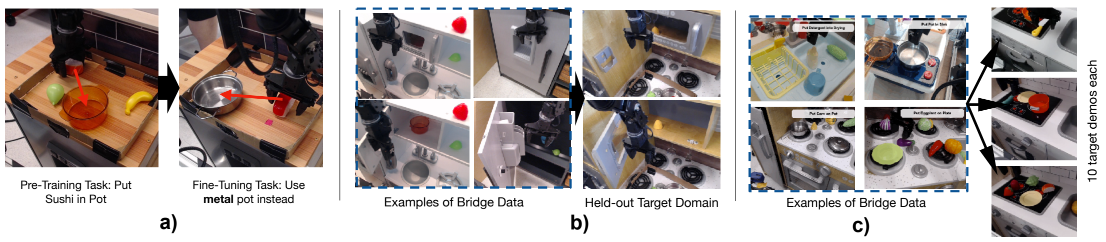

Fig. 5: **Illustrations of the three real-world experimental setups we evaluate PTR on: (a)** the "put sushi in a metallic pot" task which requires retargeting, **(b)** the task of opening an unseen door, and **(c)** fine-tuning on several novel target tasks in a held out toykitchen environment.

strations for a target task, by effectively utilizing previously-collected robotic datasets for pre-training. We also aim to understand whether the design decisions introduced in Section IV-B are crucial for attaining good robotic manipulation performance. To this end, we evaluate PTR in a variety of robotic manipulation settings, and compare it to state of the art methods, which either do not use offline RL or do not learn end-to-end by employing some form of visual representation learning. We evaluate in three scenarios: **(a)** when the target task requires retargeting the behavior of an existing skill, in this case changing the type of object types it interacts with, **(b)** when the target task requires performing a previously observed task but this time in a previously unseen domain, and **(c)** when the target task requires learning a new skill in a new domain, by using the target demonstrations. We also perform a diagnostic study in simulation in Appendix A (Table VII).

A. Setup and Comparisons

Real-world experimental setup. We directly utilize the publicly available *Bridge Dataset* [9] for pre-training, as it provides a large number of robot demonstrations for a diverse set of tasks in multiple domains, i.e., multiple different toy kitchens. We use the same WidowX250 robot platform for our evaluations. The bridge dataset contains distinct tasks, each differing in terms of the objects that the robot interacts with and the domain the task is situated in. We assign a different task identifier to each task in the dataset for pre-training. We also evaluate on an additional door-opening task not present in the Bridge Dataset, where we collected demonstrations for opening and closing a variety of doors, and test our system on new, unseen doors. More details are in Appendix B.

Comparisons. Since the datasets we use (both the pre-training bridge dataset from [9] and the newly collected door opening data) consist of human demonstrations, as indicated by prior work [37], the strongest prior method in this setting is behavioral cloning (BC), which attempts to simply imitate the action of the demonstrator based on the current state. We incorporate BC in a pipeline similar to PTR, denoted as **BC (finetune)**, where we first run BC on the pre-training dataset, and then finetune it using the demonstrations on the target task using the same batch mixing as in PTR. To ensure that our BC baselines are well-tuned, we utilize standard practices of cross-validation via a held-out validation set to tune hyperparameters and make early stopping decisions are we elaborate on in Appendix D2. Next, to assess the importance of performing pre-training *followed* by fine-tuning, we compare PTR to **(i)** jointly training on the pre-training and fine-tuning data with CQL, which is equivalent to the COG approach of Singh et al. [50], **(ii)** multi-task offline CQL (**CQL (zero-shot)**) that does not use the target demonstrations at all, and **(iii)** utilizing CQL to train on target demonstrations alone from scratch, with no pre-training data included (**CQL (target data only)**). We also make the analogous comparison for BC, jointly training BC on the pre-training and target task data from scratch (**BC (joint)**) which is equivalent to [9]. For fairness of comparison, BC, CQL, and PTR (both for zero-shot, joint-training and fine-tuning) use the *same* exact architecture, including our learned-spatial embedding described in Section IV-B.

B. Experimental Results

Method	**Success rate**
BC (zero-shot)	0/30
BC (finetune)	0/30
CQL (zero-shot)	2/30
PTR (Ours)	**14/30**

TABLE I: **Performance of PTR for "put sushi in metallic pot" in Scenario 1.** PTR substantially outperforms BC (finetune), even though it is provided access to only demonstration data. We also show some examples comparing some trajectories of BC and PTR in Appendix D.

Scenario 1: Re-targeting skills for existing tasks to handle new objects. We utilized the subset of the bridge data with pick-and-place tasks in one toy kitchen for pre-training, and selected the "put sushi in pot" task as our target task. This task is depicted in the bridge dataset, but only using an orange transparent pot (see Figure 5 (a)). In order to construct a scenario where the offline policy at the end of pre-training must be re-targeted to act on a different object, we collected only *ten* demonstrations that place the sushi in a metallic pot and used these demonstrations for fine-tuning. This scenario is challenging since the metallic pot differs significantly from the orange transparent pot visually. By pre-training on all pick-and-place tasks in this domain (32 tasks) and fine-tuning on this data and 10 demonstrations, PTR is able to obtain a policy that is re-targeted towards the metal pot. On the other hand, the policy learned by BC confuses arbitrary patches on the tabletop with the pot. Quantitatively, observe in Table I that PTR is able to complete the task with reasonable accuracy across a set of easy and hard initial positions, whereas zero-shot and fine-tuned BC are completely unable to solve the task. The fact that zero-shot CQL has difficulty solving the task indicates that target demonstrations are necessary, and PTR is able to attain successful behavior with just ten demonstrations.

			zero-shot		Joint Training		Target data only	
Task	**PTR (Ours)**	**BC (fine.)**	**CQL**	**BC**	**COG**	**BC**	**CQL**	**BC**
Open Door	**12/20**	10/20	0/20	0/20	5/20	7/20	4/20	7/20

TABLE II: **Successes vs. total trials for opening a new target door in Scenario 2**. PTR outperforms both BC (finetune) and BC (joint) given access to the same data. Note that joint training is worse than finetuning from the pre-trained initialization.

Scenario 2: Generalizing to previously unseen domains. Next, we study whether PTR can adapt behaviors seen in the pre-training data to new domains. We study a door opening task, which requires significantly more complex maneuvers and precise control compared to the pick-and-place tasks from above (as seen in the video present in the supplementary material and our anonymous website). The doors in the pre-training data exhibit different sizes, shapes, handle types and visual appearances, and the target door (shown in Figure 5(b)) we wish to open and the corresponding toy kitchen domain are never seen previously in the pre-training data. Concretely, for pre-training, we used a dataset of 800 door-opening demonstrations on 12 different doors in 4 different toy kitchen domains, and we utilize 15 demonstrations on a held-out door for fine-tuning. Table II shows that PTR improves over both BC baselines and joint training with CQL (or COG). Due to the limited target data and the associated task complexity, in order to succeed, an method must effectively leverage the pre-training data to learn a general policy that attempts to solve the task, and then specialize it to the target door.

Interestingly, Table II shows that while jointly training on the pre-training and fine-tuning data (or COG [50]) by itself does not outperform BC (joint), the pre-training and fine-tuning approach in PTR leads to significantly better performance, improving over the best BC approach. Since CQL (joint) is equivalent to PTR, but with no Phase 1, this large performance gap indicates the efficacy of offline RL methods trained on large diverse datasets at providing good initializations for learning new downstream tasks. We believe that this finding may be of independent interest to robotic offline RL practitioners: when utilizing multi-task offline RL, it might be better first to run multi-task pre-training followed by fine-tuning, as opposed to jointly training from scratch.

Scenario 3: Learning to solve new tasks in new domains. Finally, we evaluate the efficacy of PTR in learning to solve a new task in a new domain. This scenario presents a generalization requirement that is significantly more challenging than the previously studied scenarios, since both the task and the domain are never seen before. This task is represented via a new task identifier, and pre-training receives no data for this task identifier, or even any data from the kitchen where this task is situated. We pre-train on all 80 pick-and-place style tasks from the bridge dataset, while holding out any data from the new task kitchen, and then fine-tune on 10 demonstrations for 4 target tasks independently in this new kitchen, as shown in Table III. Methods that utilize more expressive policy architectures (an auto-regressive policy or behavior transformers (BeT) [47]) do not lead to improved performance compared to the standard BC (finetune) approach, and we find that PTR outperforms these approaches. Please find more details on the implementation of auto-regressive BC and BeT in Section D2. This might appear surprising, and perhaps just a hyper-parameter tuning artifact at first, but we present additional qualitative and quantitative analysis aiming at understanding the reasons behind why our offline RL-based PTR approach works better in Section V-D. We also compare to MACAW [39], an offline meta-RL method that utilizes advantage-weighted regression [44] for gradient-based few-shot adaptation, and find that this approach is unable to learn policies that succeed. We discuss the hyperparameter configurations that we tried for this approach in Appendix C4. Finally, observe in Table III that joint training with CQL or BC, or just using target data, without any pre-training for CQL or BC, all perform significantly worse than PTR.

C. *Comparison to non-RL Visual Pre-Training Methods*

We also compare PTR to approaches that utilize the diverse bridge dataset or Internet-scale data for task-agnostic visual representation learning, followed by down-stream behavioral cloning only on the target fine-tuning task which utilizes the representation learned during pre-training. In particular, we compare to two approaches: R3M [41], which utilizes the Ego4D dataset of human videos to obtain a representation, and MVP [46, 56], which trains a masked auto-encoder [16] on the Bridge Dataset and utilizes the learned latent space as the representation of the new image. Observe in Table IV that, while utilizing R3M or MAE does improve over running BC on the target data alone (compare R3M and MAE in Table IV to BC on target data only in Table III), the pre-training scheme from PTR outperforms both of these prior pre-training approaches, indicating the efficacy of offline RL pre-training on diverse robot data in recovering useful representations for downstream policy learning.

D. *Understanding the Benefits of PTR over BC*

One natural question to ask given the results in this paper is: why does utilizing an offline RL method for pre-training and fine-tuning as in PTR outperform BC-based methods even though the dataset is quite "BC-friendly", consisting of only demonstrations? The answer to this question is not obvious, especially since joint training with BC still outperforms jointly training with CQL on both pre-training and target demonstration data (COG) in our results in Table III.

To understand the reason behind improvements from RL, we perform a qualitative evaluation of the policies learned by PTR and BC (finetune) on two tasks: take croissant from metal bowl and put cucumber in bowl in Figure 6. We find that the failure mode of BC policies can be primarily explained as a lack of precision in locating the object, or a prematurely-executed

		BC finetuning			Joint training		Target data only		Meta-learning
Task	**PTR (Ours)**	**BC (fine.)**	**Autoreg. BC**	**BeT**	**COG**	**BC**	**CQL**	**BC**	**MACAW**
Take croissant from metal bowl	**7/10**	3/10	5/10	1/10	4/10	4/10	0/10	1/10	0/10
Put sweet potato on plate	**7/20**	1/20	1/20	0/20	0/20	0/20	0/20	0/20	0/20
Place knife in pot	**4/10**	2/10	2/10	0/10	1/10	3/10	3/10	0/10	0/10
Put cucumber in pot	**5/10**	0/10	1/10	0/10	2/10	1/10	0/10	0/10	0/10

TABLE III: **Performance of PTR and other baseline methods for new tasks in Scenario 3.** Note that PTR outperforms all other baselines including BC (finetune), BC with more expressive policy classes (BeT [47], Auto-regressive), offline RL with no pre-training ("Target data only") and joint training [50, 9]. PTR also outperforms few-shot gradient-based meta learning methods such as MACAW [39], which fail to attain non-zero performance.

		Pre-train. rep. + BC finetune	
Task	**PTR (Ours)**	**R3M**	**MAE**
Take croissant from bowl	**7/10**	1/10	3/10
Put sweet potato on plate	**7/20**	0/20	1/20
Place knife in pot	**4/10**	0/10	0/10
Put cucumber in pot	**5/10**	0/10	0/10

TABLE IV: **Performance of PTR and other pre-training methods (R3M and MAE).** While both R3M [41] and MAE [56] help improve performance over naïvely applying BC on the target data, PTR outperforms both.

grasping action. This is especially prevalent in settings where the object of interest is farther away from the robot gripper at the initial state, and hints at the inability of BC to prioritize learning the critical decisions (e.g., precisely moving over the object before the grasping action) over non-critical ones (e.g., the action to take to reach nearby the object from farther away). On the other hand, RL can learn to make such critical decisions correctly as shown in Figure 6. We present additional rollouts in Appendix B.

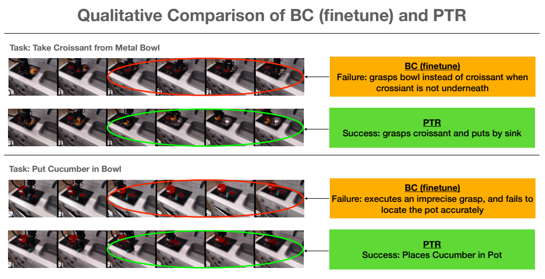
Qualitative Comparison of BC (finetune) and PTR
Task: Take Croissant from Metal Bowl
BC (finetune)
Failure: grasps bowl instead of croissant when crossiant is not underneath
PTR
Success: grasps croissant and puts by sink
Task: Put Cucumber in Bowl
BC (finetune)
Failure: executes an imprecise grasp, and fails to locate the pot accurately
PTR
Success: Places Cucumber in Pot

Fig. 6: **Qualitative successes of PTR visualized alongside failures of BC (fine-tune).** As an example, observe that while PTR is accurately able to reach to the croissant and grasp it to solve the task, BC (finetune) is imprecise and grasps the bowl instead of the croissant resulting in failure.

Task	BC (finetune)	PTR	AW-BC (finetune)
Cucumber	0/10	5/10	5/10
Croissant	3/10	7/10	6/10

TABLE V: **Performance of advantage-weighted BC** on two tasks from Table III. Observe that weighting the BC objective using advantage estimates from the Q-function learned by PTR leads to much better performance than standard BC (finetune), almost recovering PTR performance. This test indicates that the Q-function in PTR allows us to be accurate on the more critical decisions, thereby preventing the failures of BC.

Next, to verify if the performance benefits can be explained by the ability of Q-learning to prioritize critical decisions, we run a form of weighted behavioral cloning, where the weights $w_\phi(\mathbf{s}, \mathbf{a})$ are derived from the *advantage estimates computed using a frozen Q-function learned by PTR* after fine-tuning:

$$w_\phi(\mathbf{s}, \mathbf{a}) \propto \exp(Q_\phi(\mathbf{s}, \mathbf{a}) - \max_{\mathbf{a}'} Q_\phi(\mathbf{s}, \mathbf{a}')).$$

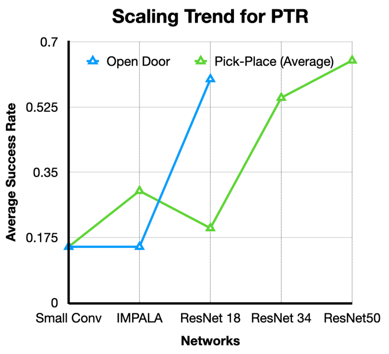

Fig. 7: **Scaling trends for PTR** on the open door task, and average over two pick and place tasks from Scenario 3. Note that with our design decisions, PTR is able to effectively benefit from high capacity networks.

Note that this is not the same as standard advantage-weighted regression [44], which uses Monte-Carlo return estimates for computing advantage weights instead of using advantages computed under a Q-function trained via PTR or CQL. As shown in Table IX, we find that this advantage-weighted BC (AW-BC) approach performs significantly better than BC (finetune) method and comparably to PTR, for two tasks (croissant and cucumber from Table III. Since AW-BC is essentially the same as BC, just with a modified weight to indicate the importance of any transition, this performance improvement clearly indicates the benefits of learning value functions via PTR in a pre-training then fine-tuning setting, even when we only have demonstration data. Note that since AW-BC uses the PTR-derived weights after fine-tuning, it cannot serve as an independent method, but rather amounts to another way to use the PTR value function.

E. *Effective Use of High-Capacity Neural Networks*

To understand the importance of designing techniques that enable us to use high-capacity models for offline RL, we examine the efficacy of PTR with different neural network architectures on the open door task from Scenario 2, and the put cucumber in pot and take croissant out of metallic bowl tasks from Scenario 3. We compare to standard three-layer convolutional network architectures used by prior work for Deepmind control suite tasks (see for example, Kostrikov et al. [23]), an IMPALA [11] ResNet that consists of 15 convolutional layers spread across a stack of 3 residual blocks, and the ResNet 18, 34, and 50 architectures with our proposed design decisions. Observe in Figure 7 that the performance of smaller networks (Small, IMPALA) is significantly worse than the ResNet in the door opening task. For the pick-and-place tasks that contain a much larger dataset, Small, IMPALA and ResNet18 all perform much worse than ResNet 34 and ResNet

	SACfD	PTR (offline → online)
All positions	0% → 0%	**53% → 73%**
Novel OOD positions	0% → 0%	**13% → 60%**

TABLE VI: **Performance before and after online fine-tuning.** The success rate of the PTR pre-trained policy is improved significantly from online fine-tuning, especially on novel out-of-distribution (OOD) initial positions that must be learned entirely from autonomous interaction in the real world. The results are reported as the average of 3 trials from each initial position.

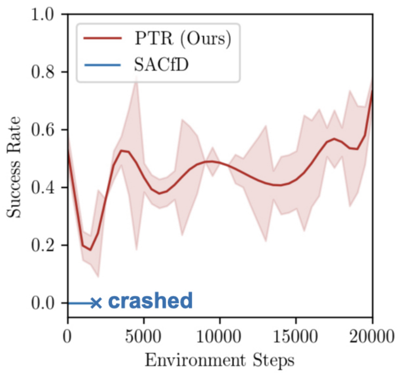

Fig. 8: **Online fine-tuning for PTR** on the open door task. PTR improves the success rate of the pre-trained policy from 53% to 73% (from 13% to 60% for the harder positions), while SACfD crashes due to unsafe behavior during exploration. We ran PTR online fine-tuning for 2 seeds in the real world.

50. In Appendix E we show that ResNet 34 models perform much worse if our prescribed design decisions are not used.

F. Autonomous Online Fine-Tuning

So far, we've evaluated PTR with offline fine-tuning to new tasks. However, by pre-training representations with offline RL, we can also enable autonomous improvement through online RL fine-tuning. In this section, we will demonstrate this benefit by showing that an offline initialization learned by PTR pre-training can be effectively fine-tuned autonomously with online rollouts. This procedure provides a way forward to build self-improving robotic RL systems that bring the best of diverse robotic datasets and learning via online interaction.

Task. For this experiment, we consider the "open door" task from Scenario 2. Our goal is to improve the success rate of the learned policy obtained after PTR pre-training and offline fine-tuning using autonomous online rollouts from ten initial positions. These ten initial positions consist of five positions obtained by randomly sampling from the target demonstrations used for offline fine-tuning, and five more challenging **out-of-distribution initial positions**, that were never seen before.

Reward functions. To run RL, we need a mechanism to annotate every online rollout with a reward signal. Following prior works [49, 22], we trained a neural-network binary classifier to detect a given visual observation as a success (+1 reward) or failure (0 reward) and use it to annotate rollouts executed during online interaction.

Reset policy. To run online fine-tuning autonomously without any human intervention in the real world, we also need a "reset policy" that closes the door after a successful online rollout. To this end, we also pre-trained a close-door policy separately, which is used only for resetting the door. Note that online fine-tuning only fine-tunes the open-door policy, while

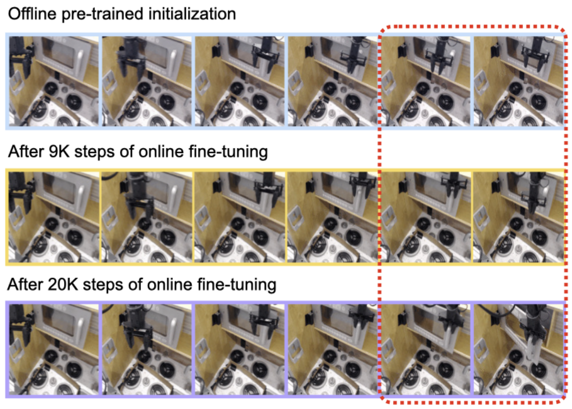
Offline pre-trained initialization
After 9K steps of online fine-tuning
After 20K steps of online fine-tuning

Fig. 9: **Evolution of learned behaviors during autonomous online fine-tuning of PTR starting from one of the hard initial positions.** The blue box illustrates that the offline initialization fails to grasp the handle. After 9K steps of online interaction, it successfully grasps the handle but fails to open the door. After 20K steps, it learns to successfully open the door.

the reset policy is kept fixed throughout.

Online training setup. Equipped with the reset policy and the reward classifier, we are able to run online fine-tuning in the real world. Starting from the pre-trained policy obtained via PTR, our method alternates between collecting a new trajectory and taking gradient steps. The update-to-data ratio [7] is set to 10, which means that we make 10 gradient updates for every environment step. More details about our implementation and evaluations can be found in Appendix F.

Results. We compare our method with a prior method that trains SAC [15] from scratch using both online data and offline demonstrations (denoted by "SACfD"). This approach is an improved version of DDPGfD [52] which uses a stronger off-policy RL algorithm (SAC). We present the learning curve during the online fine-tuning in Figure 8, and the success rates before and after fine-tuning in Table VI. As shown in Figure 8, it was difficult to run SACfD over a long time on the robot, as the system crashes due to unsafe actions during exploration (pictures shown in Appendix F). In contrast, the pre-trained PTR policy is able to perform online exploration in a stable manner, and improve the success rate of the pre-trained policy within 20K steps of online interaction. Specifically, this boost in performance stems from learning to solve the task from **3/5** of the more challenging, out-of-distribution initial positions, that were never seen before in the prior data, as shown in Figure 9. Overall, our results show the efficacy of PTR as a general-purpose pre-training paradigm for robotic RL.

VI. Discussion and Conclusion

We presented a system that uses diverse prior data for general-purpose offline RL pre-training, followed by fine-tuning to downstream tasks. The prior data, sourced from a publicly available dataset, consists of over a hundred tasks across ten scenes and our policies can be fine-tuned with as few as 10 demonstrations. We show that this approach outperforms prior pre-training and fine-tuning methods based on imitation learning. One of the most exciting directions for future work is to further scale up this pre-training to provide a single policy initialization, that can be utilized as a starting point, similar to GPT3 [5]. An exciting future direction is to scale PTR up to more complex settings, including to novel robots. Since joint training with offline RL was worse than pre-training and then fine-tuning with PTR, another exciting direction for future work is to understand the pros and cons of joint training and fine-tuning in the context of robot learning.

References

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

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[3] Aditya Bhatt, Max Argus, Artemij Amiranashvili, and Thomas Brox. CrossNorm: Normalization for Off-Policy TD Reinforcement Learning. *arXiv e-prints*, art. arXiv:1902.05605, February 2019.

[4] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning. *arXiv preprint arXiv:2106.01151*, 2021.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, et al. Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*, 2021.

[7] Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. *arXiv e-prints*, art. arXiv:2101.05982, January 2021. doi: 10.48550/arXiv.2101.05982.

[8] Ron Dorfman and Aviv Tamar. Offline meta reinforcement learning. *arXiv e-prints*, pages arXiv–2008, 2020.

[9] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021.

[10] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

[11] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[12] Ben Eysenbach, Sergey Levine, and Russ R Salakhutdinov. Replacing rewards with examples: Example-based policy search via recursive classification. *Advances in Neural Information Processing Systems*, 34:11541–11552, 2021.

[13] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[14] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

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

[16] K He, X Chen, S Xie, Y Li, P Dollár, and RB Girshick. Masked autoencoders are scalable vision learners. arxiv. 2021 doi: 10.48550. *arXiv preprint arXiv.2111.06377*, 2021.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

[19] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[20] Ryan Julian, Benjamin Swanson, Gaurav S Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. *arXiv preprint arXiv:2004.10190*, 2020.

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

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

[23] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint*

arXiv:2004.13649, 2020.

[24] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pages 5774–5783. PMLR, 2021.

[25] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. 2021.

[26] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[27] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[28] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=AP1MKT37rJ.

[29] Alex X Lee, Coline Devin, Jost Tobias Springenberg, Yuxiang Zhou, Thomas Lampe, Abbas Abdolmaleki, and Konstantinos Bousmalis. How to spend your robot time: Bridging kickstarting and offline reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:2205.03353*, 2022.

[30] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

[31] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pages 1702–1712. PMLR, 2022.

[32] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[33] Jiachen Li, Quan Vuong, Shuang Liu, Minghua Liu, Kamil Ciosek, Keith Ross, Henrik Iskov Christensen, and Hao Su. Multi-task batch reinforcement learning with metric learning. *arXiv preprint arXiv:1909.11373*, 2019.

[34] Sen Lin, Jialin Wan, Tengyu Xu, Yingbin Liang, and Junshan Zhang. Model-based offline meta-reinforcement learning with regularization. *arXiv preprint arXiv:2202.02929*, 2022.

[35] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

[36] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, 2020.

[37] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Fei-Fei Li, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=JrsfBJtDFdI.

[38] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline Meta-Reinforcement Learning with Advantage Weighting. *arXiv e-prints*, art. arXiv:2008.06043, August 2020.

[39] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline meta-reinforcement learning with advantage weighting. In *International Conference on Machine Learning*, pages 7780–7791. PMLR, 2021.

[40] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[41] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[42] Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-QL: Calibrated offline rl pre-training for efficient online fine-tuning. *arXiv preprint arXiv:2303.05479*, 2023.

[43] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[44] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[45] Vitchyr H Pong, Ashvin Nair, Laura Smith, Catherine Huang, and Sergey Levine. Offline meta-reinforcement learning with online self-supervision. *arXiv preprint arXiv:2107.03974*, 2021.

[46] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. Real-world robot learning with masked visual pre-training. *arXiv preprint arXiv:2210.03109*, 2022.

[47] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *arXiv preprint arXiv:2206.11251*, 2022.

[48] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint*

arXiv:2002.08396, 2020.

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

[50] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

[51] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

[52] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[53] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, 2007.

[54] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[55] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[56] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022.

[57] Annie Xie and Chelsea Finn. Lifelong robotic reinforcement learning by retaining experiences. *arXiv preprint arXiv:2109.09180*, 2021.

[58] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. *arXiv preprint arXiv:2102.05815*, 2021.

[59] Mengjiao Yang, Sergey Levine, and Ofir Nachum. Trail: Near-optimal imitation learning with suboptimal data. *arXiv preprint arXiv:2110.14770*, 2021.

[60] Sarah Young, Dhiraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. Visual imitation made easy, 2020.

[61] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. *NeurIPS*, 34, 2021.

[62] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Chelsea Finn, and Sergey Levine. How to leverage unlabeled data in offline rl. *arXiv:2202.01741*, 2022.

Appendix

A. Diagnostic study in simulation

We perform a diagnostic study in simulation to verify some of the insights observed in our real-world experiments. We created a bin sort task, where a WidowX250 robot is placed in front two bins and is provided with two objects (more details in Appendix B). The task is to sort each object in the correct bin associated with that object. The pre-training data provided to this robot is pick-place data, which only demonstrates how to pick *one* of the objects and place it in one of the bins, but does not demonstrate the compound task of placing both objects. In order to succeed at this such a compound task, a robot must learn an abstract representation of the skill of sorting an object during the pre-training phase and then figure out that it needs to apply this skill multiple times in a trajectory to succeed at the task from just *five* demonstrations of the desired sorting behavior.

The performance numbers (along with 95%-confidence intervals) are shown in Table VII. Observe that PTR improves upon prior methods in a statistically significant manner, outperforming the BC and COG baselines by a significant margin. This validates the efficacy of PTR in simulation and corroborates our real-world results.

Method	Success rate
BC (joint training)	7.00 ± 0.00 %
COG (joint training)	8.00 ± 1.00 %
BC (finetune)	4.88 ± 4.07 %
PTR (Ours)	$\mathbf{17.41 \pm 1.77}$ **%**

TABLE VII: **Performance of PTR in comparison with other methods** on the simulated bin sorting task, trained for many more gradient steps for all methods until each one of them converges. Observe that PTR substantially outperforms other prior methods, including joint training on the same data with BC or CQL. Training on target data only is unable to recover a non-zero performance, so we do not report it in this table. Since the 95%-confidence intervals do not overlap between PTR and other methods, it indicates that PTR improves upon baselines in a statistically significant manner.

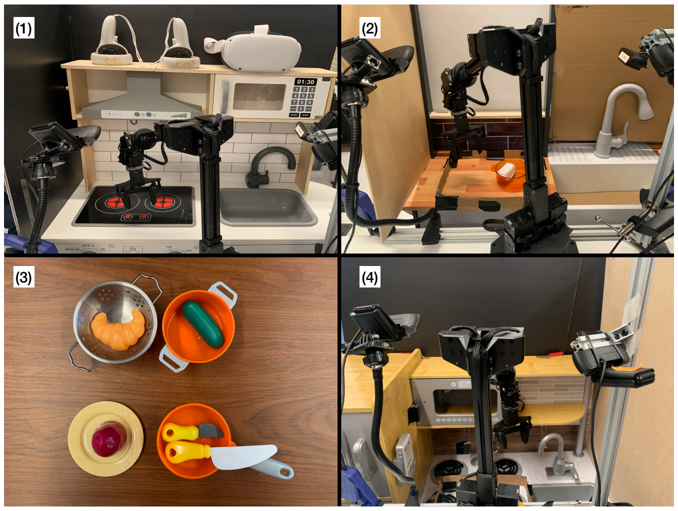

Fig. 10: **Setup Overview**: Following Ebert et al. [9], we use a toykitchen setup described in that prior work for our experiments. This utilizes a 6-DoF WidowX 250 robot. **(1):** Held-out toykitchen used for experiments in Scenario 3 (denoted "toykitchen 6"), **(2):** Re-targeting toykitchen used for experiments in Scenario 2 (denoted "toykitchen 2"), **(3):** target objects used in the experiments of scenario 3., **(4):** the held-out kitchen setup used for door opening ("toykitchen 1").

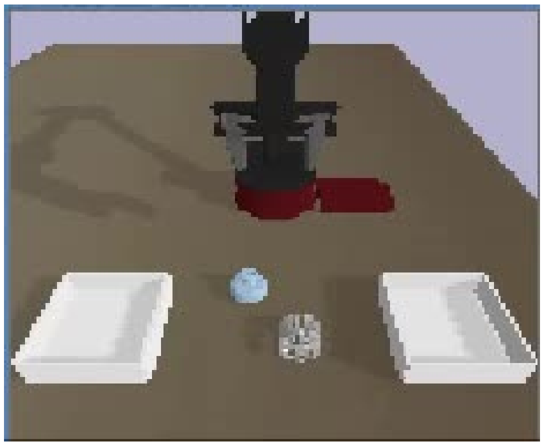

Fig. 11: **Bin-Sorting task used for our simulated evaluations.** The task requires sorting the cylinder into the left bin and the teapot into the right bin.

B. Details of Our Experimental Setup

1) Real-World Experimental Setup

A picture of our real-world experimental setup is shown in Figure 10. The scenarios considered in our experiments (Section V) are designed to evaluate the performance of our method under a variety of situations and therefore we set up these tasks in different toykitchen domains (see Figure 10) on three different WidowX 250 robot arms. We use data from the bridge dataset [9] consisting of data collected with many robots in many domains for training but exclude the task/domain that we use for evaluation from the training dataset.

2) Diagnostic Experimental Setup in Simulation

We evaluate our approach in a simulated bin-sorting task on the simulated WidowX 250 platform, aimed to mimic the setup we use for our real-world evaluations. This setup is designed in the PyBullet simulation framework provided by Singh et al. [50]. A picture is shown in Figure 11. In this task, two different bins and two different objects are placed in front of the WidowX robot. The goal of the robot is to correctly sort each of the two objects to their designated bin (e.g the cylinder is supposed to be placed in the left bin and the teapot should be placed in the right bin. We refer to this task as a *compound* task since it requires successfully combining behaviors of two different pick-and-place skills one after the other in a single trajectory while also adequately identifying the correct bin associated with each object. Success is counted only when the robot can accurately sort *both* of the objects into their corresponding bins.

Offline pre-training dataset. The dataset provided for offline pre-training only consists of demonstrations that show how the robot should pick one of the two objects and place it into one of the two bins. Each episode in the pre-training dataset is about 30-40 timesteps long. A picture showing some trajectories from the pre-training dataset is shown in Figure 12. While the downstream task only requires solving this sorting task with two specific objects (shown in Figure 13), the pre-training data consists of 10 unique objects (some shown in Figure 12). The two target objects that appear together in the downstream target scene are never seen together in the pre-training data. Since the pre-training data only demonstrates how the robot must pick up one of the objects and place it in one of the two bins (not necessarily in the target bin that the target task requires), it neither consists of any behavior that places objects into bins sequentially nor does it consist of any behavior where one of the objects is placed one of the bins while the other one is not. This is what makes this task particularly challenging.

Target demonstration data. The target task data provided to the algorithm consists of only ***five*** demonstrations that show how the robot must complete both the stages of placing both objects (see Figure 13). Each episode in the target demonstration data is 80 timesteps long, which is substantially longer than any trajectory in the pre-training data, though one would hope that good representations learned from the pick and place tasks are still useful for this target task. While all methods are able to generally solve the first segment of placing the first object into the correct bin, the primary challenge in this task is to effectively sort the second object, and we find that PTR attains a substantially better success rate than other baselines in this exact step.

C. Description of the Real-World Evaluation Scenarios

In this section, we describe the real-world evaluation scenarios considered in Section V. We additionally include a much more challenging version of Scenario 3, for which we present results in Appendix D. These harder test cases evaluate the fine-tuning performance on four different tasks, starting from the same initialization trained on bridge data except the toykitchen 6 domain in which these four tasks were set up. In the following sections, the nomenclature for the toy kitchens is drawn from Ebert et al. [9] and as described in the caption of Figure 10.

1) Scenario 1: Re-targeting skills for existing to solve new tasks

Pre-training data. The pre-training data comprises all of the pick and place data from the bridge dataset [9] from toykitchen 2. This includes data corresponding to the task of putting the sushi in the transparent orange pot (Figure 14).

Target task and data. Since our goal in this scenario is to re-target the skill for putting the sushi in the transparent orange pot to the task of putting the sushi in the metallic pot, we utilize a dataset of 20 demonstrations that place the sushi in a metallic pot as our target task data that we fine-tune with (shown in Figure 14).

Quantitative evaluation protocol. For our quantitative evaluations in Table I, we run 10 controlled evaluation rollouts that place the sushi and the metallic pot in different locations of the workspace. In all runs, the arm starts at up to 10 cm distance above the target object. The initial object and arm poses and positions are matched as closely as possible for different methods.

2) Scenario 2: Generalizing to Previously Unseen Domains

Pre-training data. The pre-training data in Scenario 2 consists of 800 door-opening demonstrations on 12 different doors across 3 different toykitchen domains.

Target task and data. The target task requires opening the door of an unseen microwave in toykitchen 1 using a target dataset of only 15 demonstrations.

Quantitative evaluation protocol. We run 20 rollouts with each method, counting successes when the robot opened the door by at least 45 degrees. To perform this successfully, there is a degree of complexity as the robot has to initially open the door till it's open to about 30 degrees. Then due to physical constraints, the robot needs to wrap around the door and push it open from the inside. To begin an evaluation rollout, we reset the robot to randomly sampled poses obtained from held-out demonstrations on the target door. This is a compound task requiring the robot to first grab the door by the handle, next move around the door, and finally push the door open. As before, we match the initial pose of the robot as closely as possible for all the methods.

3) Scenario 3: Learning to Solve New Tasks in New Domains

Pre-training data. All pick-and-place data in the bridge dataset [9] except any demonstration data collected in toykitchen 6, where our evaluations are performed.

Target task and data. The target task requires placing corn in a pot in the sink in the new target domain and the target dataset provides 10 demonstrations for this task. These target demonstrations are sampled from the bridge dataset itself.

Quantitative evaluation protocol. During the evaluation we were unable to exactly match the camera orientation used to collect the target demonstration trajectories, and therefore ran evaluations with a slightly modified camera view. This presents an additional challenge for any method as it must now generalize to a modified camera view of the target toykitchen domain, without having ever observed this domain or this camera view during training. We sampled initial poses for our method by choosing transitions from a held-out dataset of demonstrations of the target task and resetting the robot to those initial poses for each method. We attempted to match the positions of objects across methods as closely as possible.

4) More Tasks in Scenario 3: Learning to Solve Multiple New Tasks in New Domains From the Same Initialization

In Appendix D, we have now added results for more tasks in Scenario 3. The details of these tasks are as follows:

Pre-training data. All pick-and-place data from bridge dataset [9] except data from toykitchen 6.

Target task and data. We consider four downstream tasks: take croissant from a metallic bowl, put sweet potato on a plate, place the knife in a pot, and put cucumber in a bowl. We collected 10 target demonstrations for the croissant, sweet potato, and put cucumber in bowl tasks, and 20 target demonstrations for the knife in pot task. A picture of these target tasks is shown in Figure 16.

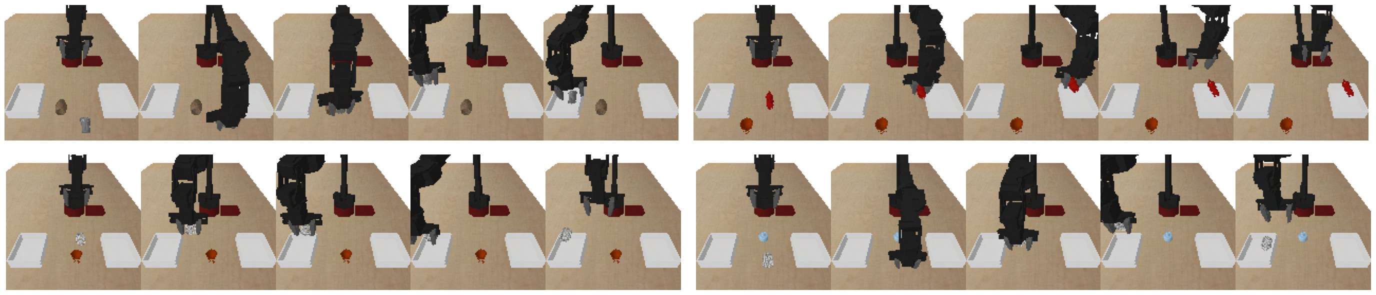
Fig. 12: Some trajectories from the pre-training data used in the simulated bin-sort task.

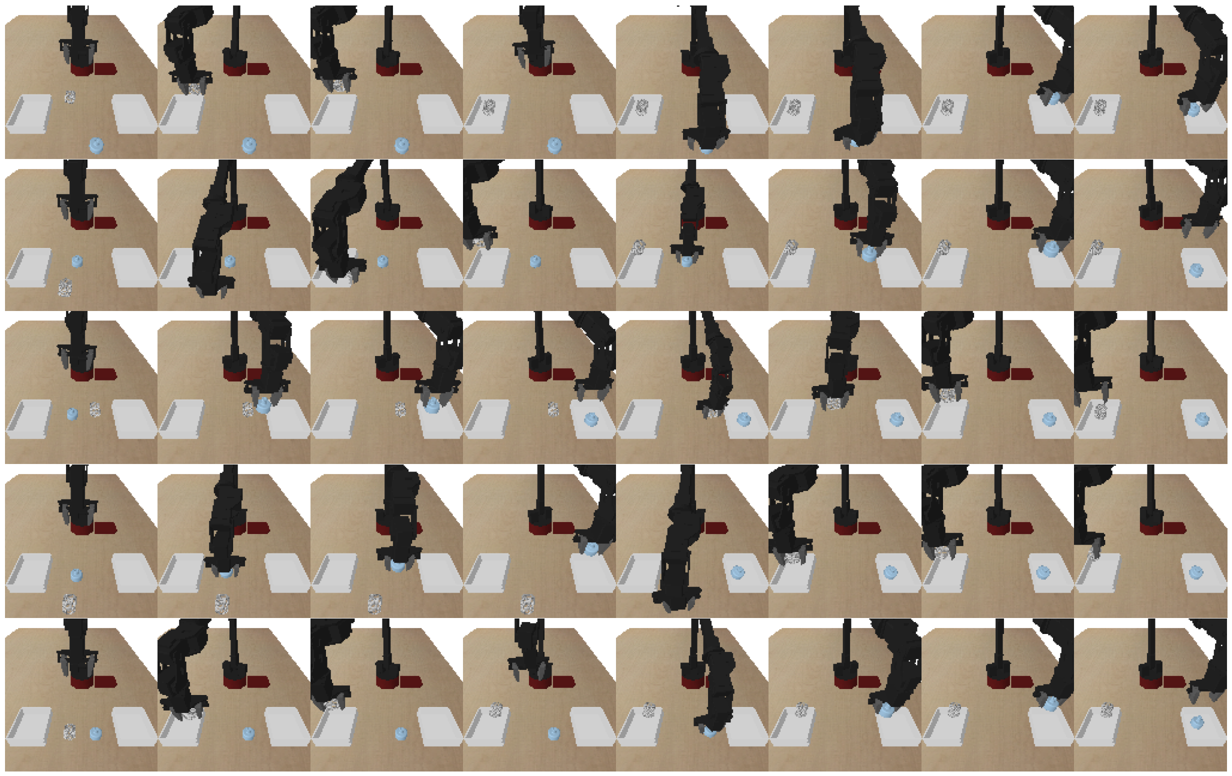
Fig. 13: The five demonstration trajectories used for Phase 2 of PTR.

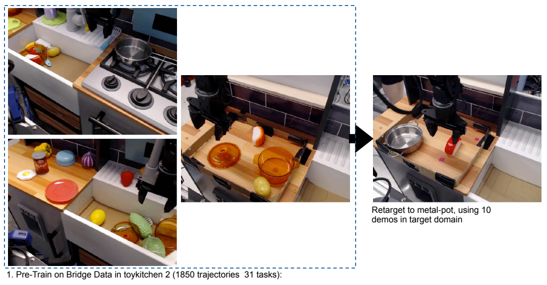
Retarget to metal-pot, using 10 demos in target domain
1. Pre-Train on Bridge Data in toykitchen 2 (1850 trajectories 31 tasks):

Fig. 14: **Illustration of pre-training data and finetuning data used for Scenario 1**: re-targeting the put sushi in metal-pot behavior to put the object in the metal pot instead of the orange transparent pot.

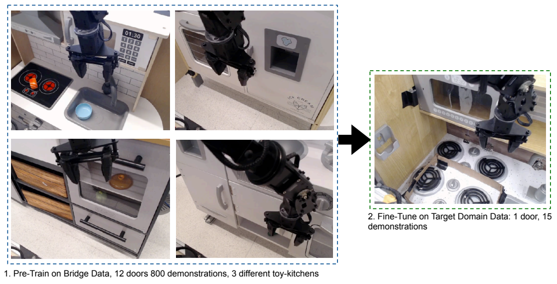
2. Fine-Tune on Target Domain Data: 1 door, 15 demonstrations
1. Pre-Train on Bridge Data, 12 doors 800 demonstrations, 3 different toy-kitchens

Fig. 15: **Illustration of pre-training data and fine-tuning data used for Scenario 2 (door opening)**: transferring a behavior to a held-out domain.

Qualitative evaluation protocol. For our evaluations, we utilize either 10 or 20 evaluation rollouts. As with all of our other quantitative results, we evaluate all the baseline approaches and PTR starting from an identical set of initial poses for the robot. These initial poses are randomly sampled from the poses that appear in the first 10 timesteps of the held-out demonstration trajectories for this target task. For the configuration of objects, we test our policies in a variety of

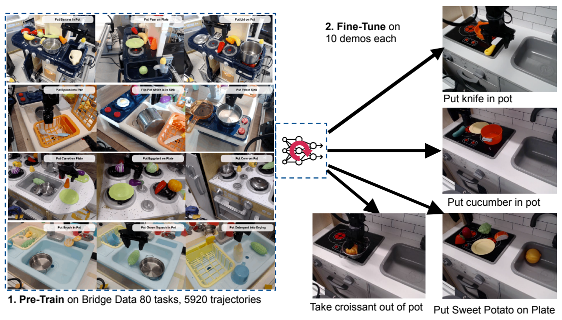
2. Fine-Tune on 10 demos each
Put knife in pot
Put cucumber in pot
Take croissant out of pot
Put Sweet Potato on Plate
1. Pre-Train on Bridge Data 80 tasks, 5920 trajectories

Fig. 16: **Illustration of pre-training data and fine-tuning data used for the new tasks we have added in Scenario 3**. The goal is to learn to solve new tasks in new domains starting from the same pre-trained initialization and when fine-tuning is only performed using 10-20 demonstrations of the target task.

task-specific configurations that we discuss below:

- **Take croissant from metallic bowl:** For this task, we alternate between two kinds of positions for the metallic bowl. In the "easy" positions, the metallic bowl is placed roughly vertically beneath the robot's initial starting pose, whereas in the "hard" positions, the robot must first move itself to the right location of the bowl and then execute the policy.
- **Put the cucumber in bowl:** We run 10 evaluation rollouts starting from 10 randomly sampled initial poses of the robot for our evaluations. Here we moved the bowl between the two stovetops in each trial.
- **Put sweet potato on plate:** For this task, we performed 20 evaluation rollouts. We only sampled 10 initial poses for the robot, but for each position, we evaluated every policy on two orientations of the sweet potato (i.e., the sweet potato is placed on the table on its flat face or on its curved face). Each of these orientations presents some unique challenges, and evaluating both of them allows us to gauge how robust the learned policy is to changes in orientation. The demonstration data had a variety of orientations for the sweet potato object that differed for each collected trajectory.
- **Place knife in pot:** We evaluate this task over 10 evaluation rollouts, where the first five rollouts use a smaller knife, while the other five rollouts use a larger knife (shown in Figure 10). Each knife was seen in the demonstration dataset with equal probability.

We will discuss the results obtained on these new tasks in Appendix D.

D. Additional Experimental Results

Finetuning to novel camera viewpoints: Even though Scenario 3 already presents a novel toy-kitchen domain and previously unseen objects during finetuning, we also evaluate PTR on a more challenging scenario where we additionally alter the camera viewpoint during finetuning. We apply two kinds of alterations to the camera: **(a)** we elevate the mounting platform of the camera by 7 cm, which necessitates adapting the way the physical coordinates of the robot end-effector are interpreted by the policy, and **(b)** we rotate the camera by about 15 degrees to induce a more oblique image observation than what was ever seen during pre-training. Note that in both of these scenarios, the robot has never encountered such camera viewpoints during pre-training, which makes this scenario even more challenging. The original dataset in [9] had the camera elevated to the same position for each domain and always ensured the kitchen was parallel to the camera platform, with translations being the primary changes in the scene for each domain. In Table VIII, we present our results comparing PTR and BC (finetune). Observe that PTR still clearly outperforms BC (finetune), and attains performance close to that of PTR in Table III, indicating that such shifts in the camera do not drastically hurt PTR.

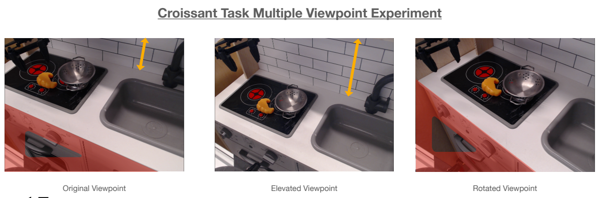

Fig. 17: **Sample observations from different camera viewpoints, only used during fine-tuning**. **Left:** the original camera viewpoint found in Figure 16. **Middle:** an elevated camera viewpoint where the robot and camera have been raised 7 cm. **Right:** a rotated camera viewpoint where the kitchen has been slightly translated and rotated 15 degrees counterclockwise relative to the camera and robot.

Method	Elevated Viewpoint	Rotated Viewpoint
BC (finetune)	2/10	3/10
PTR (Ours)	**6/10**	**7/10**

TABLE VIII: **Comparison of PTR and BC (finetune), when evaluated on novel camera viewpoints** with elevated and rotated cameras as shown in Figure 17 for the croissant task. Observe that PTR still outperforms BC (finetune) in this setting and attains more than 2x success rate of BC (finetune).

1) *Expanded Discussion: Why Does PTR Outperform BC-based methods, Even With Demonstration Data?*

One natural question to ask given the results in this paper is: why does utilizing an offline RL method for pre-training and finetuning as in PTR outperform BC-based methods even though the dataset is quite "BC-friendly", consisting of only demonstrations? One might speculate that an answer to this question is that our BC baseline can be tuned to be much better. However, note that our BC baseline is not suboptimally tuned. We utilize the procedure prescribed by prior work [9] for tuning BC as we discuss in Appendix D2. In addition, the fact that **BC (joint)** does actually outperform **CQL (joint)** in many of our experiments, indicates that our BC baselines are well-tuned. To explain the contrast to Ebert et al. [9], note that the setup in this prior work utilized many more target task demonstrations ($\geq$ 50 demonstrations from the target task) compared to our evaluations, which might explain why our BC-baseline numbers are lower in an absolute sense. Therefore, the technical question still remains: why would we expect PTR to perform better than BC? We will attempt to answer this question using some empirical evidence and visualizations. Also, we will aim to provide intuition for why our approach PTR outperforms the baseline.

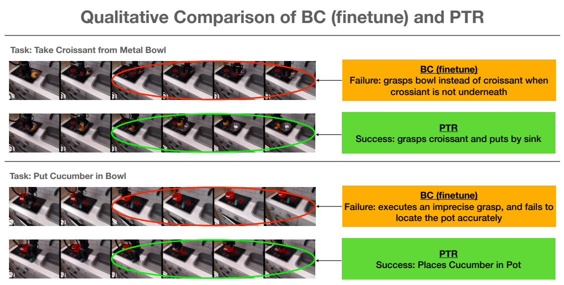
Qualitative Comparison of BC (finetune) and PTR
Task: Take Croissant from Metal Bowl
BC (finetune)
Failure: grasps bowl instead of croissant when crossiant is not underneath
PTR
Success: grasps croissant and puts by sink
Task: Put Cucumber in Bowl
BC (finetune)
Failure: executes an imprecise grasp, and fails to locate the pot accurately
PTR
Success: Places Cucumber in Pot

Fig. 18: **Qualitative successes of PTR visualized alongside failures of BC (finetune).** As an example, observe that while PTR is accurately able to reach to the croissant and grasp it to solve the task, BC (finetune) is imprecise and grasps the bowl instead of the croissant resulting in failure.

To begin answering this question, it is instructive to visualize some failures for a BC-based method and qualitatively attempt to understand why BC is worse than utilizing PTR. We visualize some evaluation rollouts for **BC (finetune)** and PTR as film strips in Figure 18. Specifically, we visualize evaluation rollouts that present a challenging initial state. For example, for the rollout from the take croissant out of metallic pot task, the robot must first accurately position itself over the croissant before executing the grasping action. Similarly, for the rollout from the cucumber task, the robot must accurately locate the bowl and precisely try to grasp the cucumber. Observe in Figure 6 that **BC (finetune)** typically fails to accurately reach the objects of interest (croissant and the bowl) and executes the grasping action prematurely. On the other hand, PTR is more robust in these situations and is able to accurately reach the object of interest before it executes the grasping action or the releasing action. Why does this happen?

To understand why this happens, one mental model is to appeal to the critical states argument from Kumar et al. [28]. Intuitively, this argument suggests that in tasks where the robot must precisely accomplish actions at only a few specific states (called "**critical states**") to succeed, but the actions at other states (called "non-critical states") do not matter as much. Thus, offline RL-style methods can outperform BC-based methods even with demonstration data. This is because learning a value function can enable the robot to reason about which states are more important than others, and the resulting policy optimization can "focus" on taking correct actions at such critical states. Our real-world evaluation scenarios exhibit such a structure. The majority of the actions that the robot must take to reach the object do not need to be precise as long as they generally move the robot in the right direction. However, in order to succeed, the robot must critically ensure to position the arm is right above the object in a correct orientation and position itself right above the container in which the object must be placed. These are the critical states and special care must be taken to execute the right action in these states. In such scenarios, the argument of Kumar et al. [28] would suggest that offline RL should be better. We believe that we observe a similar effect in our experiments: the learned BC policies are often not precise-enough at those critical states where taking the right action is critical to success.

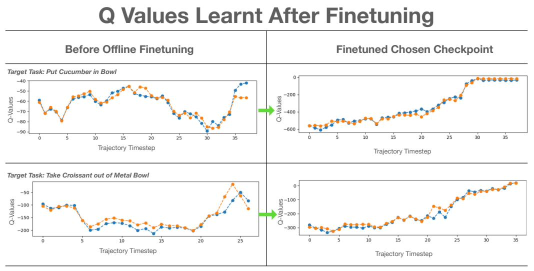

Fig. 19: **Evolution of Q-values on the target task** over the process of fine-tuning with PTR. Observe that while the learned Q-values on *held-out* trajectories from the dataset just at the beginning of Phase 2 (finetuning) do not exhibit a roughly increasing trend, the checkpoint of PTR we choose to evaluate exhibits a generally increasing trend in the Q-values despite having access to only 10 demonstrations for these target tasks.

As supporting evidence to the discussion above, we further visualize the Q-values over held-out trajectories from the target demonstration data that were never seen by PTR during fine-tuning in Figure 19. To demonstrate the contrast, we present the trend in Q-values before fine-tuning and for the checkpoint selected for evaluation after fine-tuning on the target task. Observe that the Q-values for the chosen checkpoint generally increase over the course of the trajectory indicating that the learned Q-function is able to fit well with the target data. Also, the learned Q-function generalizes to held-out trajectories despite the fact that only 10 demonstrations were provided during the fine-tuning phase. This evidence supports the claim that it is reasonable to expect the learned Q-function to be able to focus on the more critical decisions in the trajectory.

To further support our hypothesis that PTR outperforms BC-based methods because the learned value function enables us to learn about "critical" decisions, we run an experiment that essentially runs a weighted version of BC during finetuning, where the weights are provided by exponentiated advantage values, where the advantages are defined as $A_\theta(\mathbf{s}, \mathbf{a}) = Q_\theta(\mathbf{s}, \mathbf{a}) - \max_{\mathbf{a}'} Q_\theta(\mathbf{s}, \mathbf{a}')$ under a Q-function learned by PTR. This approaches essentially matches BC finetuning in all aspects: the policy parameterization, the loss function (mean-squared error), and the details of the training are kept identical to our BC baselines, with the exception of an additional weight given by $\exp(A_\theta(\mathbf{s}, \mathbf{a}))$ on a given transition $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ observed in the set of limited task-specific demonstrations. We refer to this approach as "advantage-weighted BC finetuning".

In contrast to our BC (finetune) results from Table III, where PTR significantly outperformed BC (finetune), observe in Table IX, that advantage-weighted BC (finetune) performs comparably to PTR on the two tasks we studied for our analysis. This result is significant since it implies that all other factors kept identical, utilizing the weights given by the Q-function is the crucial factor in improving the performance of BC and avoids the qualitative failure modes associated with BC methods shown in Figure 18.

2) *Hyperparameters for PTR and Baseline Methods*

In this section, we will present the hyperparameters we use in our experiments and explain how we tune the other

Task	BC (finetune)	PTR (Ours)	Advantage-weighted BC (finetune)
Put cucumber in pot	0/10	5/10	5/10
Take croissant from metal bowl	3/10	7/10	6/10

TABLE IX: **Performance of advantage-weighted BC** on two tasks from Table III. Observe that weighting the BC objective using advantage-weights computed using the Q-function learned by PTR leads to much better performance than standard BC (finetune), and close to PTR. This test indicates that the Q-function in PTR allows us to focus on more critical points, thereby preventing the failures discussed in Figure 18.

hyperparameters for both our method PTR and the baselines we consider.

PTR. Since PTR utilizes CQL as the base offline RL method, it trains two Q-functions and a separate policy, and maintains a delayed copy of the two Q-functions, commonly referred to as target Q-functions. We utilize completely independent networks to represent each of these five models (2 Q-functions, 2 target Q-functions, and the policy). We also do not share the convolutional encoders among them. As discussed in the main text, we rescaled the action space to $[-1, 1]^{|\mathcal{A}|}$ to match the one used by actor-critic algorithms, and utilized a Tanh squashing function at the end of the policy. We used a CQL α value of 10.0 for our pick-and-place experiments. The rest of the hyperparameters for training the Q-function, the target network updates, and the policy are taken from the standard training for image-based CQL from Singh et al. [50] and are presented in Table XI below for completeness. The hyperparameters we choose are essentially the network design decisions of **(1)** utilizing group normalization instead of batch normalization, **(2)** utilizing learned spatial embeddings instead of standard mean pooling, **(3)** passing in actions at each of the fully connected layers of the Q-network and the hyperparameter α in CQL that must be adjusted since our data consists of demonstrations. We will ablate the new design decisions explicitly in Appendix E.

The only other hyperparameter used by PTR is the mixing ratio τ that determines the proportion of samples drawn from the pre-training dataset and the target dataset during the offline finetuning phase in PTR. We utilize $\tau = 0.7$ for our experiments with PTR in the main paper, and use $\tau = 0.9$ for the additional experiments we added in the Appendix. This is because $\tau = 0.9$ (more bridge data, and a smaller amount of target data) was helpful in scenarios with very limited target data.

In order to perform checkpoint selection for PTR, we utilized the trends in the learned Q-values over a set of held-out trajectories on the target data as discussed in Section IV-B. We did not tune any other algorithmic hyperparameters for CQL, as these were taken directly from [50].

BC (finetune). We trained BC in a similar manner as Ebert et al. [9], utilizing the design decisions that this prior work found optimal for their experiments. The policy for BC utilizes the very same ResNet 34 backbone as our RL policy since a backbone based on ResNet 34 was found to be quite effective in Ebert et al. [9]. Following the recommendations of Ebert et al. [9] and based on result trends from our own preliminary experiments, we chose to not utilize the tanh squashing function at the end of the policy for any BC-based method, but trained a deterministic BC policy that was trained to regress to the action in the demonstration with a mean-squared error (MSE) objective.

In order to perform cross-validation, checkpoint, and model selection for our BC policies, we follow guidelines from prior work [9, 10] and track the MSE on a held-out validation dataset similar to standard supervised learning. We found that a ResNet 34 BC policy attained the smallest validation MSEs in general, and for our evaluations, we utilized a checkpoint of a ResNet 34 BC policy that attained the smallest MSE.

Analogous to the case of PTR discussed above, we also ablated the performance of BC for a set of varying values of the mixing ratio τ, but found that a large value of $\tau = 0.9$ was the most effective for BC, and hence utilized $\tau = 0.9$ for BC (finetune) and BC (joint).

BC (joint) and CQL (joint). The primary distinction between training **BC (joint)** and **BC (finetune)** and correspondingly, **CQL (joint)** and PTR was that in the case of joint training, the target dataset was introduced right at the beginning of Phase 1 (pre-training phase), and we mixed the target data with the pre-training data using the same value of the mixing ratio τ used in for our fine-tuning experiments to ensure a fair comparison.

Few-shot offline meta-RL (MACAW) [38]: We compare to two variants of this algorithm and perform an **extensive** sweep over several hyperparameters, shown in Table XII.

We trained two different variants of MACAW in our evaluation: **(1)** Pre-training on the bridge data in Scenario 3 and then fine-tuning on target data of interest, and **(2)** adapting a set of existing task identifiers to the target task of interest utilizing the same pre-training and fine-tuning domains. We performed early stopping on the meta-training based on validation losses. From there, we started the meta-testing phase, adapting to the target domain of interest. Following Mitchell et al. [38], we use a task mini-batch of 8 tasks at each step of optimization rather than using all of the training tasks. We clipped the advantage weight logits to the scale of 20 and attempted to utilize a policy network with a fixed and learned standard deviation. Additionally, we varied the number of Adaptation steps following prior work. Our evaluation protocol for MACAW entails utilizing the validation losses to choose an initial checkpoint for evaluation. Then, we consider checkpoints in the neighborhood ($\pm$ 50K gradient steps) to for evaluations as well and chose the max over all of these checkpoints as the final evaluation success rate.

Quantitatively, as seen in Table III, MACAW was unable to get non-zero success rates on any of the tasks we study. However, we did qualitatively observe nontrivial behavior seen in our evaluation rollouts. For instance, we found that the policies trained via MACAW could consistently grasp the object of interest but were unable to localize where to place the object correctly. Several trials involved hovering around with the object of interest and not placing the object in the container. Other trials involved the agent failing to grasp the

Hyperparameter	Value
Q-function learning rate	3e-4
Policy learning rate	1e-4
Target update rate	0.005 (soft update with Polyak averaging)
Optimizer type	Adam
Discount factor γ	0.96 (since trajectories have a length of only about 30-40)
Use terminals	True
Reward shift and scale	shift = -1, scale = 10.0
CQL α	10.0
Use Color Jitter	True
Use Random Cropping	True

TABLE X: **Main hyperparameters for CQL training in our real-world experiments.** In the simulation, we utilize a smaller α for CQL, $\alpha = 1.0$, and a larger discount $\gamma = 0.98$ since trajectories in the simulation are about 60-70 timesteps in length.

Hyperparameter	Value
Policy learning rate	1e-4
Optimizer type	Adam
Use Color Jitter	True
Use Random Cropping	True
Dropout	0.4

TABLE XI: **Main hyperparameters for Behavior Cloning Baseline Training in our real-world and simulation experiments.** Note: architecture design choices follow closely to PTR design choices.

object.

Hyperparameter	Value
Optimizer	Adam
Outer Policy learning rate	1e-4
Outer Value learning rate	1e-5, 1e-6
Inner Policy learning rate	1e-2, 1e-3
Inner Value learning rate	1e-3, 1e-4
Auxilary Advantage Coefficient	1e-2, 1e-3, 1e-4
Policy Parameterization	Fixed std, Learned std
AWR Policy Temperature	1, 10, 20
Number of Adaptation Steps	1, 2, 3
Task Batch Size	8
Train Adaptation Batch Size	64
Eval Adaptation Batch Size	64
Max Advantage Clip	20
Use Color Jitter	True
Use Random Cropping	True

TABLE XII: **Main hyperparameters for Training MACAW [38] in our real-world experiments.** Note: architecture design choices follow closely to PTR design choices but hyperparameter design choices follow closely the suggestions in Mitchell et al. [38].

Pre-trained R3M initialization [41]: Next we compare PTR to utilizing an off-the-shelf pre-trained representation given by R3M [41]. We compare two baselines that attempt to train an MLP policy on top of the R3M state representation by using BC (finetuning) and CQL (finetuning) respectively. To ensure that this baseline is well-tuned, we tried a variety of network sizes with 2, 3 or 4 MLP layers and also tuned the hidden dimension sizes in [256, 512, 1024]. We also utilized dropout as regularization to prevent overfitting and tuned a variety of values of dropout probability in [0, 0.1, 0.2, 0.4, 0.6, 0.8]. We observe in Table III, that on the four tasks we evaluate on, PTR outperforms R3M, which indicates that training on the bridge dataset can indeed give rise to effective visual representations that are more suited to finetuning in our setting. The numbers we report in the table are the best over each parametric policy corresponding to each hyperparameter in our ablation. Checkpoint selection was done utilizing early stopping which is the last iteration where the validation error stops decreasing. Learning curves for this baseline can be found in our Anonymous Website.

Pre-trained MAE initialization [16]: We took a similar training procedure to R3M for our MAE representation. We used an MAE trained on every image from the bridge dataset Ebert et al. [9]. We then fine-tuned a specific target task with a similar ablation on network size, hidden dimension size, and regularization techniques such as dropout. We observe in Table IV, that on the four tasks we evaluate on, PTR outperforms R3M, which indicates that training on the bridge dataset can indeed give rise to effective visual representations that are more suited to finetuning in our setting. The numbers we report in the table are the best over each parametric policy corresponding to each hyperparameter in our ablation. Checkpoint selection was done utilizing early stopping which is the last iteration where the validation error stops decreasing.

Policy expressiveness study. We considered two policy expressiveness choices for BC to compare with our reference BC implementation that is implemented with a set of MLP layers. The first of the two choices was an **autoregressive policy** where the 7-dimensional action space was discretized into 100 bins. Each action was then predicted autoregressively conditioned on the observation, task id, and the action component from the previous dimension(s). The second approach was with the BeT Architecture from Shafiullah et al. [47]. We utilized the reference implementation from the paper with the default suggested hyperparameters for this set of ablations. The window size for the MinGPT transformer was ablated over between 1, 2, and 10.

E. Validating the Design Choices from Section IV-B via Ablation Studies

In this section, we will present ablation studies aimed to validate the design choices utilized by PTR. We found these design choices quite crucial for attaining good performance. The concrete research questions we wish to answer are: **(1)** How important is utilizing a large network for attaining good performance with PTR, and how does the performance of PTR

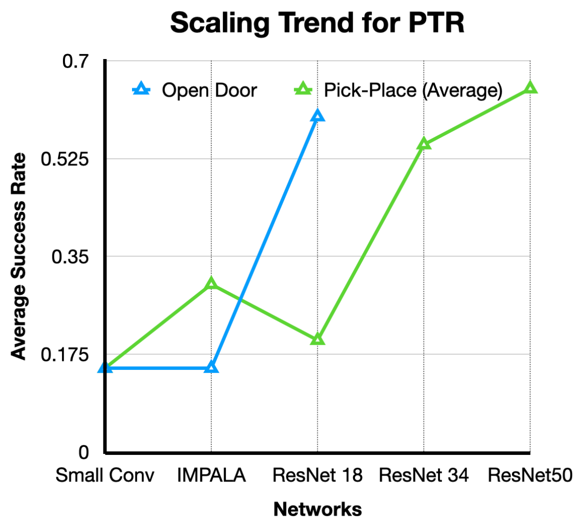

Fig. 20: **Scaling trends for PTR** on the open door task from Scenario 2, and average over two pick and place tasks (take croissant out of the metallic pot and put cucumber in the bowl) from Scenario 3. Note that more high capacity and expressive function approximators lead to the best results.

scale with the size of the Q-function?, **(2)** How effective is a learned spatial embedding compared to other approaches for aggregating spatial information? **(3)** Is concatenating actions at each fully-connected layer of the Q-function crucial for good performance?, **(4)** Is group normalization a good alternative to batch normalization? and **(5)** How does our choice of creating binary rewards for training affect the performance of PTR?. We will answer these questions next.

Highly expressive Q-networks are essential for good performance. To assess the importance of highly expressive Q-functions, we evaluate the performance of PTR with varying sizes and architectures on three tasks: the open door task from Scenario 2, and the put cucumber in the pot and take croissant out of metallic bowl tasks from Scenario 3. Our choice of architectures is as follows: **(a)** a standard three-layer convolutional network typically used by prior work for DM-control tasks (see for example, Kostrikov et al. [24]), **(b)** an IMPALA [11] ResNet that consists of 15 convolutional layers spread across a stack of 3 residual blocks, **(c)** ResNet 18 with group normalization and learned spatial embeddings, **(d)** ResNet 34 that we use in our experiments, and **(e)** an even bigger ResNet 50 with group normalization and learned spatial embeddings.

We present our results in Figure 7. To obtain more accurate scaling trends, we plot the trend in the average success rates for the pick and place tasks from Scenario 3 along with the trend in the success rate for the open door task separately since these tasks use different pre-training datasets. Observe that the performance of smaller networks (Small, IMPALA) is significantly worse than the ResNet in the door-opening task. For the pick and place tasks that contain a much larger dataset, Small, IMPALA, and ResNet18 all perform much worse than ResNet 34 and ResNet 50. We believe this result is quite exciting since it highlights the possibility of actually benefitting from using highly-expressive neural network models with TD-learning based RL methods trained on lots of diverse multi-task data (contrary to prior work [30]). We believe that this result is a valuable starting point for further scaling and innovation.

Learned spatial embeddings are crucial for performance. Next we study the impact of utilizing the learned spatial embeddings for encoding spatial information when converting the feature maps from the convolutional stack into a vector that is fed into the fully-connected part of the Q-function. We compare our choice to utilizing a spatial softmax as in Ebert et al. [9], and also global average pooling (GAP) that simply averages over the spatial information, typically utilized in supervised learning with ResNets.

Method	**Success rate**
PTR with spatial softmax	4/10
PTR with global average pooling	4/10
PTR with learned spatial embeddings **(Ours)**	**7/10**

TABLE XIII: **Ablation of PTR with spatial softmax and GAP on the croissant task.** Observe that PTR with learned spatial embeddings performs significantly better than using a spatial softmax or global average pooling.

As shown in Table XIII learned spatial embeddings outperform both of these prior approaches on the put croissant in pot task. We suspect that spatial softmax does not perform much better than the GAP approach since the softmax operation can easily get saturated when running gradient descent to fit value targets that are not centered in some range, which would effectively hinder its expressivity. This indicates that the approach of retaining spatial information like in PTR is required for attaining good performance.

Concatenating actions at each layer is crucial for performance. Next, we run PTR without passing in actions at each fully connected layer of the Q-function on the take croissant out of metallic bowl task and only directly concatenate the actions with the output of the convolutional layers before passing it into the fully-connected component of the network. On the croissant task, we find that not passing in actions at each layer only succeeds in **2/10** evaluation rollouts, which is significantly worse than the default PTR which passes in actions at each layer and succeeds in **7/10** evaluation rollouts (Table XIV).

Method	**Success rate**
PTR without actions passed in at each FC layer	2/10
PTR with actions passed in at each FC layer (Ours)	**7/10**

TABLE XIV: **Ablation of PTR with actions passed in at each layer.** Observe that passing in actions at each fully-connected layer does lead to quite good performance.

Group normalization is more consistent than batch normalization. Next, we ablate the usage of group normalization over batch normalization in the ResNet 34 Q-functions that

Method	Croissant out of metallic bowl	Cucumber in pot
PTR with batch norm. (relative)	+ 28.0% (7/10 → 9/10)	- 60.0% (5/10 → 2/10)

TABLE XV: **Relative performance of PTR with batch normalization with respect to PTR with group normalization.** Observe that while utilizing batch normalization in PTR can be sometimes more effective than using group normalization (e.g., take croissant out of metallic bowl task), it may also be highly ineffective and can reduce success rates significantly in other tasks. The performance numbers to the left of the → correspond to the performance of PTR with group normalization and the performance to the right of → is the performance with batch normalization.

PTR uses. We found that batch normalization was generally harder to train to attain Q-function plots that exhibit a roughly increasing trend over the course of a trajectory. That said, on some tasks such as the croissant in pot task, we did get a reasonable Q-function, and found that batch normalization can perform well. On the other hand, on the put cucumber in pot task, we found that batch normalization was really ineffective. These results are shown in Table XV, and they demonstrate that batch normalization may not be as consistent and reliable with PTR as group normalization.

Choice of the reward function. Finally, we present some results that ablate the choice of the reward function utilized for training PTR from data that entirely consists of demonstrations. In our main set of experiments, we labeled the last three timesteps of every trajectory with a reward of +1 and annotated all other timesteps with a 0 reward. We tried perhaps the most natural choice of labeling only the last timestep with a 0 reward on the croissant task and found that this choice succeeds **0/10** times, compared to annotating the last three timesteps with a +1 reward which succeeds **7/10** times. We suspect that this is because only annotating the last timestep with a +1 reward is not ideal for two reasons: first, the task is often completed in the dataset much earlier than the observation shows the task complete, and hence the last-step annotation procedure induces a non-Markovian reward function, and second, only labeling the last step with a +1 leads to overly conservative Q-functions when used with PTR, which may not lead to good policies.

F. More Details on Online Fine-tuning

Offline pre-training. For both PTR and BC baseline, we used 40 open-door demonstrations as target task data and combined them with the Bridge Dataset to pre-train the policy. To reduce the training time in the real system, we used ResNet 18 backbones.

Reset policy. For the reset policy, we additionally collected 22 close-door demonstrations as the target task data and pre-trained the policy with PTR. Similar to the open-door policy, we used ResNet 18 backbones to save training time.

Reward classifier. We used a ResNet 34 classification model and trained it to detect whether the door is open or closed from visual inputs. For the training data, we manipulated the robot to collect around 20 positive and negative trajectories for both open and closed doors.

Method. As shown by Nakamoto et al. [42] in simulation, offline value function initializations that learn conservative Q-functions may not be effective at fine-tuning if the learned Q-values are not at the same scale as the ground-truth return of the behavior policy. While this property does not affect offline performance, it is crucial to enforce this property during fine-tuning. That said, this property can be "baked in" by simply preventing the CQL regularizer from minimizing the learned Q-function if its values fall below the Monte-Carlo return of the trajectories in the dataset. Therefore, for the online fine-tuning experiment, we incorporate this constraint into PTR.

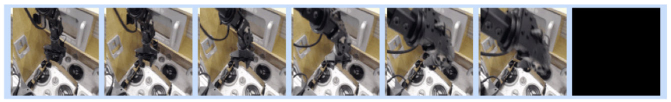
Fig. 21: **Example of unsafe behaviors when running SACfD.** The robot collides with the camera during online exploration, resulting in a system crash.

Hyperparameters. For both online fine-tuning with PTR and SACfD, we performed the experiment by mixing the Bridge Dataset, offline target data, and the online data in a ratio (β) of 0.35, 0.15, and 0.5. For PTR, we used the CQL alpha value of 5 for the offline phase and 0.5 for the online phase.

Evaluation. The results shown in Figure 8 were evaluated autonomously every 5K environment step during the online fine-tuning. Each evaluation was assessed with 10 trials, one from each initial position. The results shown in Figure VI were additionally evaluated over 3 trials from each initial position, using the offline initialization and the final checkpoint obtained after 20K environment steps of online fine-tuning.