Title
Large-Scale Real-World Robotic Manipulation Using Diverse Data

Author
Ebert, Frederik David

Publication Date
2022

Peer reviewed|Thesis/dissertation

Large-Scale Real-World Robotic Manipulation Using Diverse Data

by

Frederik D. Ebert

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Chelsea Finn, Co-chair
Professor Pieter Abbeel
Professor Hannah Stuart

Summer 2022

Large-Scale Real-World Robotic Manipulation Using Diverse Data

Abstract

Large-Scale Real-World Robotic Manipulation Using Diverse Data

by

Frederik D. Ebert

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Chair

Professor Chelsea Finn, Co-chair

Recent breakthroughs in computer vision and natural language processing have been largely propelled by scaling up both dataset diversity as well as model capacity, leading to robust generalization. In this thesis I am addressing the question of 1) whether for learning-based robotic manipulation we can similarly scale up dataset diversity and model capacity in order to achieve generalization and adaptation to new scenes and environments, new objects, new tasks and even different types of robots, and 2) the question of how re-collecting data from scratch for every new task and environment can be avoided, since this often leads to poor generalization and performance. To answer these questions we propose two different methodologies, a model-based reinforcement learning approach based on video-prediction, and a model-free and imitation-learning-based approach. We collect several of the biggest robotic interaction datasets to date, and show that by leveraging and effectively reusing diverse prior datasets, we can allow an agent to generalize to never-before-seen objects, learn new tasks based on only a handful of demonstrations, and even adapt to new robot types.

Contents

List of Figures

List of Tables

Acknowledgments

I want to express my deepest gratitude towards my advisors Sergey Levine and Chelsea Finn, who have taught me innumerable skills and guided my through the ups and downs of this long journey. For their great support, I would like to thank the two other members of my thesis committee, Pieter Abbeel and Hannah Stuart.

I am very grateful to my collaborators Yanlai Yang, Stephen Tian, Aviral Kumar, Anikait Singh, Annie Xie, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Karl Pertsch, Oleh Rybkin, Dinesh Jayaraman, Alex Lee, Roberto Calandra, and Sudeep Dasari without whom none of this would have been possible.

I also want to thank my Bachelor and Master thesis mentors at TU Munich Sang-ik An, Dongheui Lee and Patrick van der Smagt for inspiring me to conduct research in learning-based robotics.

For the arduous task of collecting thousands of trajectories of demonstration data (and saving my sanity), I want to thank Nicholas Lofrese and Abhraham Lee. I also want to thank my mentors at TU Munich who gave invaluable advice during the development of my exoskeleton robot during Gymnasium (German high school), Ingo Kresse, Alexis Maldonado and Federico Ruiz. This time really helped lay the foundation of excitement for intelligent robotics.

My most profound debt goes to towards my family, Frank, Kathrin, Sophie, Kai and Nici for supporting me since the very beginning. I want to thank my father Frank for teaching me mechanical and electronics skills and supporting me build my exoskeleton robots for the science fair competitions during Gymnasium.

Finally I want to thank my partner Yisheng for her support and encouragement and Gnocchi Li for his moral support and donation of experiment objects.

Chapter 1

Introduction

It has been an old dream of our civilization to develop machines that are able to follow a person's command to manipulate their environment with similar dexterity and intelligence as humans [124, 182, 156]. However even after many decades of research and engineering of robotic systems, the level of dexterity of even the most advanced robotic manipulation systems is still a far cry from human capabilities. Traditionally, robots have used a pipe-lined approach [17] where objects are detected in the scene, a plan is computed, e.g. through logical inference [83], and the plan is executed on the robot, e.g. by using motion planning. However such a pipe-lined approach requires engineers to precisely specify each component, i.e. the object detector, the abstraction of the logical reasoning system and the motion planner, resulting in poor scalability, since each of the modules often has to be adapted for every single new use-case, often requiring accurate 3D-models of all objects in the environment, which can become prohibitively difficult in unstructured environments such as a household or a nursing home.

With the advent of end-to-end computer vision [95], a new paradigm of robotic control and robot learning started, where we seek to learn as many parts of the decision making system as possible. In unstructured environments this often has the advantage of being more scalable since the system learns the correct features based on the data [105, 16], rather than requiring labor-intensive engineering cycles, leading to both better generalization, e.g. a system that is able to grasp objects never seen before, or better performance e.g. in the case of autonomous car racing [180].

In this thesis we focus on the problem setting of learning-based robotic manipulation, however many of the proposed methods can also be applied to other robotic domains, or even outside of the field of robotics.

Most learning-based robotic control methods today require recollecting data from scratch for every new task and environment. This is both inefficient and leads to poor generalization. For example hundreds of demonstrations are needed [208] to reliably solve a task via imitation learning when no prior data is used. We instead propose to *reuse* data by leveraging large and diverse existing datasets of robotic experience to learn representations that allow learning new tasks faster, i.e. with a smaller number of demonstrations. Both natural language

processing (NLP) and computer vision recently experienced breakthroughs [18, 145] in terms of generalization capabilities which is largely driven by the scale of the dataset and the size of the model. We hypothesize that scaling up dataset and model sizes can similarly provide a generalization boost in robotics.

Problem Statement The problem statement of this work is threefold 1) we aim to develop robotic control algorithms that enable a robot to learn new tasks as efficiently as possible, e.g. with the fewest number of human demonstrations or with the least amount of environment interactions possible 2) to develop algorithms that allow skills to generalize to new, unseen objects and 3) algorithms that allow skills to be adapted to held-out scenes and environments as well as held-out robots.

We propose to solve these questions by pre-training algorithms on large and diverse robot datasets and finetuning or jointly training with small amounts of target data which can come from new domains or environments, contain new objects or even new robots.

In particular we study reuse of robotic data in two different settings — in the setting where the data is collected by scripted random robot motions, pushing and picking objects in a bin, and where the data is collected via human teleoperation, by using a VR-headset and controlling the robot's end-effector to solve a wide range of every-day-like kitchen tasks various toykitchen environments. Both types of data have different advantages and shortcomings: While data from random scripted policies is cheap to collect as it only needs minimal human supervision, human teleoperated data is much more costly in terms of human effort. However human teleoperation allows the robot to visit much more complex sequences of states, e.g. opening a micro-wave and taking something out, which are virtually impossible to visit by chance through a random scripted policy. Therefore a robot learning system which uses random scripted data mainly learns to solve basic pushing and grasping motions, whereas a system that uses teleoperated data is able to produce more complex behaviors as long as a sufficient number of demonstrations is provided for a particular task. However at the same time, since random scripted policies collect data much more cheaply, we are able to obtain orders of magnitudes more data.

We found a model-based reinforcement learning approach to work well on the random scripted data, while a model-free reinforcement learning and imitation learning approach has become our method of choice the human-teleoperated robot data.

Structure and Contributions of the Thesis In Part I of this thesis I will introduce *visual foresight*, our video-prediction-based model-based reinforcement learning algorithm which learns a visual dynamics model from large amounts of data collected with a scripted random policy and uses that model to plan a sequences of actions leading to a user-specified goal.

In chapter 2, which is based on our paper [56], we describe deep neural network architectures that are effective for predicting pixel-level observations amid occlusions and with novel objects. We present several practical methods for specifying and evaluating progress towards

the goal—including distances to goal pixel positions, registration to goal images, and success classifiers—and compare their effectiveness and use-cases. Our evaluation on a real robot includes manipulation of previously unseen objects, handling multiple objects, pushing objects around obstructions, handling clutter, manipulating deformable objects such as cloth, recovering from large perturbations, and grasping and maneuvering objects to user-specified locations in 3D-space. Our results represent a significant advance in the *generality* of skills that can be acquired by a real robot operating on raw pixel values using a single model.

In chapter 3, which based on our paper [31], we describe how we expanded the dataset with a number of additional robots (7 in total) gripper types, viewpoints and backgrounds and we show that by applying visual foresight on this diverse data, it allows obtaining models that generalize in zero shot to novel objects, novel viewpoints, and novel table surfaces. We also show that, when these models are finetuned with small amounts of data (around 400 trajectories), they can generalize to unseen grippers and new robot platforms, and perform better than robot-specific and environment-specific training. We believe that this work takes an important step towards large-scale data-driven approaches to robotics, where data can be shared across institutions for greater levels of generalization and performance.

In Part II, we use human-teleoperated robot data instead of trajectories of random, scripted behavior, since it allows learning more complex tasks, that cannot be discovered with random data alone. We found imitation-learning-based and offline reinforcement-learning-based approaches to work better for teleoperated data, and we suspect that this is because with teleoperated data, actions become highly correlated with states, as well as next states leading to spurious correlations that render dynamics models inaccurate.

In chapter 4, which based on our paper [45], we introduce the bridge dataset which consists of 7,200 human teleoperated demonstrations for 71 tasks in 10 environments, and constitutes the largest public multi-domain, multi-task dataset of human demonstrations published to date. We show that by jointly training an imitation learning agent on the bridge dataset and a small amount of target data, we can obtain much better generalization than when training on the target domain data alone. Moreover we show that jointly training with bridge and target domain data provides for a way to boost the performance of an entirely new skill. Our results suggest that accumulating and reusing diverse multi-task and multi-domain datasets, at least when all data is collected with the same type of robot, may make it possible for researchers to endow robots with generalizable skills using only a modest amount of in-domain data for their desired task.

Finally in chapter 5[1], we reuse the same bridge data set and apply offline reinforcement learning instead of imitation learning, and show that it allows for significantly higher success rates. We use the CQL algorithm [97] in combination with multi-task pre-training to provide a general offline RL-based initialization. To train on such diverse datasets with offline RL, we propose a number of important design decisions, including neural net architectures that we find particularly effective in this setting. Empirically, our results demonstrate that offline RL pre-training followed by offline RL finetuning on individual tasks can significantly improve

[1] which is based on our recent conference pre-preprint

over the performance of imitation learning methods as well as running joint training on all tasks from scratch. This finding is significant because it indicates that, even though both the prior dataset and the new task-specified demonstrations consist of expert human demonstrations, can leverage such data more effectively than imitation learning and standard offline RL methods.

Part I

Large-Scale Model-Based Robot Learning

Chapter 2

Visual Foresight: Video-Prediction-Based Control

2.1 Introduction

Humans are faced with a stream of high-dimensional sensory inputs and minimal external supervision, and yet, are able to learn a range of complex, generalizable skills and behaviors. While there has been significant progress in developing deep reinforcement learning algorithms that learn complex skills and scale to high-dimensional observation spaces, such as pixels [164, 123, 107, 159], learning behaviors that *generalize* to new tasks and objects remains an open problem. The key to generalization is diversity. When deployed in a narrow, closed-world environment, a reinforcement learning algorithm will recover skills that are successful only in a narrow range of settings. Learning skills in diverse environments, such as the real world, presents a number of significant challenges: external reward feedback is extremely sparse or non-existent, and the agent has only indirect access to the state of the world through its senses, which, in the case of a robot, might correspond to cameras and joint encoders.

We approach the problem of learning generalizable behavior in the real world from the standpoint of sensory prediction. Prediction is often considered a fundamental component of intelligence [19]. Through prediction, it is possible to learn useful concepts about the world even from a raw stream of sensory observations, such as images from a camera. If we predict raw sensory observations directly, we do not need to assume availability of low-dimensional state information or an extrinsic reward signal. Image observations are both information-rich and high-dimensional, presenting both an opportunity and a challenge. Future observations provide a substantial amount of supervisory information for a machine learning algorithm. However, the predictive model must have the capacity to predict these high-dimensional observations, and the control algorithm must be able to use such a model to effectively select actions to accomplish human-specified goals. Examples of such goals are shown in figure 2.1.

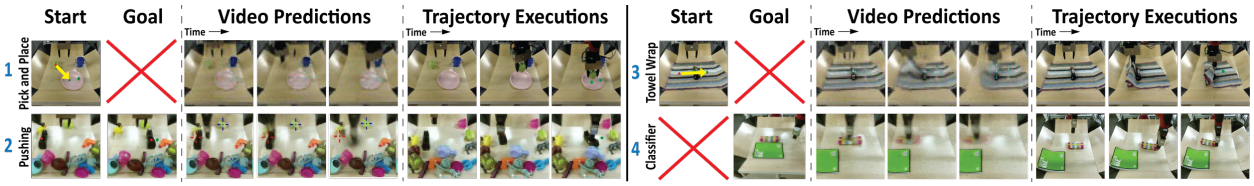


Figure 2.1: Our approach trains a single model from unsupervised interaction that generalizes to a wide range of tasks and objects, while allowing flexibility in goal specification and both rigid and deformable objects not seen during training. Each row shows an example trajectory. From left to right, we show the task definition, the video predictions for the planned actions, and the actual executions. Tasks can be defined as (top) moving pixels corresponding to objects, (bottom left) providing a goal image, or (bottom right) providing a few example goals. Best viewed in PDF.

We study control via prediction in the context of robotic manipulation, formulating a model-based reinforcement learning approach centered around prediction of raw sensory observations. One of the biggest challenges in learning-based robotic manipulation is generalization: how can we learn models that are useful not just for a narrow range of tasks seen during training, but that can be used to perform new tasks with new objects that were not seen previously? Collecting a training dataset that is sufficiently rich and diverse is often challenging in highly-structured robotics experiments, which depend on human intervention for reward signals, resets, and safety constraints. We instead set up a minimally structured robotic control domain, where data is collected by the robot via unsupervised interaction with a wide range of objects, making it practical to collect large amounts of interaction data. The robot collects a stream of raw sensory observations (image pixels), without any reward signal at training time, and without the ability to reset the environment between episodes. This setting is both realistic and necessary for studying RL in diverse real-world environments, as it enables automated and unattended collection of diverse interaction experience. Since the training setting affords no readily accessible reward signal, learning by prediction presents an appealing option: the supervision signal for prediction is always available even in the stream of unsupervised experience. We therefore propose to learn action-conditioned predictive models directly on raw pixel observations, and show that they can be used to accomplish a range of pixel-based manipulation tasks on a real robot in the physical world at test-time.

The main contributions of this work are as follows. We present *visual MPC*, a general framework for deep reinforcement learning with sensory prediction models that is suitable for learning behaviors in diverse, open-world environments (see figure 2.2). We describe deep neural network architectures that are effective for predicting pixel-level observations amid occlusions and with novel objects. Unlike low-dimensional representations of state, specifying and evaluating the reward from pixel predictions at test-time is nontrivial: we present several practical methods for specifying and evaluating progress towards the goal—including distances to goal pixel positions, registration to goal images, and success classifiers—and

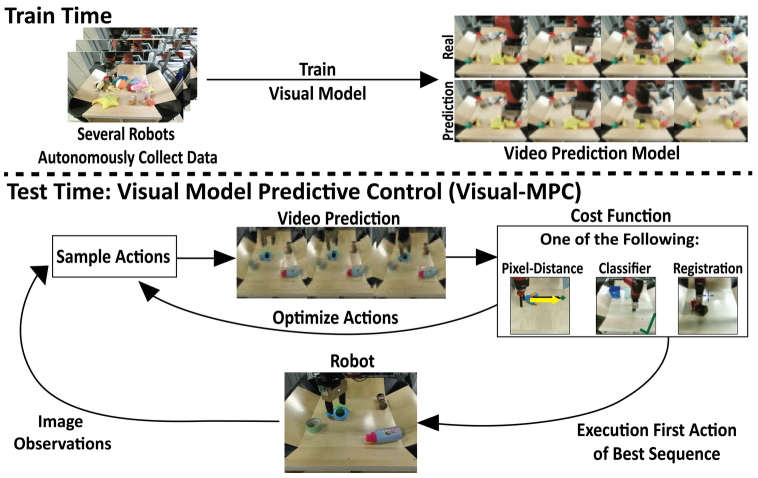


Figure 2.2: Overview of visual MPC. (top) At training time, interaction data is collected autonomously and used to train a video-prediction model. (bottom) At test time, this model is used for sampling-based planning. In this work we discuss three different choices for the planning objective.

compare their effectiveness and use-cases. Finally, our evaluation shows how these components can be combined to enable a real robot to perform a range of object manipulation tasks from raw pixel observations. Our experiments include manipulation of previously unseen objects, handling multiple objects, pushing objects around obstructions, handling clutter, manipulating deformable objects such as cloth, recovering from large perturbations, and grasping and maneuvering objects to user-specified locations in 3D-space. Our results represent a significant advance in the *generality* of skills that can be acquired by a real robot operating on raw pixel values using a single model.

This article combines and extends material from several prior conference papers [56, 47, 46, 187], presenting them in the context of a unified system. We include additional experiments, including cloth manipulation and placing tasks, a quantitative *multi-task* experiment assessing the performance of our method on a wide range of distinct tasks with a single model, as well as a comprehensive, open-sourced simulation environment to facilitate future research and better reproducibility. The code and videos can be found on the project webpage[1].

2.2 Related Work

Model-based reinforcement learning. Learning a model to predict the future, and then using this model to act, falls under the general umbrella of model-based reinforcement learning. Model-based RL algorithms are generally known to be more efficient than model-free

[1]For videos & code: `https://sites.google.com/view/visualforesight`

methods [35], and have been used with both low-dimensional [33] and high-dimensional [106] model classes. However, model-based RL methods that directly operate on raw image frames have not been studied as extensively. Several algorithms have been proposed for simple, synthetic images [177] and video game environments [42, 71, 133], but have not been evaluated on generalization or in the real world, while other work has also studied model-based RL for individual robotic skills [58, 207, 21]. In contrast to these works, we place special emphasis on *generalization*, studying how predictive models can enable a real robot to manipulate previously unseen objects and solve new tasks. Several prior works have also sought to learn inverse models that map from pairs of observations to actions, which can then be used greedily to carry out short-horizon tasks [2, 127]. However, such methods do not directly construct longer-term plans, relying instead on greedy execution. In contrast, our method learns a forward model, which can be used to plan out a sequence of actions to achieve a user-specified goal.

Self-supervised robotic learning. A number of recent works have studied self-supervised robotic learning, where large-scale unattended data collection is used to learn individual skills such as grasping [141, 109, 23, 143], push-grasp synergies [204], or obstacle avoidance [84, 64]. In contrast to these methods, our approach learns predictive models that can be used to perform a variety of manipulation skills, and does not require a success measure, event indicator, or reward function during data collection.

Sensory prediction models. We propose to leverage sensory prediction models, such as video-prediction models, to enable large-scale self-supervised learning of robotic skills. Prior work on action-conditioned video prediction has studied predicting synthetic video game images [133, 27], 3D point clouds [21], and real-world images [15, 54, 88], using both direct autoregressive frame prediction [120, 54, 88] and latent variable models [8, 99]. Several works have sought to use more complex distributions for future images, for example by using pixel autoregressive models [88, 151]. While this often produces sharp predictions, the resulting models are extremely demanding computationally. Video prediction without actions has been studied for unstructured videos [120, 189, 175] and driving [114, 32]. In this work, we extend video prediction methods that are based on predicting a transformation from the previous image [54, 32].

2.3 Overview

In this section, we summarize our visual model-predictive control (MPC) method, which is a model-based reinforcement learning approach to end-to-end learning of robotic manipulation skills. Our method, outlined in Figure 2.2, consists of three phases: unsupervised data collection, predictive model training, and planning-based control via the model at test-time.

Unsupervised data collection: At *training time*, data is collected autonomously by applying random actions sampled from a pre-specified distribution. It is important that this distribution allows the robot to visit parts of the state space that are relevant for solving the

intended tasks. For some tasks, uniform random actions are sufficient, while for others, the design of the exploration strategy takes additional care, as detailed in Sections 2.7 and 2.9.
Model training: Also during *training time*, we train a video prediction model on the collected data. The model takes as input an image of the current timestep and a sequence of actions, and generates the corresponding sequence of future frames. This model is described in Section 2.4.
Test time control: At *test time*, we use a sampling-based, gradient free optimization procedure, similar to a shooting method [12], to find the sequence of actions that minimizes a cost function. Further details, including the motivation for this type of optimizer, can be found in Section 2.6.

Depending on how the goal is specified, we use one of the following three cost functions. When the goal is provided by clicking on an object and a desired goal-position, a *pixel-distance cost-function*, detailed in Section 2.5, evaluates how far the designated pixel is from the goal pixels. We can specify the goal more precisely by providing a goal image in addition to the pixel positions and make use of *image-to-image registration* to compute a cost function, as discussed in Section 2.5. Finally, we show that we can specify more conceptual tasks by providing one or several examples of success and employing a *classifier-based* cost function as detailed in Section 2.5. The strengths and weaknesses of different costs functions and trade-offs between them are discussed in Section 2.5.

The model is used to plan T steps into the future, and the first action of the action sequence that attained lowest cost, is executed. In order to correct for mistakes made by the model, the actions are iteratively replanned at each real-world time step[2] $\tau \in \{0, ..., \tau_{max}\}$ following the framework of model-predictive control (MPC). In the following sections, we explain the video-prediction model, the planning cost function, and the trajectory optimizer.

2.4 Video Prediction for Control

In visual MPC, we use a transformation-based video prediction architecture, first proposed by Finn et al. [54]. The advantage of using transformation-based models over a model that directly generates pixels is two-fold: (1) prediction is easier, since the appearance of objects and the background scene can be reused from previous frames and (2) the transformations can be leveraged to obtain predictions about where pixels will move, a property that is used in several of our planning cost function formulations. The model, which is implemented as a recurrent neural network (RNN) g_θ parameterized by θ, has a hidden state h_t and takes in a previous image and an action at each step of the rollout. Future images $\hat{I}_{t+1}$ are generated by warping the previous generated image $\hat{I}_t$ or the previous true image I_t, when available, according to a 2-dimensional flow field $\hat{F}_{t+1 \leftarrow t}$. A simplified illustration of model's structure

[2]With real-world step we mean timestep of the real-world as opposed to predicted timesteps.

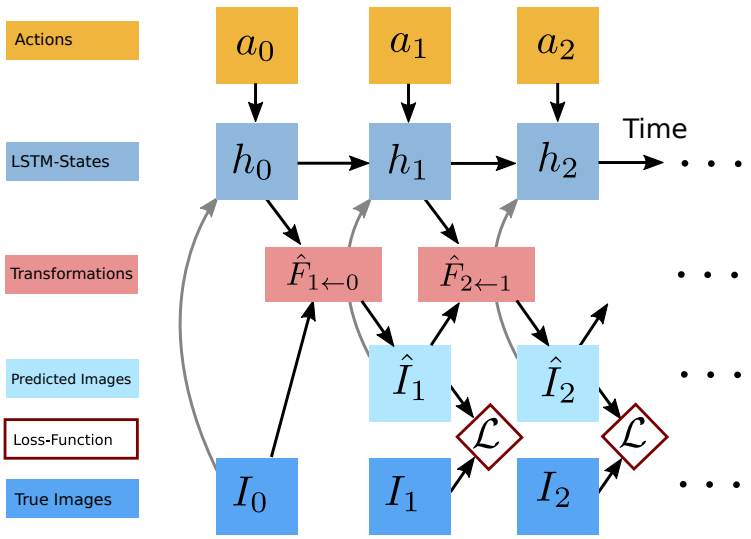


Figure 2.3: Computation graph of the video-prediction model. Time goes from left to right, a_t are the actions, h_t are the hidden states in the recurrent neural network, $\hat{F}_{t+1\leftarrow t}$ is a 2D-warping field, I_t are real images, and $\hat{I}_t$ are predicted images, $\mathcal{L}$ is a pairwise training-loss.

is given in figure 2.3. It is also summarized in the following two equations:

$$[h_{t+1}, \hat{F}_{t+1\leftarrow t}] = g_\theta(a_t, h_t, I_t) \tag{2.1}$$

$$\hat{I}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t \tag{2.2}$$

Here, the bilinear sampling operator $\diamond$ interpolates the pixel values bilinearly with respect to a location (x, y) and its four neighbouring pixels in the image, similar to [209]. Note that, as shown in figure 2.3, at the first time-step the real image is transformed, whereas at later timesteps previously generated images are transformed in order to generate multi-frame predictions. The model is trained with gradient descent on a ℓ_2 image reconstruction loss, denoted by $\mathcal{L}$ in figure 2.3. A forward pass of the RNN is illustrated in figure 2.4. We use a series of stacked convolutional LSTMs and standard convolutional layers interleaved with average-pooling and upsampling layers. The result of this computation is the 2 dimensional flow-field $\hat{F}_{t+1\leftarrow t}$ which is used to transform a current image I_t or $\hat{I}_t$. More details on the architecture are provided in Appendix A.1.

Predicting pixel motion. When using visual MPC with a cost-function based on start and goal pixel positions, we require a model that can effectively predict the 2D motion of the user-selected start pixels $\overset{(1)}{0}, \ldots, \overset{(P)}{0}$ up to T steps into the future[3]. More details about the cost functions are provided in section 2.5. Since the model we employ is transformation-based, this motion prediction capability emerges automatically, and therefore no external

[3]Note that when using a classifier-based cost function, we do not require the model to output transformations.

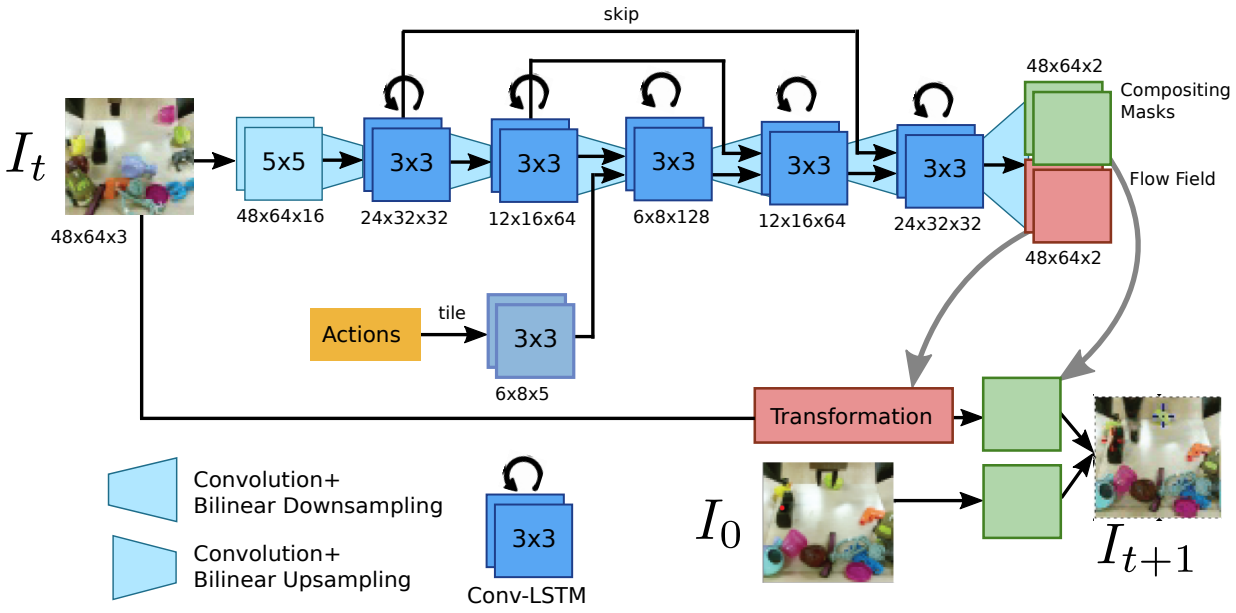


Figure 2.4: Forward pass through the recurrent SNA model. The image from the first time step I_0 is concatenated with the transformed images $\hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t$ multiplying each channel with a separate mask to produce the predicted frame for step $t+1$.

pixel motion supervision is required. To predict the future positions of the designated pixel d, the same transformations used to transform the images are applied to the distribution over designated pixel locations. The warping transformation $\hat{F}_{t+1\leftarrow t}$ can be interpreted as a stochastic transition operator allowing us to make probabilistic predictions about future locations of individual pixels:

$$\hat{P}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{P}_t \tag{2.3}$$

Here, P_t is a distribution over image locations which has the same spatial dimension as the image. For simplicity in notation, we will use a single designated pixel moving forward, but using multiple is straightforward. At the first time step, the distribution $\hat{P}_0$ is defined as 1 at the position of the user-selected designated pixel and zero elsewhere. The distribution $\hat{P}_{t+1}$ is normalized at each prediction step.

Since this basic model, referred to as dynamic neural advection (DNA), predicts images only based on the previous image, it is unable to recover shapes (e.g., objects) after they have been occluded, for example by the robot arm. Hence, this model is only suitable for planning motions where the user-selected pixels are not occluded during the manipulation, limiting its use in cluttered environments or with multiple selected pixels. In the next section, we introduce an enhanced model, which lifts this limitation by employing temporal skip connections.

Skip connection neural advection model. To enable effective tracking of objects through occlusions, we can add temporal skip connections to the model: we now transform pixels not

only from the previously generated image $\hat{I}_t$, but from all previous images $\hat{I}_1, ...\hat{I}_t$, including the context image I_0, which is a real image. All these transformed images can be combined to a form the predicted image $\hat{I}_{t+1}$ by taking a weighted sum over all transformed images, where the weights are given by masks $\mathbf{M}_t$ with the same size as the image and a single channel:

$$\hat{I}_{t+1} = \mathbf{M}_0(\hat{F}_{t+1\leftarrow 0} \diamond I_t) + \sum_{j=1}^{\tau} \mathbf{M}_j(\hat{F}_{t+1\leftarrow j} \diamond \hat{I}_j). \tag{2.4}$$

We refer to this model as the *skip connection neural advection model (SNA)*, since it handles occlusions by using temporal skip connections such that when a pixel is occluded, e.g., by the robot arm or by another object, it can still reappear later in the sequence. Transforming from all previous images comes with increased computational cost, since the number of masks and transformations scales with the number of time-steps τ. However, we found that in practice a greatly simplified version of this model, where transformations are applied only to the previous image and the *first image* of the sequence I_0, works equally well. Moreover we found that transforming the first image of the sequence is not necessary, as the model uses these pixels primarily to generate the image background. Therefore, we can use the first image directly, without transformation. More details can be found in the appendix A.1 and [47].

2.5 Planning Cost Functions

In this section, we discuss how to specify and evaluate goals for planning. One naïve approach is to use pixel-wise error, such as ℓ_2 error, between a *goal image* and the *predicted image*. However there is a severe issue with this approach: large objects in the image, i.e. the arm and shadows, dominate such a cost; therefore a common failure mode occurs when the planner matches the arm position with its position in the goal image, disregarding smaller objects. This failure motivates our use of more sophisticated mechanisms for specifying goals, which we discuss next.

Pixel Distance Cost

A convenient way to define a robot task is by choosing one or more *designated pixels* in the robot's camera view and choosing a destination where each pixel should be moved. For example, the user might select a pixel on an object and ask the robot to move it 10 cm to the left. This type of objective is general, in that it can define any object relocation task on the viewing plane. Further, success can be measured quantitatively, as detailed in section 3.5. Given a distribution over pixel positions P_0, our model predicts distributions over its positions P_t at time $t \in \{1, \ldots, T\}$. One way of defining the cost per time-step c_t is by using the expected Euclidean distance to the goal point d_g, which is straight-forward to calculate

from P_t and g, as follows:

$$c = \sum_{t=1,\ldots,T} c_t = \sum_{t=1,\ldots,T} \mathbb{E}_{\hat{d}_t \sim P_t} \left[\|\hat{d}_t - d_g\|_2 \right] \tag{2.5}$$

The per time-step costs c_t are summed together giving the overall planing objective c. The expected distance to the goal provides a smooth planning objective and enables longer-horizon tasks, since this cost function encourages movement of the designated objects into the right direction for each step of the execution, regardless of whether the goal-position can be reached within T time steps or not. This cost also makes use of the uncertainty estimates of the predictor when computing the expected distance to the goal. For multi-objective tasks with multiple designated pixels $d^{(i)}$ the costs are summed to together, and optionally weighted according to a scheme discussed in section 2.5.

Registration-Based Cost

We now propose an improvement over using pixel distances. When using pixel distance cost functions, it is necessary to know the *current* location of the object, $\overset{(1)}{0}, \ldots, \overset{(P)}{0}$ at each replanning step, so that the model can predict the positions of this pixel from the current step forward. To update the belief of where the target object currently is, we propose to register the current image to the start and optionally also to a *goal image*, where the designated pixels are marked by the user. Adding a goal image can make visual MPC more precise, since when the target object is close to the goal position, registration to the goal-image greatly improves the position estimate of the designated pixel. Crucially, the registration method we introduce is self-supervised, using the same exact data for training the video prediction model and for training the registration model. This allows both models to continuously improve as the robot collects more data.

Test time procedure. We will first describe the registration scheme at test time (see Figure 2.6(a)). We separately register the current image I_t to the start image I_0 and to the goal image I_g by passing it into the registration network R, implemented as a fully-convolutional neural network. The registration network produces a flow map $\hat{F}_{0\leftarrow t} \in \mathbb{R}^{H\times W\times 2}$, a vector field with the same size as the image, that describes the relative motion for every pixel between the two frames.

$$\hat{F}_{0\leftarrow t} = R(I_t, I_0) \qquad\qquad \hat{F}_{g\leftarrow t} = R(I_t, I_g) \tag{2.6}$$

The flow map $\hat{F}_{0\leftarrow t}$ can be used to warp the image of the current time step t to the start image I_0, and $\hat{F}_{g\leftarrow t}$ can be used to warp from I_t to I_g (see Figure 2.5 for an illustration). There is no difference to the warping operation used in the video prediction model, explained in section 2.4, equation B.2:

$$\hat{I}_0 = \hat{F}_{0\leftarrow t} \diamond I_t \qquad\qquad \hat{I}_g = \hat{F}_{g\leftarrow t} \diamond I_t \tag{2.7}$$

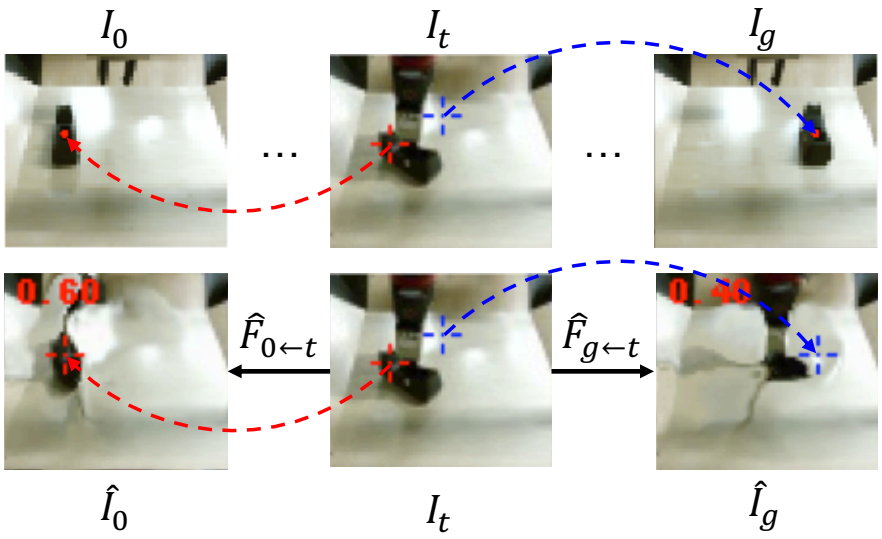


Figure 2.5: Closed loop control is achieved by registering the current image I_t globally to the first frame I_0 and the goal image I_g. In this example registration to I_0 succeeds while registration to I_g fails since the object in I_g is too far away.

In essence for a current image $\hat{F}_{0\leftarrow t}$ puts I_t in correspondence with I_0, and $\hat{F}_{g\leftarrow t}$ puts I_t in correspondence with I_g. The motivation for registering to both I_0 and I_g is to increase accuracy and robustness. In principle, registering to either I_0 or I_g is sufficient. While the registration network is trained to perform a global registration between the images, we only evaluate it at the points d_0 and d_g chosen by the user. This results in a cost function that ignores distractors. The flow map produced by the registration network is used to find the pixel locations corresponding to d_0 and d_g in the current frame:

$$\hat{d}_{0,t} = d_0 + \hat{F}_{0\leftarrow t}(d_0) \qquad \hat{d}_{g,t} = d_g + \hat{F}_{g\leftarrow t}(d_g) \tag{2.8}$$

For simplicity, we describe the case with a single designated pixel. In practice, instead of a single flow vector $\hat{F}_{0\leftarrow t}(d_0)$ and $\hat{F}_{g\leftarrow t}(d_g)$, we consider a neighborhood of flow-vectors around d_0 and d_g and take the median in the x and y directions, making the registration more stable. Figure 2.7 visualizes an example tracking result while the gripper is moving an object.

Registration-based pixel distance cost. Registration can fail when distances between objects in the images are large. During a motion, the registration to the first image typically becomes harder, while the registration to the goal image becomes easier. We propose a mechanism that estimates which image is registered correctly, allowing us to utilize only the successful registration for evaluating the planning cost. This mechanism gives a high weight λ_i to pixel distance costs c_i associated with a designated pixel $\hat{d}_{i,t}$ that is tracked successfully and a low, ideally zero, weight to a designated pixel where the registration is poor. We use the photometric distance between the true frame and the warped frame evaluated at $d_{0,i}$

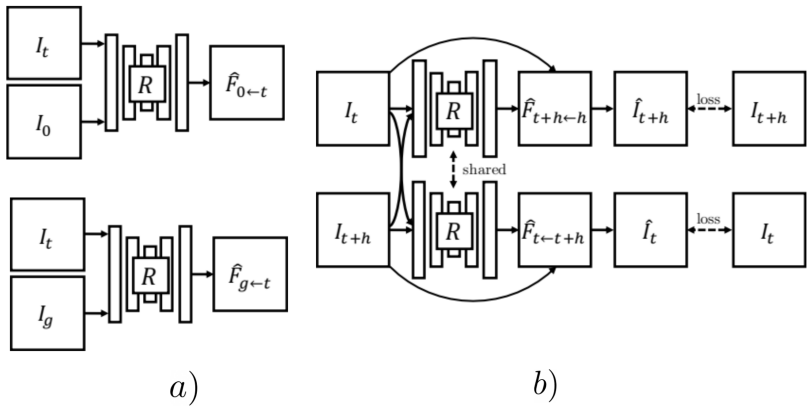


Figure 2.6: (a) At test time the registration network registers the current image I_t to the start image I_0 (top) and goal image I_g (bottom), inferring the flow-fields $\hat{F}_{0\leftarrow t}$ and $\hat{F}_{g\leftarrow t}$. (b) The registration network is trained by warping images from randomly selected timesteps along a trajectory to each other.

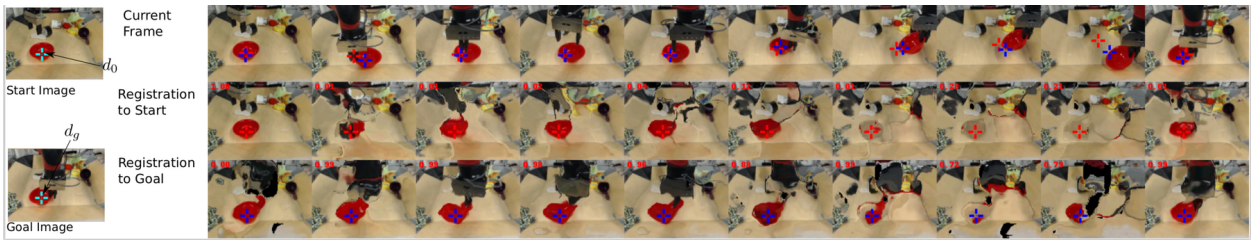


Figure 2.7: Outputs of registration network. The first row shows the timesteps from left to right of a robot picking and moving a red bowl, the second row shows each image warped to the initial image via registration, and the third row shows the same for the goal image. A successful registration in this visualization would result in images that closely resemble the start- or goal image. In the first row, the locations where the designated pixel of the start image d_0 and the goal image d_g are found are marked with red and blue crosses, respectively. It can be seen that the registration to the start image (red cross) is failing in the second to last time step, while the registration to the goal image (blue cross) succeeds for all time steps. The numbers in red, in the upper left corners indicate the trade off factors λ between the views and are used as weighting factors for the planning cost. (Best viewed in PDF)

and $d_{g,i}$ as an estimate for *local* registration success. A low photometric error indicates that the registration network predicted a flow vector leading to a pixel with a similar color, thus indicating warping success. However this does not necessarily mean that the flow vector points to the correct location. For example, there could be several objects with the same color and the network could simply point to the wrong object. Letting $I_i(d_i)$ denote the pixel value in image I_i for position d_i, and $\hat{I}_i(d_i)$ denote the corresponding pixel in the image warped by the registration function, we can define the general weighting factors λ_i as:

$$\lambda_i = \frac{||I_i(d_i) - \hat{I}_i(d_i)||_2^{-1}}{\sum_j^N ||I_j(d_j) - \hat{I}_j(d_j)||_2^{-1}}. \tag{2.9}$$

where $\hat{I}_i = \hat{F}_{i\leftarrow t} \diamond I_t$. The MPC cost is computed as the average of the costs c_i weighted by λ_i, where each c_i is the expected distance (see equation B.4) between the registered point $\hat{d}_{i,t}$ and the goal point $d_{g,i}$. Hence, the cost used for planning is $c = \sum_i \lambda_i c_i$. In the case of the single view model and a single designated pixel, the index i iterates over the start and goal image (and $N = 2$).

The proposed weighting scheme can also be used with multiple designated pixels, as used in multi-task settings and multi-view models, which are explained in section 2.8. The index i then also loops over the views and indices of the designated pixels.

Training procedure. The registration network is trained on the same data as the video prediction model, but it does not share parameters with it.[4] Our approach is similar to the optic flow method proposed by [121]. However, unlike this prior work, our method computes registrations for frames that might be many time steps apart, and the goal is not to extract optic flow, but rather to determine correspondences between potentially distant images. For training, two images are sampled at random times steps t and $t + h$ along the trajectory and the images are warped to each other in both directions.

$$\hat{I}_t = \hat{F}_{t\leftarrow t+h} \diamond I_{t+h} \qquad\qquad \hat{I}_{t+h} = \hat{F}_{t+h\leftarrow t} \diamond I_t \tag{2.10}$$

The network, which outputs $\hat{F}_{t\leftarrow t+h}$ and $\hat{F}_{t+h\leftarrow t}$, see Figure 2.6 (b), is trained to minimize the photometric distance between $\hat{I}_t$ and I_t and $\hat{I}_{t+h}$ and I_{t+h}, in addition to a smoothness regularizer that penalizes abrupt changes in the outputted flow-field. The details of this loss function follow prior work [121]. We found that gradually increasing the temporal distance h between the images during training yielded better final accuracy, as it creates a learning curriculum. The temporal distance is linearly increased from 1 step to 8 steps at 20k SGD steps. In total 60k iterations were taken.

The network R is implemented as a fully convolutional network taking in two images stacked along the channel dimension. First the inputs are passed into three convolutional layers each followed by a bilinear downsampling operation. This is passed into three layers of convolution each followed by a bilinear upsampling operation (all convolutions use stride

[4] In principle, sharing parameters with the video prediction model might be beneficial, but this is left for future work.

1). By using bilinear sampling for increasing or decreasing image sizes we avoid artifacts that are caused by strided convolutions and deconvolutions.

Classifier-Based Cost Functions

An alternative way to define the cost function is with a goal classifier. This type of cost function is particularly well-suited for tasks that can be completed in multiple ways. For example, for a task of rearranging a pair objects into relative positions, i.e. pushing the first object to the left of the second object, the absolute positions of the objects do not matter nor does the arm position. A classifier-based cost function allows the planner to discover any of the possible goal states.

Unfortunately, a typical image classifier will require a large amount of labeled examples to learn, and we do not want to collect large datasets for each and every task. Instead, we aim to learn a goal classifier from only a few positive examples, using a meta-learning approach. A few positive examples of success are easy for people to provide and are the minimal information needed to convey a goal.

Formally, we consider a goal classifier $\hat{y} = f(\mathbf{o})$, where $\mathbf{o}$ denotes the image observation, and $\hat{y} \in [0, 1]$ indicates the predicted probability of the observation being of a successful outcome of the task. Our objective is to infer a classifier for a new task $\mathcal{T}_j$ from a few positive examples of success, which are easy for a user to provide and encode the minimal information needed to convey a task. In other words, given a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1)|k = 1...K\}_j$, our goal is to infer a classifier for task $\mathcal{T}_j$.

Meta-learning for few-shot goal inference. To solve the above problem, we propose learning a few-shot classifier that can infer the goal of a new task from a small set of goal examples, allowing the user to define a task from a few examples of success. To train the few-shot classifier, we first collect a dataset of both positive and negative examples for a wide range of tasks. We then use this data to learn how to learn goal classifiers from a few positive examples. Our approach is illustrated in Figure 2.8.

We build upon model-agnostic meta-learning (MAML) [52], which learns initial parameters θ for model f that can efficiently adapt to a new task with one or a few steps of gradient descent. Grant et al. [67] proposed an extension of MAML, referred to as concept acquisition through meta-learning (CAML), for learning to learn new concepts from positive examples alone. We apply CAML to the setting of acquiring goal classifiers from positive examples, using a meta-training data with both positive and negative examples. The result of the meta-training procedure is an initial set of parameters that can be used to learn new goal classifiers at test time.

Test time procedure. At test time, the user provides a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1)|k = 1...K\}_j$, which are then used to infer a task-specific goal classifier C_j. In particular, the meta-learned parameters θ are updated through gradient descent to adapt to task $\mathcal{T}_j$:

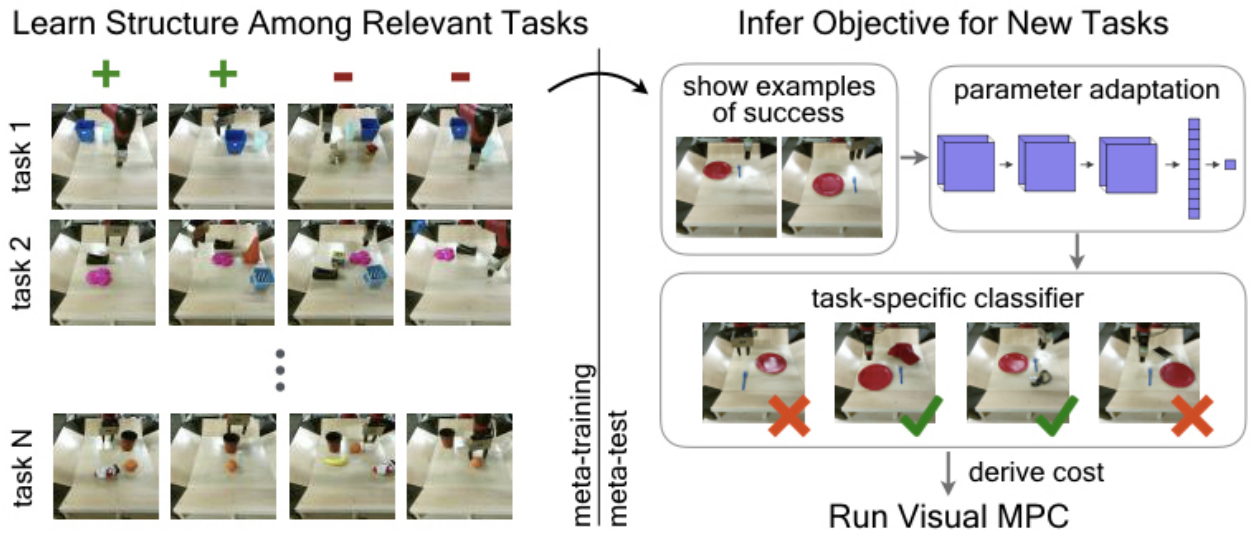


Figure 2.8: We propose a framework for quickly specifying visual goals. Our goal classifier is meta-trained with positive and negative examples for diverse tasks (left), which allows it to meta-learn that some factors matter for goals (e.g., relative positions of objects), while some do not (e.g. position of the arm). At meta-test time, this classifier can learn goals for new tasks from a few of examples of success (right - the goal is to place the fork to the right of the plate). The cost can be derived from the learned goal classifier for use with visual MPC.

$$C_j(\mathbf{o}) = f(\mathbf{o}; \theta'_j) = f\big(\mathbf{o}; \theta - \alpha \nabla_\theta \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_j^+} \mathcal{L}(y_n, f(\mathbf{o}_n; \theta))\big)$$

where $\mathcal{L}$ is the cross-entropy loss function, α is the step size, and θ' denotes the parameters updated through gradient descent on task $\mathcal{T}_j$.

During planning, the learned classifier C_j takes as input an image generated by the video prediction model and outputs the predicted probability of the goal being achieved for the task specified by the few examples of success. To convert this into a cost function, we treat the probability of success as the planning cost for that observation. To reduce the effect of false positives and mis-calibrated predictions, we use the classifier conservatively by thresholding the predictions so that reward is only given for confident successes. Below this threshold, we give a reward of 0 and above this threshold, we provide the predicted probability as the reward.

Training time procedure. During meta-training, we explicitly train for the ability to infer goal classifiers for the set of training tasks, $\{\mathcal{T}_i\}$. We assume a small dataset $\mathcal{D}_i$ for each task $\mathcal{T}_i$, consisting of both positive and negative examples: $\mathcal{D}_i := \{(\mathbf{o}_n, y_n) | n = 1...N\}_i$. To learn the initial parameters θ, we optimize the following objective using Adam [90]:

$$\min_{\theta} \sum_{i} \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_i^{\text{test}}} \mathcal{L}(y_n, f(\mathbf{o}_n; \theta_i'))$$

In our experiments, our classifier is represented by a convolutional neural network, consisting of three convolutional layers, each followed by layer normalization and a ReLU nonlinearity. After the final convolutional layer, a spatial soft-argmax operation extracts spatial feature points, which are then passed through fully-connected layers.

When to Use Which Cost Function?

We have introduced three different forms of cost function, pixel distance based cost functions with and without registration, as well as classifier-based cost functions. Here we discuss the relative strengths and weaknesses of each.

Pixel distance based cost functions have the advantage that they allow moving objects precisely to target locations. They are also easy to specify, without requiring any example goal images, and therefore provide an easy and fast user interface. The pixel distance based cost function also has a high degree of robustness against distractor objects and clutter, since the optimizer can ignore the values of other pixels; this is important when targeting diverse real-world environments. By incorporating an image of the goal, we can also add a registration mechanism to allow for more robust closed-loop control, at the cost of a more significant burden on the user.

The classifier-based cost function allows for solving more abstract tasks since it can capture invariances, such as the position of the arm, and settings where the absolute positions of an object is not relevant, such as positioning a cup in front of a plate, irrespective of where the plate is. Providing a few example images takes more effort than specifying pixel locations but allows a broader range of goal sets to be specified.

2.6 Trajectory Optimizer

The role of the optimizer is to find actions sequences $a_{1:T}$ that minimize the sum of the costs $c_{1:T}$ along the planning horizon T. We use a simple stochastic optimization procedure for this, based on the cross-entropy method (CEM), a gradient-free optimization procedure. CEM consists of iteratively resampling action sequences and refitting Gaussian distributions to the actions with the best predicted cost.

Although a variety of trajectory optimization methods may be suitable, one advantage of the stochastic optimization procedure is that it allows us to easily ensure that actions stay within the distribution of actions the model encountered during training. This is crucial to ensure that the model does not receive out-of-distribution inputs and makes valid predictions. Algorithm 1 illustrates the planning process. In practice this can be achieved by defining admissible ranges for each dimension of the action vector and rejecting a sample if it is outside of the admissible range.

Algorithm 1 Planning in Visual MPC

Inputs: Predictive model g, planning cost function c
for $t = 0...T-1$ **do**
  for $i = 0...n_{iter}-1$ **do**
    if $i == 0$ **then**
      Sample M action sequences $\{a^{(m)}_{t:t+H-1}\}$ from $\mathcal{N}(0, I)$ or custom sampling distribution
    else
      Sample M action sequences $a^{(m)}_{t:t+H-1}$ from $\mathcal{N}(\mu^{(i)}, \Sigma^{(i)})$
    Check if sampled actions are within admissible range, otherwise resample.
    Use g to predict future image sequences $\hat{I}^{(m)}_{t:t+H-1}$ and probability distributions $\hat{P}^{(m)}_{t:t+H-1}$
    Evaluate action sequences using a cost function c
    Fit a diagonal Gaussian to the k action samples with lowest cost, yielding $\mu^{(i)}, \Sigma^{(i)}$
  Apply first action of best action sequence to robot

In the appendix A.3 we present a few improvements to the CEM optimizer for visual MPC.

2.7 Custom Action Sampling Distributions

When collecting data by sampling from simple distributions, such as a multivariate Gaussian, the skills that emerged were found to be generally restricted to pushing and dragging objects. This is because with simple distributions, it is very unlikely to visit states like picking up and placing of objects or folding cloth. Not only would the model be imprecise for these kinds of states, but also during planning it would be unlikely to *find* action sequences that grasp an object or fold an item of clothing. We therefore explore how the sampling distribution used both in data collection and sampling-based planning can be changed to visit these, otherwise unlikely, states more frequently, allowing more complex behavior to emerge.

To allow picking up and placing of objects as well as folding of cloth to occur more frequently, we incorporate a simple "reflex" during data collection, where the gripper automatically closes, when the height of the wrist above the table is lower than a small threshold. This reflex is inspired by the palmar reflex observed in infants [157]. With this primitive, when collecting data with rigid objects about 20% of trajectories included some sort of grasp. For deformable objects such as towels and cloth, this primitive helps increasing the likelihood of encountering states where cloths are folded. We found that the primitive can be slightly adapted to avoid cloths becoming tangled up. More details are provided in Appendix A.2.

It is worth noting that, other than this reflex, no grasping-specific or folding-specific engineering was applied to the policy, allowing a joint pushing, grasping and folding policy to emerge through planning (see figure A.4 in the appendix). In our experiments, we evaluate our method using data obtained both with and without the grasping reflex, evaluating both purely non-prehensile and combined prehensile and non-prehensile manipulation.

2.8 Multi-View Visual MPC

The visual MPC algorithm as described so far is only able to solve manipulation tasks specified in 2D, like rearranging objects on the table. However, this can impose severe limitations; for example, a task such as lifting an object to a particular position in 3D cannot be fully specified with a single view, since it would be ambiguous. We use a combination of two views, taken from two cameras arranged appropriately, to jointly define a 3D task. Figure 2.9 shows the robot setup, including two standard webcams observing the workspace from different angles. The registration method described in the previous section is used separately per view to allow for dynamic retrying and solving temporally extended tasks. The planning costs from each view are combined using weighted averaging where the weights are provided by the registration network (see equation 2.9). Rows 5 and 6 of figure 2.12 show a 3D object positioning task, where an object needs to be positioned at a particular point in 3D space. This task needs two views to be fully specified.

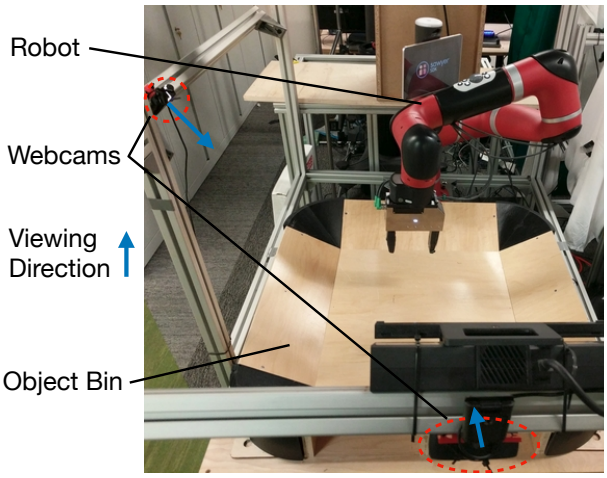


Figure 2.9: Robot setup, with 2 standard webcams arranged at different viewing angles.

2.9 Experimental Evaluation

In this section we present both qualitative and quantitative performance evaluations of visual MPC on various manipulation tasks assessing the degree of generalization and comparing different prediction models and cost functions and with a hand-crafted baseline. In Figures 2.1 and 2.12 we present a set of qualitative experiments showing that visual MPC trained fully self-supervised is capable of solving a wide range of complex tasks. Videos for the qualitative examples are at the following webpage[5]. In order to perform quantitative comparisons, we define a set of tasks where the robot is required to move object(s) into a goal configuration.

[5]Videos & code: `https://sites.google.com/view/visualforesight/`

	moved imp. ± std err. of mean	stationary imp. ± std err. of mean
DNA [56]	0.83 ±0.25	-1.1 ± 0.2
SNA	**10.6 ± 0.82**	**-1.5 ± 0.2**

Table 2.1: Results for multi-objective pushing on 8 object/goal configurations with 2 seen and 2 novel objects. Values indicate improvement in distance from starting position, higher is better. Units are pixels in the 64x64 images.

For measuring success, we use a distance-based evaluation where a human annotates the positions of the objects after pushing allowing us to compute the remaining distance to the goal.

Comparing Video Prediction Architectures

We first aim to answer the question: Does visual MPC using the occlusion-aware SNA video prediction model that includes temporal skip connections outperform visual MPC with the dynamic neural advection model (DNA)[56] *without* temporal skip-connections?

To examine whether our skip-connection model (SNA) helps with handling occlusions, we devised a task that requires the robot to push one object, while keeping another object stationary. When the stationary object is in the way, the robot must move the target object around it. This is illustrated on the left side of Figure A.5 in the appendix. While pushing the target object, the gripper may occlude the stationary object, and the task can only be performed successfully if the model can make accurate predictions through this occlusion. These tasks are specified by selecting one starting pixel on the target object, a goal pixel location for the target object, and commanding the obstacle to remain stationary by selecting the same pixel on the obstacle for both start and goal.

We use four different object arrangements with two training objects and two objects that were not seen during training. We find that, in most cases, the SNA model is able to find a valid trajectory, while the DNA model, that is not able to handle occlusion, is mostly unable to find a solution. The results of our quantitative comparisons are shown in Table 2.1, indicating that temporal skip-connections indeed help with handling occlusion in combined pushing and obstacle avoidance tasks.

Evaluating Registration-Based Cost Functions

In this section we ask: How important is it to update the model's belief of where the target objects currently are? We first provide two qualitative examples: In example (5)-(6) of Figure 2.12 the task is to bring the stuffed animal to a particular location in 3D-space on the other side of the arena. To test the system's reaction to perturbations that could be

	Short	Long
Visual MPC + predictor propagation	83%	20%
Visual MPC + OpenCV tracking	83%	45%
Visual MPC + registration network	83%	**66%**

Table 2.2: Success rate for long-distance pushing experiment with 20 different object/goal configurations and short-distance experiment with 15 object/goal configurations. Success is defined as bringing the object closer than 15 pixels to the goal, which corresponds to around 7.5cm.

encountered in open-world settings, during execution a person knocks the object out of the robot's hand (in the 3rd frame). The experiment shows that visual MPC is able to naturally perform a new grasp attempt and bring the object to the goal. This trajectory is easier to view in the supplementary video.

In Figure A.3 in the appendix, the task is to push the bottle to the point marked with the green dot. In the beginning of the trajectory the object behaves differently than expected, it moves downwards instead of to the right. However the system recovers from the initial failure and still pushes the object to the goal.

The next question we investigate is: How much does tracking the target object using the learned registration matter for short horizon versus long horizon tasks? In this experiment, we disable the gripper control, which requires the robot to push objects to the target. We compare two variants of updating the positions of the designated pixel when using a pixel-distance based cost function. The first is a cost function that uses our registration-based method, trained in a fully self-supervised fashion, and the second is with a cost function that uses off-the shelf tracking from OpenCV [9]. Additionally we compare to visual MPC, which uses the video-prediction model's own prior predictions to update the current position of the designated pixel, rather than tracking the object with registration or tracking.

We evaluate our method on 20 long-distance and 15 short-distance pushing tasks. For long distance tasks the initial distance between the object and its goal position is 30cm while for short distance tasks it is 15cm. Table 2.2 lists quantitative comparisons showing that on the long distance experiment visual MPC using the registration-based cost not only outperforms prior work [47], but also outperforms the hand-designed, supervised object tracker [9]. By contrast, for the short distance experiment, all methods perform comparably. Thus, theses results demonstrate the importance of tracking the position of the target object for *long-horizon tasks*, while for short-horizon tasks object tracking appears to be irrelevant.

Evaluating Classifier-Based Cost Function

The goal of the classifier-based cost function is to provide an easy way to compute an objective for new tasks from a few observations of success for that task, so we compare our approach to alternative and prior methods for doing so under the same assumptions:

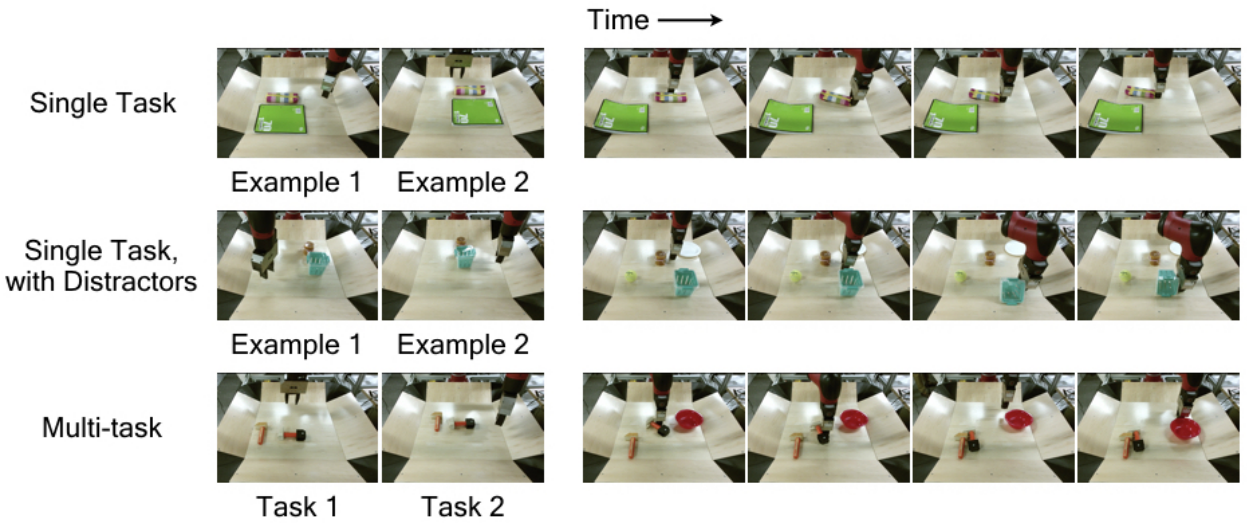


Figure 2.10: Object arrangement performance of our goal classifier with distractor objects and with two tasks. The left shows a subset of the 5 positive examples that are provided for inferring the goal classifier(s), while the right shows the robot executing the specified task(s) via visual planning.

pixel distance and latent space distance. In the latter, we measure the distance between the current and goal observations in a learned latent space, obtained by training an autoencoder (DSAE) [58] on the same data used for our classifier. Since we are considering a different form of task specification incompatible with user-specified pixels, we do not compare the classifier-based cost function to the cost function based on designated pixels.

To collect data for meta-training the classifier, we randomly select a pair of objects from our set of training objects, and position them in many different relative positions, recording the image for each configuration. Each task corresponds to a particular relative positioning of two objects, e.g. the first object to the left of the second, and we construct positive and negative examples for each task by labeling the aforementioned images. We randomly position the arm in each image, as it is not a determiner of task success. A good classifier should ignore the position of the arm. We also include randomly-positioned distractor objects in about a third of the collected images.

We evaluate the classifier-based cost function in three different experimental settings. In the first setting, the goal is to arrange two objects into a specified relative arrangement. The second setting is the same, but with distractor objects present. In the final and most challenging setting, the goal is to achieve two tasks in sequence. We provide positive examples for both tasks, infer the classifier for both, perform MPC for the first task until completion, followed by MPC for the second task. The arrangements of the evaluation tasks were chosen among the eight principal directions (N, NE, E, SE, etc.). To evaluate the ability to generalize to new goals and settings, we use novel, held-out objects for all of the task and distractor objects in our evaluation.

We qualitatively visualize the tasks in Figure 2.10. On the left, we show a subset of the five

Figure 2.11: Quantitative performance of visual planning for object rearrangement tasks across different goal specification methods: our meta-learned classifier, DSAE [58], and pixel error. Where possible, we include break down the cause of failures into errors caused by inaccurate prediction or planning and those caused by an inaccurate goal classifier.

images provided to illustrate the task(s), and on the left, we show the motions performed by the robot. We see that the robot is able to execute motions which lead to a correct relative positioning of the objects. We quantitatively evaluate the three cost functions across 20 tasks, including 10 unique object pairs. A task was considered successfully completed if more than half of the object was correctly positioned relative to the other. The results, shown in Figure 2.11, indicate that the distance-based metrics struggle to infer the goal of the task, while our approach leads to substantially more successful behavior on average.

Evaluating Multi-Task Performance

One of the key motivations for visual MPC is to build a system that can solve a *wide variety* of different tasks, involving completely different objects, physics and, objectives. Examples for tasks that can be solved with visual MPC are shown in Figure 2.1 and 2.12. Task 1 in Figure 2.1 shows a "placing task" where an object needs to be grasped and placed onto a plate while not displacing the plate. Task 2 is an object rearrangement tasks. The example shown in Task 4 and all examples in Figure 2.10 show relative object rearrangement tasks. Examples 5 and 6 show the same 3D object positioning tasks from different views. In Task 7, the goal is to move the black object to the goal location while avoiding the obstacle in the middle which is marked with a designated- and goal pixel. We also demonstrate that visual MPC – without modifications to the algorithm – solves tasks involving deformable objects such as a task where a towel needs to be wrapped around an object (Task 3), or folding a pair of shorts (Task 8). To the best of our knowledge this is the first algorithm for robotic manipulation handling both rigid and deformable objects. For a full illustration of each of these tasks, we encourage the reader to watch the supplementary video.

The generality of visual MPC mainly stems from two components — the generality of the visual dynamics model and the generality of the task definition. We found that the dynamics model often generalizes well to objects outside of the training set, if they have similar properties to the objects it was trained with. For example, Task 8 in Figure 2.12 shows the model predicting a pair of shorts being folded. We observed that a model, which

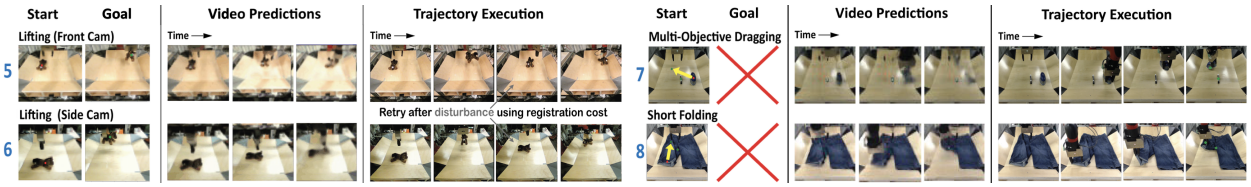


Figure 2.12: Visual MPC successfully solves a wide variety of tasks including multi-objective tasks, such as placing an object on a plate (row 5 and 6), object positioning with obstacle avoidance (row 7) and folding shorts (row 8). Zoom in on PDF.

was only provided videos of towels during training, generalized to shorts, although it had never seen them before. In all of the qualitative examples, the predictions are performed by the *same model*. We found that the model sometimes exhibits confusion about whether an object follows the dynamics of a cloth or rigid objects, which is likely caused by a lack of training data in the particular regime. To overcome this issue we add a binary token to the state vector indicating whether the object in the bin is hard or soft. We expect that adding more training data would remove the need for this indicator and allow the model to infer material properties directly from images.

The ability to specify tasks in multiple different ways adds to the flexibility of the proposed system. Using designated pixels, object positioning tasks can be defined in 3D space, as shown in Task 1 and 2 in Figure 2.1 and task 5-6 in Figure 2.12. When adding a goal image, the positioning accuracy can be improved by utilizing the registration scheme discussed in Section 2.5. For tasks where we care about *relative* rather than absolute positioning, a meta-learned classifier can be used, as discussed in Section 2.5.

Next, we present a quantitative evaluation to answer the following question: How does visual MPC compare to a hand-engineered baseline on a large number of diverse tasks? For this comparison, we engineered a simple trajectory generator to perform a grasp at the location of the initial designated pixel, lift the arm, and bring it to the position of the goal pixel. Camera calibration was performed to carry out the necessary conversions between image-space and robot work-space coordinates, which was not required for our visual MPC method. For simplicity, the baseline controller executes in open loop. Therefore, to allow for a fair comparison, visual MPC is also executed open-loop, i.e. no registration or tracking is used. Altogether we selected 16 tasks, including the qualitative examples presented earlier. The quantitative comparison is shown in Table 2.3, illustrating that visual MPC substantially outperforms this baseline. Visual MPC succeeded for most of the tasks. While the baseline succeeded for some of the cloth folding tasks, it failed for almost all of the object relocation tasks. This indicates that an implicit understanding of physics, as captured by our video prediction models, is indeed essential for performing this diverse range of object relocation and manipulation tasks, and the model must perform non-trivial physical reasoning beyond simply placing and moving the end-effector.

	% of Trials with Final Pixel Distance < 15
Visual MPC	**75%**
Calibrated Camera Baseline	18.75 %

Table 2.3: Results for a multi-task experiment of 10 hard object pushing and grasping tasks, along with 6 cloth folding tasks, evaluating using a single model. Values indicate the percentage of trials that ended with the object pixel closer than 15 pixels to the designated goal. Higher is better.

Discussion of Experimental Results

Generalization to many distinct tasks in visually diverse settings is arguably one of the biggest challenges in reinforcement learning and robotics today. While deep learning has relieved us from much of the problem-specific engineering, most of the works either require extensive amounts of labeled data or focus on the mastery of single tasks while relying on human-provided reward signals. From the experiments with visual MPC, especially the qualitative examples and the multi-task experiment, we can conclude that visual MPC *generalizes* to a wide range of tasks it has never seen during training. This is in contrast to many model-free approaches for robotic control which often struggle to perform well on novel tasks. Most of the generalization performance is likely a result of large-scale self-supervised learning, which allows to acquire a rich, task-agnostic dynamics model of the environment.

Chapter 3

RoboNet: Leveraging Data from Multiple Robots

3.1 Introduction

In the previous chapter we have introduced a model-based reinforcement learning algorithm to plan a sequence of actions by using a dynamics model trained on data collected by a single robot in a single environment. While the system is able to execute a range of pushing tasks, and even generalize to new objects, it is unable to generalize to new viewpoints, environments, or new robots. In this chapter we will extend the system to use more diverse data — data from multiple robots, backgrounds, and camera viewpoints to enable these generalization capabilities.

The background and motivation for this is as follows: Two of the most commonly raised criticisms of machine learning applied to robotics are the amount of data required per environment due to limited data-sharing, and the resulting algorithm's poor generalization to even modest environmental changes. A number of works have tried to address this by developing simulations from which large amounts of diverse data can be collected [152, 7], or by attempting to make robot learning algorithms more data efficient [33, 34]. However, developing simulators entails a deeply manual process, which so far has not scaled to the breadth and complexity of open-world environments. The alternative of using less real-world data often also implies using simpler models, which are insufficient for capturing the many details present in complex real-world environments such as object geometry or appearance.

Instead, we propose the opposite – using dramatically larger and more varied datasets collected in the real world. Inspired by the breadth of the ImageNet dataset [37], we introduce *RoboNet*, a dataset containing roughly 162,000 trajectories with video and action sequences recorded from 7 robots, interacting with hundreds of objects, with varied viewpoints and environments, corresponding to nearly 15 million frames. The dataset is collected autonomously with minimal human intervention, in a self-supervised manner, and is designed to be easily extensible to new robotic hardware, various sensors, and different collection poli-

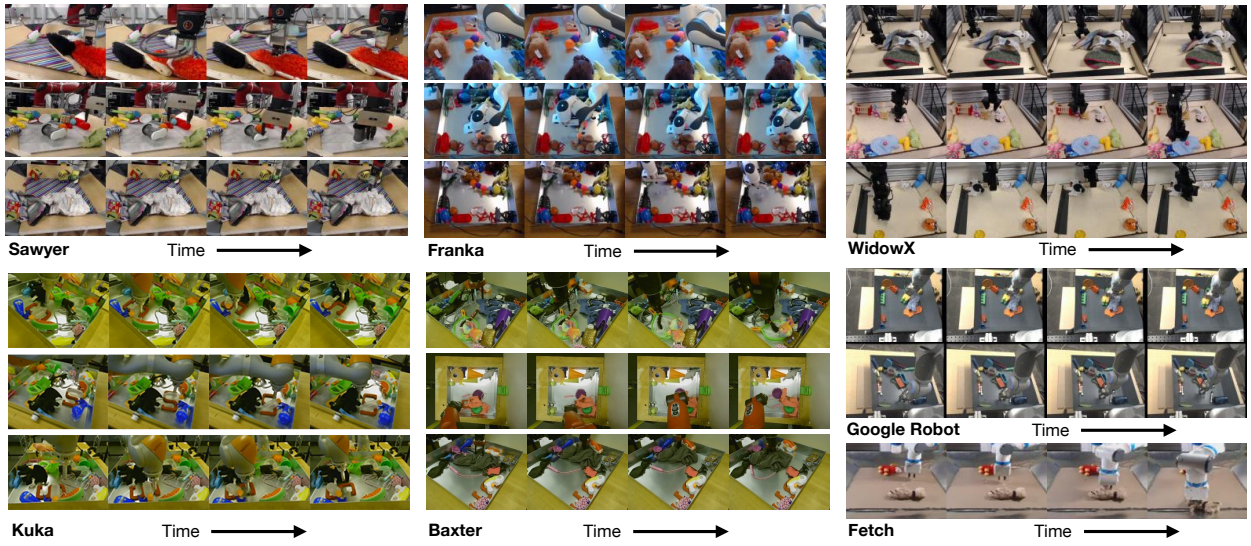


Figure 3.1: A glimpse of the RoboNet dataset, with example trajectories, robots, and viewpoints. We collected data with Sawyer, Franka, WidowX, Kuka, and Baxter robots, and augmented the dataset with publicly-available data from a robot from Google [55], a Fetch [202], and a Sawyer [49]. We use RoboNet to study the viability of large-scale data-driven robot learning, as a means to attain broad generalization across robots and scenes.

cies.

The common practice of re-collecting data from scratch for every new environment essentially means re-learning basic knowledge about the world — an unnecessary effort. In this work, we show that sharing data across robots and environments makes it possible to pre-train models on a large dataset of experience, thus extracting priors that allow for fast learning with new robots and in new scenes. If the models trained on this data can acquire the underlying shared patterns in the world, the resulting system would be capable of manipulating *any* object in the dataset using *any* robot in the dataset, and potentially even transfer to new robots and objects.

To learn from autonomously-collected data without explicit reward or label supervision, we require a self-supervised algorithm. To this end, we study two methods for sharing data across robot platforms and environments. First, we study the visual foresight algorithm [57, 49], a deep model-based reinforcement learning method that is able to learn a breadth of vision-based robotic manipulation skills from random interaction. Visual foresight uses an action-conditioned video prediction model trained on the collected data to plan actions that achieve user-specified goals. Second, we study deep inverse models that are trained to predict the action taken to reach one image from another image, and can be used for goal-image reaching tasks [2, 115]. However, when trained in a single environment, robot learning algorithms, including visual foresight and inverse models, do not generalize to large domain variations, such as different robot arms, grippers, viewpoints, and backgrounds, precluding

the ability to share data across multiple experimental set-ups and making it difficult to share data across institutions.

Our main contributions therefore consist of the RoboNet dataset, and an experimental evaluation that studies our framework for multi-robot, multi-domain model-based reinforcement learning based on extensions of the visual foresight algorithm and prior inverse model approaches. We show that, when trained on RoboNet, we can acquire models that generalize in zero shot to novel objects, novel viewpoints, and novel table surfaces. We also show that, when these models are finetuned with small amounts of data (around 400 trajectories), they can generalize to unseen grippers and new robot platforms, and perform better than robot-specific and environment-specific training. We believe that this work takes an important step towards large-scale data-driven approaches to robotics, where data can be shared across institutions for greater levels of generalization and performance.

3.2 Related Work

Deep neural network models have been used widely in a range of robotics applications [65, 205, 26, 204, 10, 78]. However, most work in this area focuses on learning with a single robot in a single domain, while our focus is on curating a dataset that can enable a single model to generalize to multiple robots and domains. The multi-task literature [36, 5], lifelong learning literature [165, 166], and meta-learning literature [53, 4] describe ideas that are tightly coupled with this concept. By collecting task-agnostic knowledge in wide variety of domains, a robotic system should be able to rapidly adapt to new, unseen environments using relatively little target domain data.

Large-scale, self-supervised robot learning approaches have adopted a similar viewpoint [142, 110, 68, 57, 204, 2, 136, 49]. Unlike these methods, we specifically consider transfer across multiple robots and environments, as a means to enable researchers to share data across institutions. We demonstrate the utility of our data by building on the visual foresight approach [57, 49], as it further enables generalization across tasks without requiring reward signals. This method is related to a range of recently proposed techniques that use predictive models for vision-based control [22, 167, 177, 100, 130]. Further, we also study how we can extend vision-based inverse models [2, 136, 202, 115] for generalizable robot-agnostic control.

A number of works have studied learning representations and policies that transfer across domains, including transfer from simulation to the real world [152, 169, 80], transfer across different dynamics [28, 203, 139, 7], transfer across robot morphologies with invariant feature spaces [69] and modularity [38], transfer across viewpoints through recurrent control [153], and transfer across objects [60, 79], tasks [44] or environments [29] through meta-learning. In contrast to these works, we consider transfer at a larger scale across not just one factor of variation, but across objects, viewpoints, tasks, robots, and environments, without the need to manually engineer simulated environments.

Outside of robotics, large and diverse datasets have played a pivotal role in machine learning. One of the best known datasets in modern computer vision is the ImageNet

dataset [37], which popularized an idea presented earlier in the tiny image dataset [171]. In particular, similar to our work, the main innovation in these datasets was not in the quality of the labels or images, but in their diversity: while prior datasets for image classification typically provided images from tens or hundreds of classes, the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) contained one thousand classes. Our work is inspired by this idea: while prior robotic manipulation methods and datasets [55, 196, 25, 68, 117, 49, 155] generally consider a single robot at a time, our dataset includes 7 different robots and data from 4 different institutions, with dozens of backgrounds and hundreds of viewpoints. This makes it feasible to study broad generalization in robotics in a meaningful way.

3.3 Data-Driven Robotic Manipulation

In this work we take a *data-driven* approach to robotic manipulation. We do not assume knowledge of the robot's kinematics, the geometry of objects or their physical properties, or any other specific property of the environment. Instead, basic common sense knowledge, including rigid-body physics and the robot's kinematics, must be implicitly learned purely from data.

Problem statement: learning image-based manipulation skills. We use data-driven robotic learning for the task of object relocation – moving objects to a specified location either via pushing or grasping and placing. However, in principle, our approach is applicable to other domains as well. Being able to perform tasks based on camera images alone provides a high degree of generality. We learn these skills using a dataset of trajectories of images $I_{0:T}$ paired with actions $a_{0:T}$, here T denotes the length of the trajectory. The actions are sampled randomly and need to provide sufficient exploration of the state space, which has been explored in prior work [49, 188]. This learning and data collection process is self-supervised, requiring the human operator only to program the initial action distribution for data collection and to provide new objects at periodic intervals. Data collection is otherwise unattended.

Preliminaries: robotic manipulation via prediction. We build on visual foresight [57, 49], a method based on an action-conditioned video prediction model that is trained to predict future images, up to a horizon h, from on past images: $\hat{I}_{t+1:t+h} = f(I_t, a_{t:t+h-1})$, using unlabeled trajectory data such as the data presented in the next section. The video prediction architecture used in visual foresight is a deterministic variant of the SAVP video prediction model [103] based heavily on prior work [55]. This model both predicts future images and the motion of pixels, which makes it straightforward to set goals for relocating objects in the scene simply by designating points $d_{0,i}$ (e.g., pixels on objects of interest), and for each one specifying a goal position $d_{g,i}$ to which those points should be moved. We refer to $d_{0,i}$ as designated pixels. These goals can be set by a user, or a higher-level planning algorithm. The robot can select actions by optimizing over the action sequence to find one that results in the desired pixel motion, then executing the first action in this sequence, observing a new image, and replanning. This effectively implements image-based model-predictive

control (MPC). With an appropriate choice of action representation, this procedure can automatically choose how to best relocate objects, whether by pushing, grasping, or even using other objects to push the object of interest. Full details can be found in Appendix B.1 and in prior work [49].

Preliminaries: robotic manipulation via inverse models. To evaluate RoboNet's usefulness for robot learning beyond use with the visual foresight algorithm, we evaluate a simplified version of the inverse model in [115]. Given context data, $\{\ldots, (I_{t-2}, a_{t-2}), (I_{t-1}, a_{t-1})\}$, the current image observation I_t, and a goal image I_{t+T}, the inverse model is trained to predict actions $a_t, \ldots, a_{t+T-1}$ (where T is a given horizon) that are needed to take the robot from the start to the goal image. Our experiments train a one-step inverse model where $T = 1$, which can be trained with supervised regression. At test time, the model takes as input 2 context frame/action pairs, the current image, and a goal image and then will predict an action which ought to bring the robot to the goal. This process is can be repeated at the next time-step, thus allowing us to run closed loop visual control for multiple steps.

3.4 The RoboNet Dataset

To enable robots to learn from a wide range of diverse environments and generalize to new settings, we propose RoboNet, an open dataset for sharing robot experience. An initial set of data has been collected across 7 different robots from 4 different institutions, each introducing a wide range of conditions, such as different viewpoints, objects, tables, and lighting. By having only loose specifications[1] on how the scene can be arranged and which objects can be used, we naturally obtain a *large amount of diversity*, an important feature of this dataset. By framing the data collection as a cross-institutional effort, we aim to make the diversity of the dataset grow over time. *Any research lab is invited to contribute to it.*

Data Collection Process

All trajectories in RoboNet share a similar action space, which consists of deltas in position and rotation to the robot end-effector, with one additional dimension of the action vector reserved for the gripper joint. The frame of reference is the root link of the robot, which need not coincide with the camera pose. This avoids the need to calibrate the camera, but requires any model to infer the relative positioning between the camera and the robots' reference frames from a history of context frames. As we show in Section 3.5, current models can do this effectively. The action space can also be a subset of the listed dimensions. We chose an action parametrization in end-effector space rather than joint-space, as it extends naturally to robot arms with different degrees of freedom. Having a unified action space throughout the dataset makes it easier to train a single model on the entire dataset. However, even with a consistent action space, variation in objects, viewpoints, and robot platforms has a substantial effect on how the action influences the next image.

[1] Specifications can be found here: `http://www.robonet.wiki`

In our initial version of RoboNet, trajectories are collected by applying actions drawn at random from simple hand-engineered distributions. We most commonly use a diagonal Gaussian combined the automatic grasping primitive developed in [46]. More details on the data collection process are provided in Appendix B.2.

The Diverse Composition of RoboNet

The environments in the RoboNet dataset vary both in robot hardware, i.e. robot arms and grippers, as well as environment, i.e arena, camera-configuration and lab setting, which manifests as different backgrounds and lighting conditions (see Figure 3.1 and 3.2). In theory, one could add any type (depth, tactile, audio, etc.) of sensor data to RoboNet, but we stick to consumer RGB video cameras for the purposes of this project. There is no constraint on the type of camera used, and in practice different labs used cameras with different exposure settings. Thus, the color temperature and brightness of the scene varies through the dataset. Object sets also vary substantially between different lab settings. To increase the number of tables, we use inserts with different textures and colors. To increase the number of gripper configurations, we 3D printed different finger attachments. We collected 104.4k trajectories for RoboNet on a Sawyer arm, Baxter robot, low-cost WidowX arm, Kuka LBR iiwa arm, and Franka Panda arm. We additionally augment the dataset with publicly available data from prior works, including 5k trajectories from a Fetch robot [202] and 56k trajectories from a robot at Google [55]. The full dataset composition is summarized in Table 3.1.

Using and Contributing to RoboNet

The RoboNet dataset allows users to easily filter for certain attributes. For example, it requires little effort to setup an experiment for training on all robots with a certain type of gripper, or all data from a Sawyer robot. An overview of the current set of attributes is shown in Table 3.1, and image examples are provided in Figure 3.2. We provide code infrastructure and common usage examples on the project website.[2]

Scripts for controlling common types of robots, for collecting data, and for storing data in a standard format are available on the project website. On the same webpage we are also providing a platform that allows anyone to upload trajectories. After data has been uploaded we will perform manual quality tests to ensure that the trajectories comply with the standards used in RoboNet: the robot setup should occupy enough space in the image, the action space should be correct, and the images should be of the right size. After passing the quality test, trajectories are added to the dataset. An automated quality checking procedure is planned for future work.

[2]The project webpage is at `http://www.robonet.wiki/`

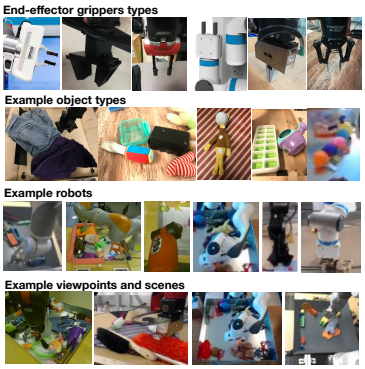


Figure 3.2: Qualitative examples of the various attributes in the RoboNet dataset.

Robot type	Sawyer (68k), Baxter (18k), WidowX (5k), Franka (7.9k), Kuka (1.8k), Fetch (5k) [202], GoogleRobot (56k) [55]
Gripper type	Weiss Robotics WSG-50, Robotiq, WidowX, Baxter, Franka, Kuka
Arena types	7
Arena inserts	10
Gripper configurations	10
Camera configuration	113
Lab environments	4

Table 3.1: Quantitative overview of the various attributes in the RoboNet dataset, including the 7 different robot arms and 7 different grippers.

3.5 Robot-Agnostic Visual Control: Model Training and Experiments

A core goal of this paper is to study the viability of large-scale data-driven robot learning as a means to acquire broad generalization, across scenes, objects, and even robotic platforms. To this end, we design a series of experiments to study the following questions: (1) can we leverage RoboNet to enable zero-shot generalization or few-shot adaptation to novel viewpoints and novel robotic platforms? (2) how does the breadth and quantity of data affect generalization? (3) do predictive models trained on RoboNet memorize individual contexts or learn generalizable concepts that are shared across contexts? Finally, we evaluate a simple inverse model to test if RoboNet can be used with learning algorithms other than visual foresight.

Visual Foresight: Experimental Methodology

For our visual foresight robot experiments, we evaluate models in terms of performance on the object relocation tasks described in Section 3.3. A task is defined as moving an object not in the training set to a particular location in the image. After running the learned policy or planner, we measure the distance between the achieved object position and the goal position. We judge a task to be successful if the operator judges the object is mostly covering the goal location at the end of the rollout. Models within an experiment are compared on the same set of object relocation tasks. We use this evaluation protocol through the rest of the experiments. Please refer to Appendix B.4 for some images of the testing environments.

Note that results should not be compared across different experiments, since task difficulty varies across robots and human operators.

Visual Foresight: Zero-Shot Generalization to New Viewpoints and Backgrounds

In this section, we study how well models trained on RoboNet can generalize, without any additional data, to novel viewpoints and held-out backgrounds with a previously seen robot. Generalizing to a new viewpoint requires the model to implicitly estimate the relative positioning and orientation between the camera and the robot, since the actions are provided in the robot's frame of reference. We attempt five different object relocation tasks from two views in order to compare a model that has been trained on 90 different viewpoints against a model that was only trained on single viewpoint. The arrangement of the cameras is shown in Appendix B.4. In Table 3.2, we show object relocation accuracy results for both of these models when testing on both the seen viewpoint (left) and a novel viewpoint (right). The results show that the model trained on varied viewpoints achieves lower final distance to the goal on the benchmark tasks for *both* views, thus illustrating the value of training on diverse datasets.

We tested the same multi-view model on a similar set of tasks in an environment substantially different from the training environment. In Figure 3.3 we show a successful execution of a pushing task in this new environment. The multi-view model achieves an average final distance of 14.4 ± 2 cm (std. error) in the new setting. This performance is comparable to that achieved by the multi-view model in a novel viewpoint, which suggests the model is also able to effectively generalize to novel surroundings.

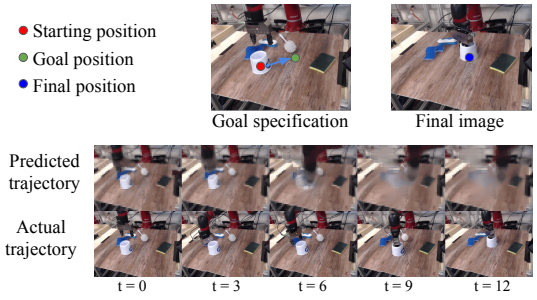


Figure 3.3: Zero-shot generalization to new backgrounds with a model trained across multiple views.

	Avg. dist. (cm) seen view	Avg. dist. (cm) held-out view
single view	14.8 ± 3.8	23.2 ± 2.6
multi-view	**9 ± 2.2**	**16.2± 2.9**

Table 3.2: Evaluation of viewpoint generalization, showing the average distance to the goal after executing the action sequence and standard error. A model trained on multiple views can better generalize to a new viewpoint.

Visual Foresight: Few-Shot Adaptation to New Robots

When evaluating on domains that differ more substantially from any domain present in the dataset, such as settings that contain an entirely new robotic arm, zero-shot generalization

is not possible. In this section, we evaluate how well visual foresight can adapt to entirely new robots that were not shown to the model during training. This is one of the most challenging forms of generalization, since robots have not only different appearances, but also different dynamics when interacting with objects, different kinematics, and different work-space boundaries.

To test our hypothesis, we collect a small number (300-400) of random trajectories from the target robot environment. Models are then pre-trained on the entirety of RoboNet, but holding out the data from the target robot. These models are then fine-tuned using the aforementioned collected trajectories. We compare to a separate model that is trained from scratch on those trajectories. Additionally, for the Franka experiments another model is trained on all the Franka data in RoboNet, and for the Baxter experiment one model is pre-trained on just Sawyer data in RoboNet and fine-tuned to Baxter. The R3 and Fetch were also not included in the pre-training data due to computational constraints.

Kuka Experiments	Success rate
Random Initialization Train on N=400	10%
Random Initialization Train on N=1800	30%
Pre-train on RoboNet w/o Kuka, R3, Fetch Finetune on N=400	**40%**

Table 3.3: Results for adaptation to an unseen Kuka robot. The model pre-trained on RoboNet without the Kuka, R3, and Fetch data, achieves the best performance when fine-tuned with 400 trajectories from the test robot.

Franka Experiments	Success rate
Random Initialization Train on N=400	20%
Random Initialization Train on N=8000	35%
Pre-train on RoboNet w/o Franka, R3, Fetch Finetune on N=400	**40%**

Table 3.4: Results for adaptation to an unseen Franka robot. The model pre-trained on RoboNet without the Franka, R3, and Fetch data, achieves the best performance when fine-tuned with 400 trajectories from the test robot.

Baxter Experiments	Success rate
Random Initialization Train on N=300	33%
Pre-train on Sawyer Finetune on N=300	**83%**
Pre-train on RoboNet w/o Baxter Finetune on N=300	58%

Table 3.5: Evaluation results for adaptation to an unseen Baxter robot. The model pre-trained on RoboNet's Sawyer data, achieves the best performance when fine-tuned with 300 trajectories from the test robot.

The quantitative results are summarized in Table 3.3, Table 3.4, and Table 3.5. The results show that RoboNet pre-training provides substantial improvements over training from scratch, on all three test robots. In the Kuka and Franka experiments, a model fine-tuned on *just 400 samples* is able to outperform its counterpart trained on all of RoboNet's data from the respective robot. These results suggest that RoboNet pre-training can offer large advantages over training tabula rasa, by substantially reducing the number of samples needed in a new environment. Figure 3.4 shows a successful rollout of visual foresight on a challenging task of positioning a plastic cup to a desired location.

In the Baxter experiment, we also find that pre-training on specific subsets of RoboNet (in this case the Sawyer, which is visually more similar to the Baxter than other robots) can perform significantly better than training on the entire dataset. Hence, this experiment (as well as the Robotiq gripper generalization experiment in Appendix B.5) demonstrates that increased diversity during pre-training can sometimes hurt performance when compared to pre-training on a subset of RoboNet. We hypothesize that more specific pre-training works better, because our models under-fit when trained on all of RoboNet, which we study in more detail in the next section.

Visual Foresight: Model Capacity Experiments

When training video prediction models on RoboNet, we observe clear signs of underfitting. Training error and validation error are generally similar, and both plateau before reaching very high performance on the training sequences. During test time, inaccurate predictions are often the cause of poor performance on the robot. Thus, we perform an additional experiment to further validate the underfitting hypothesis. We train two large models, using a simplified deterministic version of the network architecture presented in [174], on RoboNet's Sawyer data: one model has 200M parameters and the other has 500M parameters. The 200M parameter model has 0.104 ± 0.057 average ℓ_1 per-pixel error on a held out test set, whereas the 500M model has 0.0847 ± 0.045 ℓ_1 per-pixel error. These results suggest that current visual foresight models – even ones much larger than the 5M - 75M parameter models used in our control experiments – suffer from underfitting, and future research on higher capacity models will likely improve performance.

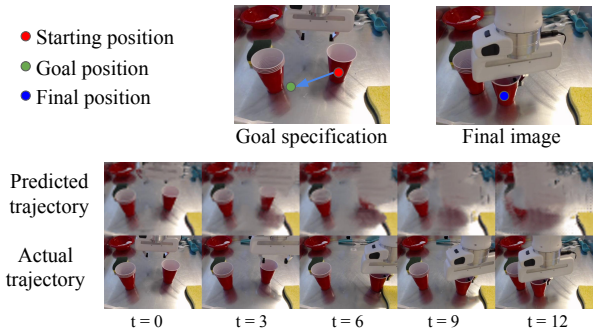


Figure 3.4: Example task of grasping and moving a thin plastic cup with the Franka robot, using visual foresight pre-trained on RoboNet w/o Franka and fine-tuned on 400 trajectories from the Franka robot.

Inverse Model: Multi-Robot and Multi-Viewpoint Reaching

To evaluate RoboNet's applicability to different control algorithms, we train a simple version of the inverse model from [115] (refer to Section 3.3 for details) on a subset of RoboNet containing only Sawyer and Franka data. The *same* model is evaluated on both robots: the Sawyer experiments also contain a held-out view. We evaluate model per-

Inverse Model	Success
Sawyer Reaching Front View	4/5
Sawyer Reaching Unseen View	5/5
Franka Reaching Front View	4/5

Table 3.6: Inverse model results on 5 reaching tasks.

formance on simple reaching tasks. Tasks are constructed by supplying a goal image, by taking an image of the gripper in a different reachable state. After task specification, the model runs continuously, re-planning each step until a maximum number of steps is reached. Success is determined by a human judge. This model is able to perform visual reaching tasks on both robots, including from a novel viewpoint not seen during training. However, because of its comparatively greedy action selection procedure, we observe that it tends to perform poorly on more complex tasks that require object manipulation.

3.6 Discussion

We presented RoboNet, a large-scale and extensible database of robotic interaction experience that combines data from 7 different robots, multiple environments and backgrounds, over a hundred camera viewpoints, and four separate geographic locations. We demonstrated two example use-cases of the dataset by (1) applying the visual foresight algorithm [49] and (2) learning vision-based inverse models. We evaluated generalization across many different experimental conditions, including varying viewpoints, grippers, and robots. Our experiments suggested that fine-tuning models pretrained on RoboNet offers a powerful way to quickly allow robot learning algorithms to acquire vision-based skills on unseen robot hardware.

Our experiments further found that video prediction models with $\leq$ 75M parameters tend to heavily *underfit* on RoboNet. While much better, we even observe underfitting on 500M-parameter models. As a result, prediction models struggle to take advantage of the breadth and diversity of data from multiple robots, domains, and scenes, and instead seem to perform best when using a subset of RoboNet that looks most similar to the test domain. This suggests two divergent avenues for future work. On one hand, we can develop algorithms that automatically select subsets of the dataset based on various attributes in a way that maximizes performance on the test domain. In the short term, this could provide considerable improvements with our current models. However, an alternative view is to instead research how to build more flexible models and policies, that are capable of learning from and larger and more diverse datasets across many robots and environments. We hope that the RoboNet dataset can serve as a catalyst for such research, enabling robotics researchers to study such problems in large-scale learning. Next, we discuss limitations of the dataset and evaluation, and additional directions for future work.

Limitations. While our results demonstrated a large degree of generalization, a number of important limitations remain, which we aim to study in future work. First and foremost, the tasks we consider are relatively simple manipulation tasks such as pushing and pick-and-place, with relatively low fidelity. This is an important limitation that hinders the ability of these models to be immediately of practical use. However, there are a number of promising recent works that have demonstrated how predictive models of observations can be used

for solving tasks of greater complexity such as tool use [188] and rope manipulation [100], and tasks at greater fidelity such as block mating [130] and die rolling [167]. Further, one bottleneck that likely prevents better performance is the quality of the video predictions. We expect larger, state-of-the-art models [178, 174] to produce significantly better predictions, which would hopefully translate to better control performance.

Another limitation of our current approach and dataset is the source of data being from a pre-determined random policy. This makes data collection scalable, but at the cost of limiting more complex and nuanced interactions. In future work, we plan to collect and solicit data from more sophisticated policies. This includes demonstration data, data from modern exploration methods that scale to pixel observations [11, 20, 135], and task-driven data from running reinforcement learning on particular tasks. As shown in prior work [188], improving the forms of interactions in the dataset can significantly improve performance.

In selecting how and where to collect additional data, our experiments suggest that adaptation to new domains is possible with only modest amounts of data, on the order of a few hundred trajectories. This suggests that prioritizing variety, i.e. small amounts of data from many different domains, is more important than quantity in future collection efforts.

Future Directions. This work takes the first step towards creating learned robotic agents that can operate in a wide range of environments and across different hardware. While in this work, we explored two particular classes of approaches, we hope that RoboNet will inspire the broader robotics and reinforcement learning communities to investigate how to scale model-based *or* model-free RL algorithms to meet the complexity of the real world, and to contribute the data generated from their experiments back into a shared community pool. In the long term, we believe this process will iteratively strengthen the dataset, and thus allow the algorithms derived from it to achieve greater levels of generalization across tasks, environments, robots, and experimental set-ups.

Part II

Large-Scale Model-Free Robot Learning

Chapter 4

Bridge Data: Imitation Across Tasks and Domains

4.1 Introduction

In Part I we have addressed how we can construct a model-based reinforcement learning system that can generalize to new objects, and be fine-tuned to new robots, and scenes. However the system is not able to learn new tasks, outside the scope of pushing and simple pick-place motions, due the fact that it collects data with random scripted motions. In Part II we will introduce imitation learning and offline reinforcement learning-based methods that leverage large amounts of human demonstration data which contains a variety of more complex tasks such as turning a faucet or flipping a pot upright. Data for such tasks would be impossible to collect with random scripted motions.

Humans and animals can generalize a learned skill to a wide variety of contexts without needing to relearn the skill every time. Endowing robots with the same capability would be a significant advance toward making robots more applicable to a range of real-world settings. However, the prevailing paradigm of robot learning is to repeat data collection and policy training from scratch for every new task and environment. Learning policies in isolation not only increases the costs of data collection, but also limits the policy's scope of generalization.

In other fields, such as computer vision [95] and natural language processing (NLP) [39], utilizing large, diverse datasets has shown considerable success in enabling generalization to new problems or domains with a small amount of data (e.g., via pretraining and finetuning). However, in robotics, datasets are usually collected with a specific robotic platform and domain in mind, typically by the same researcher who intends to use that dataset. What would it take to make datasets reusable in robotics in the same way as large supervised datasets are reused (e.g., ImageNet [37])? Each end-user of such a dataset might want their robot to learn a different task, which would be situated in a different domain (e.g., a different laboratory, home, etc.). It is currently an open question whether such reuse is feasible in robotics, and we posit that any such dataset would need to cover both multiple different

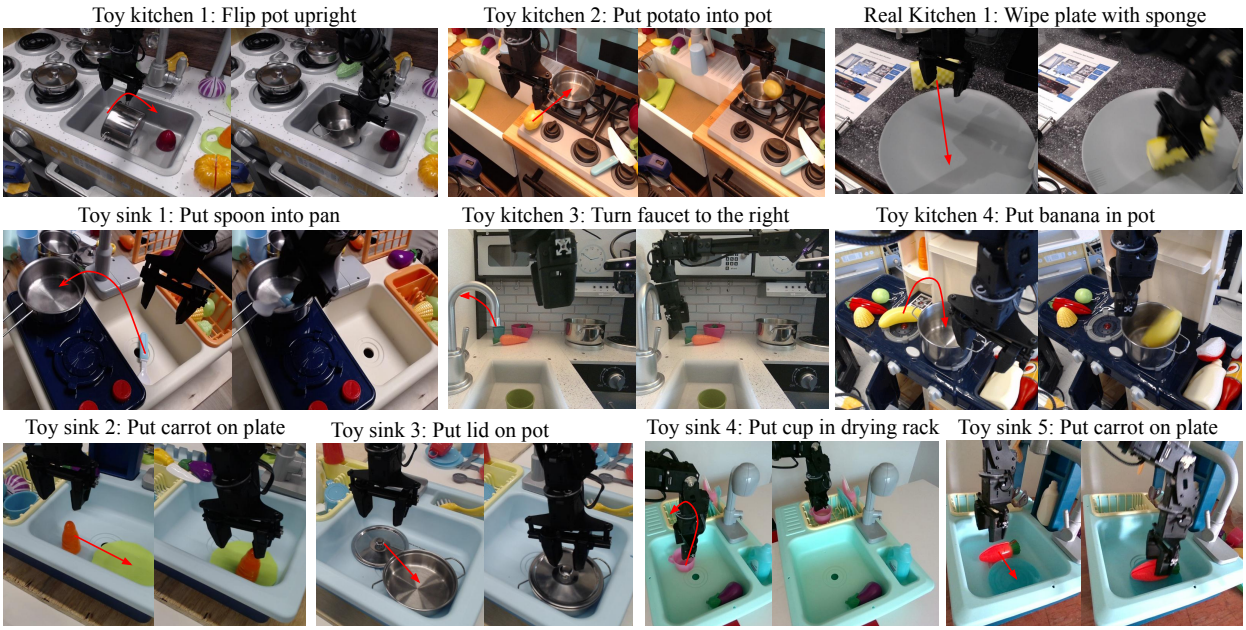


Figure 4.1: Illustration of our bridge dataset. The dataset includes demonstrations in 10 environments (4 toy kitchens and 5 toy sinks and 1 real kitchen), collected using a WidowX250 robot controlled via an Oculus Quest2 VR device, and consists of 7200 demonstrations. The red arrows indicate the desired movement of the target object.

tasks and multiple different domains. To this end, the aim of our paper is to investigate the degree to which such a multi-task and multi-domain dataset, which we refer to as a *bridge* dataset, can enable a new robot in a new domain (which was not seen in the bridge data) to more effectively generalize when learning a new task (which was also not seen in the bridge data), as well as to transfer tasks from the bridge data to the target domain. We also propose a new dataset that enables this goal in the context of kitchen-themed tasks with a low-cost robotic arm and is intended to be reused by other researchers.

The notion that multi-task data can speed up learning or improve generalization has been studied in many prior works [198, 85]. However, unlike this paper, the focus in these prior works, as we discuss in Section 4.2, is not on enabling new users to quickly train generalizable skills in a new setting or domain, but rather to utilize multi-task learning to lower the data requirements of acquiring a pre-defined set of tasks. More closely related to our work, RoboNet [31] contains data from multiple robots and domains, but this data is collected using random motions, and does not provide examples of multiple different tasks that can be used for more complex task-directed manipulation. We discuss other datasets in Section 4.2; but in summary, no existing dataset covers *both* multiple tasks *and* multiple domains in a way that is suitable to study our central hypothesis: can prior data be used to improve the generalization of *new* tasks in *new* domains? We will call this the *bridge data hypothesis*.

We believe this is a critical requirement for effective data reuse in robotics, where different labs and researchers can all bootstrap from the same shared datasets. To study this, we collected a new multi-domain manipulation dataset with 7,200 demonstrations of 71 distinct and semantically meaningful tasks, themed around household tasks in kitchen environments. The data was collected across 10 distinct "toy" kitchens, as shown in Figure 4.1. This data is suitable for imitation learning, which is the focus of our work, though it could also be repurposed for offline RL and other algorithms in the future. We present our new dataset, and then use it to evaluate the bridge data hypothesis that is stated above, using three types of transfer scenarios: **(1)** When the user needs to train an existing task in a new domain, does the inclusion of bridge data boost performance? This roughly corresponds to a standard domain adaptation setting. **(2)** After the user has collected some data for a few tasks in a new domain, can their robot then perform other tasks that were *not* seen in the new domain, but are only present in the bridge data (i.e., can it "import" tasks from the bridge data)? **(3)** When the user collects some data in a new domain for a task that was *not* seen in the bridge data, can the performance and generalization of this task be boosted by including the bridge data in training? Scenario **(3)** directly evaluates our central hypothesis, while the other scenarios illustrate other potential uses for bridge data.

The main contributions of our work consist of an empirical evaluation of the bridge data hypothesis and a practical example of a bridge dataset with 7,200 demonstrations for 71 tasks in 10 environments, which we have released publicly on the project website[1]. To the best of our knowledge, our work is also the first to demonstrate transfer scenarios **(2)** and **(3)** above. This is significant, because **(2)** provides users with a low-cost way to "import" all of the skills in the bridge dataset into their own domain with just a small number of demonstrations in their domain, while **(3)** provides for a way to boost the performance of an entirely new skill with previously collected reusable bridge data. Our results suggest that accumulating and reusing diverse multi-task and multi-domain datasets, at least when all data is collected with the same type of robot, may make it possible for researchers to endow robots with generalizable skills using only a modest amount of in-domain data for their desired task.

4.2 Related Work

While most prior work on deep visuomotor learning trains a single task in a single domain [60, 44, 77, 201, 113, 154, 66, 168, 208], our goal is not to develop better learning methods, but rather to illustrate how generic multi-domain, multi-task datasets can be used with existing algorithms to boost the generalization of *new* tasks in *new* domains. Prior work on multi-task reinforcement learning [85] has shown that data from other tasks can boost generalization of new tasks, however this study is carried out in a *single* domain.

Existing robot learning datasets do not exhibit the right properties for boosting the generalization of new tasks in *new* domains or zero-shot transferring skills from the prior

[1] `https://sites.google.com/view/bridgedata`

Dataset	# Tasks	# Trajec.	# Domains	suitable for BC/IL
DAML [201]	> 100	2.9k	1	✓
MIME [155]	22	8.2k	1	✓
RoboNet [31]	N/A	162k	7	✗
RoboTurk [117, 118]	3	2.1k	1	✓
Vis. Imit. Made [195]	2	2k	50	✓
Ours	**71**	7.2k	10	✓

Figure 4.2: Comparison of our dataset and prior works. Our dataset has by far the most tasks, and is the only dataset with more than 2 tasks that has many domains. This is critical for evaluating the bridge data hypothesis.

dataset to a target domain. We provide an overview of the most related datasets in Figure 4.2. Most existing robot datasets, such as MIME [155], DAML [201], RoboTurk [117, 118], and many others [142, 57, 110, 87, 50, 206, 85] only feature a single domain, making them difficult to use for boosting the generalization in *other* domains. Merging multiple existing datasets into one multi-domain dataset is difficult due to inconsistencies in data collection protocols, time discretization, robot morphologies, and sensors. Learning from multiple robots has been studied with RoboNet [31], which provides a dataset with 7 different robots in different domains. Here the data is generated with random motions which do not produce semantically meaningful tasks. This limits task complexity to pushing and basic grasping, and makes the data poorly suited for imitation learning.

Some prior works have also used datasets collected by humans without a robot, across multiple domains. For example, Young et al. [195] presents results on data across many more domains than our bridge data, collected via a hand-held gripper, but only presents two grasping tasks.

4.3 Bridge Datasets

In this section, we describe the basic principles behind bridge datasets and how they can be used to boost generalization. Then, we present a description of the specific bridge dataset that we collected using teleoperation of a low-cost robotic arm for a range of kitchen-themed manipulation tasks. We use the term *bridge dataset* to refer to a large and diverse dataset of robotic behaviors collected in a range of settings (e.g., different viewpoints, lighting conditions, objects, and scenes), for a range of *different* tasks, so as to make it possible to "bridge" gaps in the generalization that arise when the user provides a small to medium amount of data in their specific target domain. We define the term "target domain" to refer to the environment where the robot must perform the desired task. This target domain is distinct from *any* of the settings seen in the bridge dataset: the intent is for the same large bridge dataset to be used by all users for whichever target domain they require.

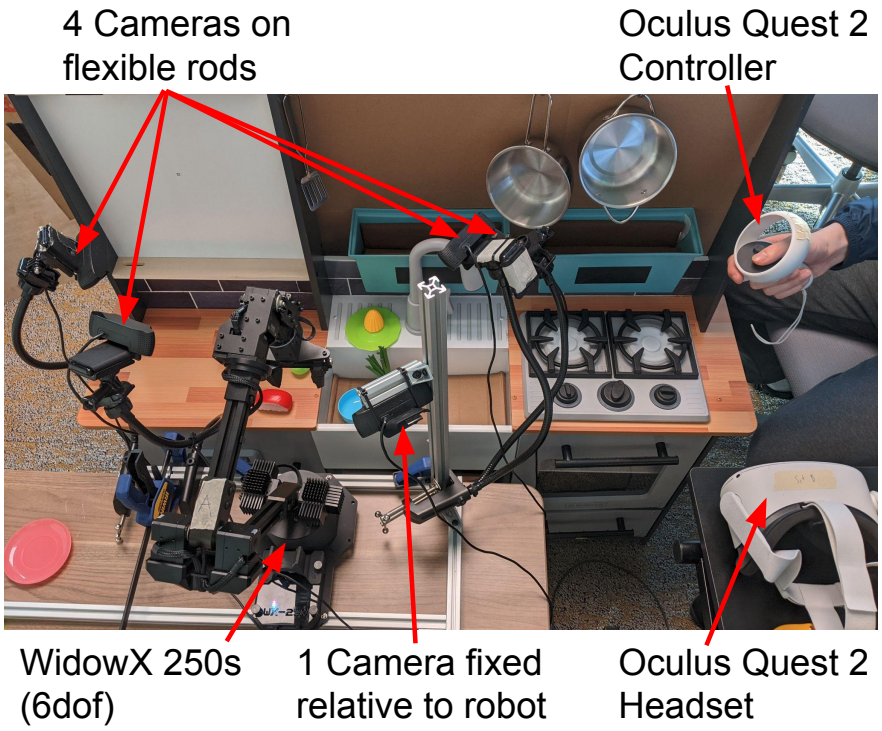


Figure 4.3: Demonstration data collection setup using VR Headset. The scene is captured by 5 cameras simultaneously. While one of the cameras is fixed, the others are mounted on flexible rods.

Boosting Generalization via Bridge Datasets

We consider three types of generalization in our experiments, though other modes may also be feasible:

(1) Transfer with matching behaviors, where the user collects some small amount of data in their target domain for tasks that are also present in the bridge data (e.g., around 50 demos per task), and uses the bridge data to boost the performance and generalization of these tasks. We illustrate this scenario in Figure 4.4. This scenario is the most conventional, and resembles domain adaptation in computer vision, but it is also the most limiting, since it requires the user's desired tasks to be present in the bridge data. However, as we will show, bridge data can enable very significant performance and generalization boosts in this setting.

(2) Zero-shot transfer with target support, where the user utilizes data from a few tasks in their target domain to "import" other tasks that are present in the bridge data *without* additionally collecting new demonstrations for them in the target domain. For example, the bridge data contains the tasks of putting a sweet potato into a pot or a pan, the user provides data in their domain for putting brushes in pans, and the robot is then

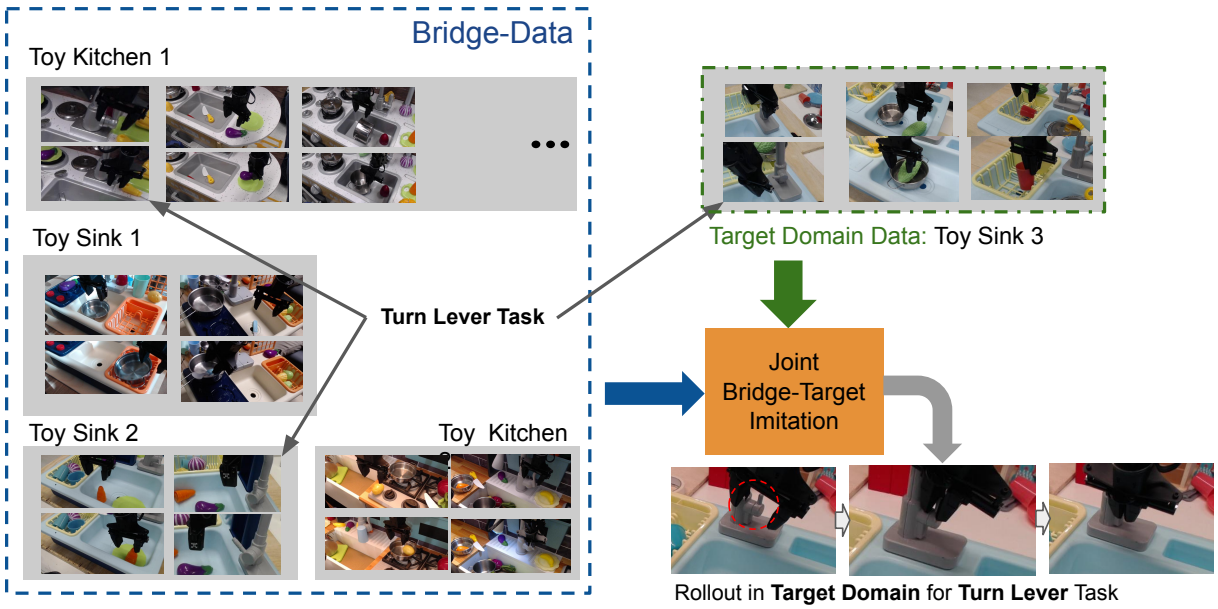


Figure 4.4: Scenario (1): transfer with matching behaviors. In this setting, bridge data is used to improve the performance and generalization of tasks in the target domain for which the user has collected some amount of data. These tasks must also be present in the bridge data. In this example, the user demonstrates the "turn lever," "squash into pot," and "flip cup" tasks in the target domain, and these tasks are also present in several domains in the bridge data. After including the bridge data in training, the performance and generalization of these tasks in the target is significantly higher.

able to *both* put brushes as well as put sweet potatoes in pans. We illustrate this scenario in Figure 4.5. This scenario increases the repertoires of skills that are available in the user's target environment, simply by including the bridge data, thus eliminating the need to recollect data for every task in every target environment.

(3) Boosting generalization of new tasks, where the user provides a small amount of data (50 demonstrations in practice) for a new task that is not present in the bridge data, and then utilizes the bridge data to boost generalization and performance of this task. This scenario, illustrated in Figure 4.6, most directly reflects our primary goals, since it uses the bridge data without requiring *either* the domains or tasks to match, leveraging the diversity of the data and structural similarity to boost performance and generalization of entirely new tasks.

To enable this kind of generalization boosting, we conjecture that the key features that bridge datasets must have are: (i) a sufficient variety of settings, so as to provide for good generalization; (ii) shared structure between bridge data domains and target domains (i.e., it is unreasonable to expect generalization for a construction robot using bridge data of kitchen

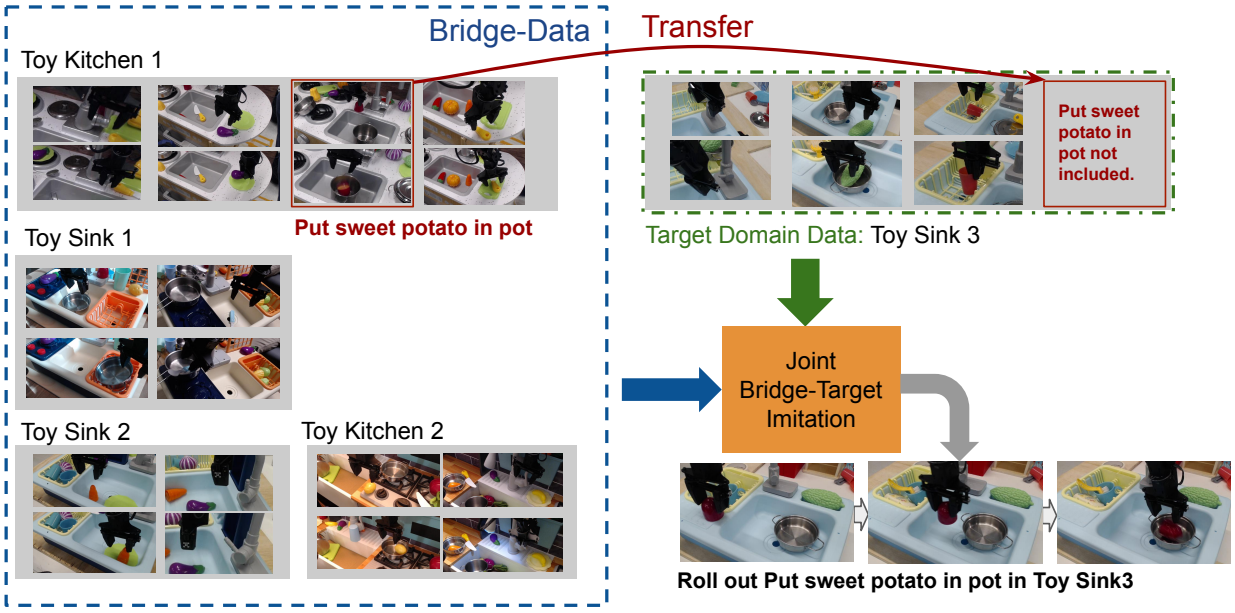


Figure 4.5: Scenario (2): zero-shot transfer with target support. In this setting, the goal is to "import" a task from the bridge data that was *not* seen in the target domain. The user provides a few tasks in the target domain that are used to connect to the bridge data, and then asks the robot to perform a task that they did not provide, but which was seen in the bridge data. In this case, the "put sweet potato in pot" task is present in the toy kitchen 1 domain in the bridge data, but is *not* demonstrated by the user in the target domain. After training with user-provided data for other tasks, the robot is able to perform "put sweet potato in pot" in the target domain.

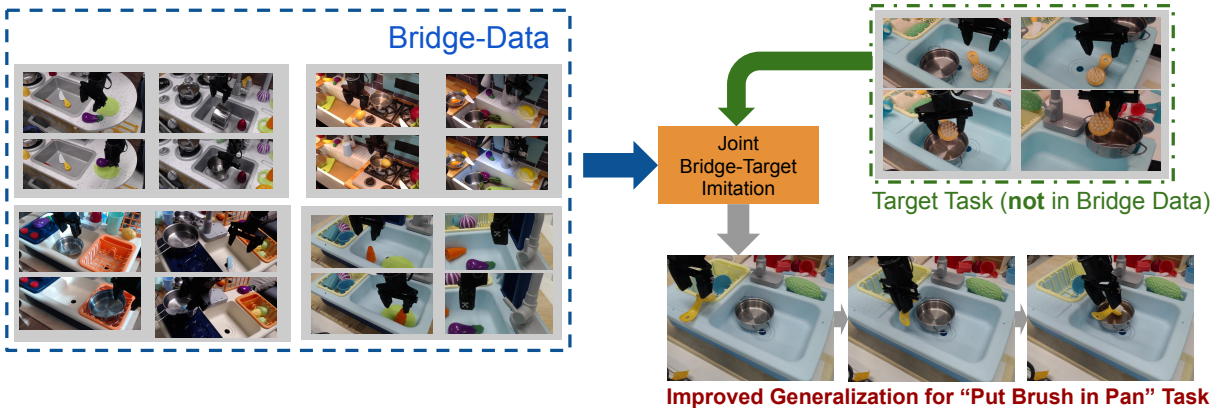


Figure 4.6: Scenario (3): boosting generalization of new tasks. The user provides some data for a *new* task that was not seen in the bridge data, and the bridge data is included in training to boost performance and generalization for this new task.

tasks); (iii) a sufficient range of tasks that breaks unwanted correlations between tasks and domains. Analogously to how the ImageNet dataset [37] provides broad coverage that makes it possible to boost generalization for a range of computer vision tasks, the broader a bridge dataset is, the more likely target tasks receive a generalization boost in a particular target domain.

A Bridge Dataset of Large-Scale Kitchen Tasks

We instantiate a bridge dataset based on the principles above as follows:
Robotic system overview. Since our dataset is likely the most useful for users with the same or similar type of robot, we chose to use a low-cost and widely available robot, a 6-dof WidowX250s (US$2900), which many other users of our dataset are likely to be able to obtain. The total cost of the setup is less than US$3600 (excluding the computer). To collect demonstrations, we use an Oculus Quest headset, where we put the headset on a table as illustrated in Figure 5.1 next to the robot and track the user's handset while applying the user's motions to the robot end-effector via inverse kinematics. We capture images from 3 to 5 cameras concurrently, using standard webcams as well as Intel RealSense depth cameras.
Data collection protocol. Our proposed bridge dataset, illustrated in Figure 4.1 consists of a total of 7200 demonstrations for 71 different tasks, collected in 10 different environments, focusing on the theme of household kitchen tasks. Each task has between 50 and 300 demonstrations. We opted to use kitchen and sink "play sets" for children, since they are smaller than real-world kitchens and therefore ideal for small-scale robots, and they are comparatively robust and low-cost, while still providing settings that resemble typical household scenes. During data collection we randomize the kitchen position (translations of 0-20cm) and the camera positions (translations of 0-10cm and rotations of 0-30 degrees) for all cameras on flexible rods every 25 trajectories. The positions of distractor objects (i.e. objects not needed for a task) are randomized at least every 5 trajectories. All environments except toy sink 4, toy sink 5, and kitchen 3 were collected at Institution 1 and use Logitech C920 webcams, the three remaining environments were collected at Institution 2 and use Intel RealSense RGB-D cameras. The trajectories collected at Institution 2 randomize all camera positions once every 50 trajectories. Instructions for how users can reproduce our setup and collect data in new environments can be found on the project website.[2]

4.4 Using Bridge Data in Imitation Learning

As a proof-of-concept to illustrate the utility of bridge datasets for boosting generalization in robot learning, we will present experimental results for an imitation-based approach that utilizes this data, although the data could also be used with a variety of other robotic learning algorithms such as offline RL and model-based planning.

[2] `https://sites.google.com/view/bridgedata`

Incorporating bridge data. While a variety of transfer learning methods have been proposed in the literature for combining datasets from distinct domains, we found that a simple joint training approach is effective for deriving considerable benefit from bridge data. For each of the scenarios outlined in Section 4.3, we take the user-provided demonstrations in the target domain and combine them with the entire bridge dataset for training. Since the sizes of these datasets are significantly different, we rebalance the datasets by weighting each datapoint, as discussed at the end of this section. Imitation learning then proceeds normally, simply training the policy with supervised learning on the combined dataset using the architecture described in the following paragraph. It is also possible to incorporate bridge data in other ways, for example by pretraining and finetuning. We found pretraining to be significantly less effective than joint training in our experiments, a finding that is consistent with prior works [110], but we emphasize that bridge datasets can be combined with target domain data in a variety of ways. **Policy architecture.** We use task-conditioned behavioral cloning (BC) with an additional task-id input to the policy, which is used to distinguish tasks during training and testing. In some cases, a task cannot be uniquely determined by only observing the input image, and a one-hot vector representing the task will solve this issue. The images are first fed into a 34-layer ResNet [73] and the resulting feature maps are passed through a spatial softmax [59, 108], which extracts a set of spatial positions of the relevant features. The spatial features are then concatenated with the one-hot task-id vector, and are fed into 3 layers of fully-connected networks by which the final action prediction is produced. During training, for a batch of training data containing tuples of task ids, images, and ground-truth actions, the network is trained by minimizing the standard ℓ_2-error between the ground-truth actions and the predicted actions given by the policy provided the task id and the image observation as the input.

Training details. Since the amount of target domain data is usually significantly less than the amount of bridge data, we rebalance the two datasets during training. In the matching behaviors and zero-shot transfer with target support scenarios, the ratio between the number of trajectories in the bridge and target data is roughly 10:1, and we rebalance the data such that 70% of the dataset is bridge data and 30% is target domain data. In the "boosting generalization of new tasks" scenario the imbalance is more severe, roughly 60:1, and so we rebalance such that 90% of the dataset is bridge data and 10% is target domain data. Lower rebalancing ratios of bridge data and target domain data tend to produce overfitting when the amount of target domain data is as low as 50 demonstrations.

4.5 Experimental Results

Our experimental evaluation aims to study how well bridge data can facilitate generalization in scenarios **(1)**, **(2)**, and **(3)**, as outlined in Section 4.3. We utilize the bridge dataset described in Section 4.3. We evaluate generalization on a set of new target domains with limited target domain data for each of the generalization scenarios, and compare the performance of learned policies with and without bridge data. Videos of the experi-

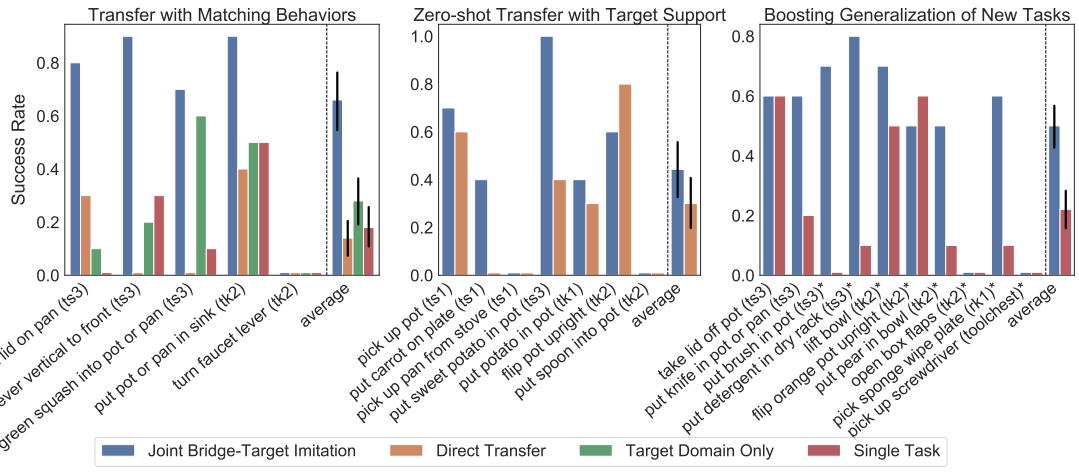


Figure 4.7: Comparisons of joint training with bridge data (blue) and other approaches for each type of scenario. The black vertical lines on the average success rate bar denote the standard error of the mean across different tasks for that scenario. Left: Performing joint training on bridge and target data leads to improved performance, here the task is included both in the bridge and target dataset. Middle: Using target domain data from other tasks helps transferring tasks from the bridge dataset to the target domain. Right: Joint training with the bridge data and a target task that is *not* contained in the bridge dataset enables significant generalization improvement compared to only training on the target task alone. Tasks with an asterisk (*) uses objects that are not part of the bridge dataset.

ments are included in the supplementary materials and on the project webpage, which we encourage the reader to view to get a clearer sense for the diversity of the tasks: `https://sites.google.com/view/bridgedata`

Quantitative metrics. All quantitative evaluations use 10 trials per task,varying object positions and distractors on every trial and varying the position of the robot relative to the environment every 5 trials. This ensures that all test configurations are unique and different from any condition seen in training, providing a measurement of generalization performance for the policy. When the experiments in toy kitchen 1-3 and toy sink 1-3 were conducted, the bridge dataset only comprised 4700 trajectories. Other experiments use the full dataset with 7200 trajectories total.

Scenario (1): transfer with matching behaviors. Figure 4.7 (left) shows results for the transfer learning with matching behaviors scenario, where the user provides some data for a set of tasks in the target domain (which are also present in the bridge data), and we evaluate whether including bridge data during training improves performance and generalization. For comparison, we include the performance of the policy when trained *only* on the target domain data, without bridge data (Target Domain Only), a baseline that uses only the bridge data without any target domain data (Direct Transfer), as well as baseline that trains a single-

Figure 4.8: Examples of successful trajectories performed by the policy jointly trained with prior data and target domain data. Left: put pot in sink (scenario 1); middle: put carrot on plate (scenario 2); Right: wipe plate with sponge (scenario 3).

task policy on data in the target domain only (Single Task). The Toy Kitchen 2 (tk2) target domain has 6 tasks, and Toy Sink 3 (ts3) has 10 tasks, each with 50 demonstrations.

As can be seen in the results, jointly training with the bridge data leads to significant gains in performance (66% success averaged over tasks) compared to the direct transfer (14% success), target domain only (28% success) and the single task (18% success) baseline. This is not surprising, since this scenario directly augments the training set with additional data of the *same* tasks, but it still provides a validation of the value of including bridge data in training (for a qualitative example see Figure 4.8, left).

Scenario (2): zero-shot transfer with target support. In the next experiment, we evaluate tasks in the target domain for which the user did *not* provide any data. Instead, the user only collected data for *other* tasks in the target domain. This experiment evaluates whether bridge data can be used to "import" tasks into the target domain. We provide a qualitative example for this scenario in Figure 4.8 middle, which shows an experiment where we transfer the "put carrot on plate" task into the Toy Sink 1 target domain using the bridge data and target domain data consisting of 10 *other* tasks. Due to space constraints, We provide a visualization of these other tasks on the project webpage.

Since there is no target domain data for these tasks, we cannot compare to a baseline that does not use bridge data at all, since such a baseline would have no data for these tasks. However, we do include the "direct transfer" baseline, which utilizes a policy trained only on the bridge data. Note that this comparison is non-trivial: it is not at all clear a priori that target domain data for *other* tasks should boost transfer performance of tasks that are only present in the bridge data. The results, shown in Figure 4.7 (middle), indicate that the jointly trained policy which obtains 44% success averaged over tasks indeed attains a very significant increase in performance over direct transfer (30% success), suggesting that the zero-shot transfer with target support scenario offers a viable way for users to "import" tasks from the bridge dataset into their domain.

Scenario (3): boosting generalization of new tasks. The last generalization scenario, which most directly evaluates the *bridge data hypothesis*, aims to study how well bridge data can boost the generalization of entirely new tasks in the target domain, which are not present in the bridge data. To study this question, we collected data for 10 different unique tasks in 4 different environments and excluded them from the bridge data to simulate a user collecting their own unique task in their new target environment. Figure 4.8 right illustrates one of these scenarios, where we collected 50 demonstrations for the "wipe place with sponge" task

in the the real kitchen 1 target domain. *Neither* data from the target domain nor this task or this object are present in the bridge data. After jointly training with both bridge and target data we obtain a significant generalization boost when running the policy in the target domain, compared to a policy trained on only the single-task target domain data. Direct transfer is impossible here, because the bridge data does not contain this task. The results are presented in Figure 4.7 (right), and show that training jointly with the bridge data leads to significant improvement on 6 out of 10 tasks across three evaluation environments, leading to 50% success averaged over tasks, whereas single task policies attain around 22% success – a 2× improvement in overall performance (the asterisks denote in which experiments the objects are *not* contained in the bridge data). The significant improvements obtained from including the bridge data suggest that bridge datasets can be a powerful vehicle for boosting generalization of new skills, and that a single shared bridge dataset can be utilized across a range of domains and applications. Of course, structural similarity between environments and tasks is important, and all of these evaluations use other toy kitchen or sink setups. We expect the applicability of a bridge dataset to increase as the breadth of domains and tasks in the dataset increases. **When does bridge data help?** In Figure C.1 we provide a list of example scenarios where the bridge data helps and where it does not (the first 7 rows). More qualitative results, including videos of these tasks and additional discussion, are provided on the project https://tinyurl.com/rt3uwwebsite due to space constraints. Qualitatively, we observed that the tasks that most consistently benefit from the inclusion of bridge data contain objects that visually resemble those seen in the bridge data (e.g., there are gains for 'put pear in bowl,' where the pear resembles the vegetables in the bridge data, but no gains in 'flip orange pot upright,' since the orange pot looks very different from any container in the bridge data), contain behavior that physically is related to behavior seen in the bridge data (e.g., in 'put detergent in dry rack,' the bridge data helps since the motion resembles the pick-and-place motions in the prior data, whereas in 'open box flaps' bridge data does not help since the type of pushing motions involved in this task are very rare in the bridge data), and take place in domains that are visually and structurally related to those in the bridge data (e.g., the bridge data helps with 'wipe plate with sponge' in Figure 4.8 in a real kitchen, but does not help with 'pick screwdriver from tool chest task,' since the scene does not resemble the toy kitchens in the bridge data). Unfortunately, it is difficult to provide a more precise and formal treatment of when transfer learning succeeds in general, though we expect this would be an exciting direction for future research.

4.6 Conclusion

We show how a large, diverse bridge dataset can be leveraged to improve generalization in robotic learning. Our experiments demonstrate that including bridge data when training skills in a new domain can improve performance across a range of scenarios, both for tasks that are present in the bridge data and, perhaps surprisingly, entirely new tasks. This means that bridge data may provide a generic tool to improve generalization in a user's

target domain. In addition, we showed that bridge data can also function as a tool to *import* tasks from the prior dataset to a target domain, thus increasing the repertoires of skills a user has at their disposal in a particular target domain. This suggests that a large, shared bridge dataset, like the one we have released, could be used by different robotics researchers to boost the generalization capabilities and the number of available skills of their imitation-trained policies.

Both our experimental evaluation and our technical approach do have a number of limitations. While we carefully set up our experiments to reflect a likely real-world usage scenario, where the target domain is distinct from the bridge data (i.e., to reflect what would happen if *someone else* were to use our bridge data for their robot in their lab), we still only evaluate in a few distinct settings, namely in 5 different environments at Institution 1.

However, our imitation learning results do illustrate the benefits of diverse bridge data, and we hope that by releasing our dataset to the community, we can take a step toward generalizing robotic learning and make it possible for anyone to train robotic policies that readily generalize to varied environments without repeatedly collecting large and exhaustive datasets.

Chapter 5

Pre-training For Robots: Offline RL on Diverse Data

5.1 Introduction

The previous chapter discussed how a simple behavioral cloning algorithm can be used to leverage diverse multi-domain multi-task data to learn new tasks in new domains more efficiently and to transfer tasks from one domain to the other. In this chapter we will focus on applying offline-reinforcement learning to the same problem and show that success rates are significantly improved. The rationale for applying offline RL to this problem is as follows:

Robotic reinforcement learning (RL) algorithms aim to learn skills via trial-and-error interaction with the real world [91, 140]. While reinforcement learning algorithms have led to impressive robotic demonstrations [108, 86, 85, 3, 195], running trial-and-error learning from scratch often limits the capabilities of these algorithms. This is because training on large diverse datasets is crucial for obtaining neural network policies that can generalize well, but the human effort, wall-clock time, and safety challenges associated with trial-and-error reinforcement learning make it difficult for a robot to autonomously collect diverse experience in a scalable manner. This in turn prevents a trial-and-error style RL algorithm from learning generalizable policies. A natural way to circumvent this limitation is to incorporate existing diverse robotic datasets into the training pipeline of a robotic RL algorithm.

An appealing paradigm for incorporating large and diverse datasets into robotic RL without the need to constantly recollect large datasets in the real world is to employ offline RL, where a large previously collected dataset can be used to bootstrap a policy for a particular task. This paradigm bears the promise of inducing effective generalization capabilities by training highly expressive neural networks on large amounts of diverse, robotic experience, aiming to mimic the success of deep learning in computer vision [43] and NLP [18, 39]. However, in practical robotic learning settings, we typically want the robot to perform a task with *new* objects that are not illustrated in the prior data. Can we develop a framework for robotic reinforcement learning where large amounts of prior data for *other* objects and tasks

are used in conjunction with much more limited amounts of task-specific data to enable the robot to learn to manipulate new objects efficiently?

In this paper, we develop a robotic system that can utilize general-purpose robotic datasets along with small amounts of task-specific data to learn policies that can solve a downstream task with novel objects. Our robotic system relies on offline pre-training via offline RL methods, followed by finetuning. In the first phase, our approach extracts a generalist policy by running offline multi-task RL on large robotic datasets, labeling the offline dataset with sparse binary rewards. This kind of large-scale multi-task training requires being able to utilize large, highly expressive deep neural networks, which presents a challenge for offline RL methods [14], and we find some practical architectural design choices that allow us to train highly-expressive ResNets successfully with offline RL. In the second phase, we convert this generalist policy into a specialist policy for a given target task by finetuning on limited amounts of data for the downstream task. This finetuning can be performed entirely offline, only using between 10 to 50 demonstrations from the target task, without needing to repeat the entire offline RL process or utilizing any sophisticated tools from prior work [104].

Our main contribution is pre-training for robots, or PTRin short, a robotic learning system that can utilize diverse robotic datasets to pre-train a general-purpose initialization for downstream task learning. PTR combines several previously proposed ideas into a complete robotic learning system, using the CQL algorithm [97] in combination with multi-task pre-training to provide a general offline RL-based initialization. We use large datasets, such as the Bridge dataset [45] and extended versions thereof. To train on such diverse datasets with offline RL, we discuss a number of important design decisions, including neural net architectures that we find particularly effective in this setting. Empirically, our results demonstrate that offline RL pre-training followed by offline RL finetuning on individual tasks can significantly improve over the performance of imitation learning methods proposed in prior work [45] as well as running joint training on all tasks from scratch. This finding is significant because it indicates that, even though both the prior dataset and the new task-specified demonstrations consist of expert human demonstrations, PTRcan leverage such data more effectively than imitation learning and standard offline RL methods.

5.2 Related Work

A number of prior works have proposed algorithms for offline RL [63, 98, 97, 94, 93, 184, 81, 61, 158]. Our goal is not to develop new offline RL methods, but to devise a complete robotic learning system based on offline RL pre-training from multi-task datasets and finetuning to downstream tasks. Prior works that study finetuning have examined parameter sharing [181, 134, 163, 51, 75, 198, 191, 194, 162, 85, 197], applying some form of data sharing or relabeling strategy [197, 6, 199, 85, 186], and utilizing diverse multi-task data for representation learning [116, 193, 192, 129]. Our focus is on developing an offline RL system for real-world robotic learning, which can specifically be used to pre-train on a diverse multi-task dataset and then specialize the policy to one or multiple downstream tasks.

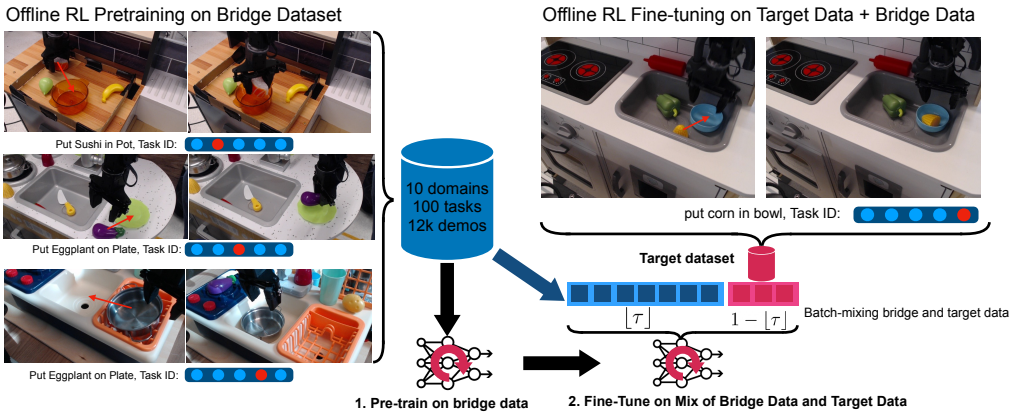


Figure 5.1: Overview of the proposed system, PTR: We first perform general offline pre-training on diverse multi-task robot data and subsequently finetune on one or several a target tasks while mixing batches between the prior data and the target dataset using a batch mixing ratio of τ.

Methods for online finetuning from offline RL initialization [126, 93, 104] focus on improving policies learned from offline datasets with minimal online interaction. Unlike these works, we consider the setting where the offline dataset does not need to include data nor rewards for the downstream task. This resembles the problem setting considered by offline meta-RL methods [111, 41, 122, 144, 112]. However, we consider notably more complex tasks (e.g., grasping objects from image observations on a real robot) than these prior works.

In the field of robotic learning, quite a few prior works have sought to utilize offline datasets [142, 50, 57, 68, 116, 30, 147, 96, 119, 160, 161, 183, 129, 150] to avoid trial-and-error learning, some building on offline RL methods [1, 176, 97, 137, 200]. Prior works have also sought to utilize offline datasets in conjunction with some task-specific online interaction, primarily focusing on the single-task setting [148, 138, 173, 92, 70, 128, 126, 93, 101, 104, 119]. Some prior works do consider the multi-task setting, but the offline dataset already contains skills needed to solve the target task [89, 125, 160, 3]. In our setting, the offline dataset does not necessarily have to contain the skill needed to solve the downstream task, and this skill can be acquired during the process of finetuning. A number of prior methods have sought to apply online RL to robotic control by utilizing simulated training, with various methods for transfer into the real world [76, 149]. This work is distinct and partly complementary to ours: our method could also use simulated prior data, but our focus is specifically on developing methods that don't require online RL, thus removing the need for hand-engineered simulators all together.

Most closely related to our work are prior methods that run model-free offline RL on diverse real data and then finetune to new tasks [160, 85, 82, 24, 101]. We propose a multi-task pre-training scheme followed by finetuning to one or multiple target tasks with a small amount of data. In contrast, these prior methods use *significantly* more data for the new

task than our system: PTR can learn a new task with as few as 10 demonstrations, while prior works use several thousand trials for the new task [24, 85] or multiple hours or even days of online collection [82, 101]. Our work also shows that it is possible to pre-train on a previously-collected multi-task and multi-domain dataset, in contrast to prior work that uses data from a single environment [160, 85, 24] or a single task [82]. We also show that pre-training and finetuning with offline RL exceeds the performance of imitation learning, even when all data consists of demonstrations, unlike prior work that showed imitation to perform better in such settings [119] and unlike prior RL work that uses high-coverage scripted data [160, 85, 82, 24].

5.3 Preliminaries and Problem Statement

Robotic RL methods are derived under the formal model of a Markov decision process (MDP), which is defined as 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. $\mu_0(s)$ 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 finetune 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 5.1 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. Our goal is to utilize this multi-task dataset $\mathcal{D}$, to find the best possible policy for one or multiple target tasks (denoted without loss of generality as task $\mathcal{T}_{\text{target}} = \{k+1, \cdots, n\}$), for which no experience is observed in $\mathcal{D}$. While the diverse dataset $\mathcal{D}$ does not contain any experience for the target tasks, 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. Note that the size of $\mathcal{D}^*$ is extremely small: in our experiments we consider between 10 to 50 demonstrations for a given target task, such that a policy that simply ignores the diverse offline dataset is unlikely to succeed. Our goal is to attain the best possible policy for tasks $\mathcal{T}_{\text{target}}$ at the end.

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 of any policy in an MDP 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 [98, 63, 81, 97, 61] 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) [97]. We will discuss how we adapt CQL for general-purpose pre-training on diverse data followed by single-task finetuning in Section 5.4.

Tasks and domains. Our problem statement involves pre-training on data from many tasks and domains, which we source from the bridge dataset [45], and finetuning to a new task in a new domain. This is only viable when the domains and tasks have something in common. Our terminology for "task" and "domain" follows ebert2021bridge: a task corresponds to a skill-object pair, such as "put potato in pot" and a domain corresponds to a particular 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 resemble the ones seen in training (i.e., come from the same distribution), but are not seen in the prior data. We describe the specific tasks in Section 5.5.

5.4 Learning Specialist Policies for New Tasks from Diverse Data Pre-training

To effectively solve new tasks from diverse offline datasets, a robotic learning system 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 this section, we present our system, PTR, that provides these benefits by training a single, highly-expressive deep neural network policy on the diverse robotic dataset via multi-task 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 system in Section 5.4 and then discuss some practical design choices that are crucial for attaining good performance in Section 5.4.

The Components of PTR

To satisfy both requirements **(1)** and **(2)** from above, our robotic system uses a multi-task offline RL approach, where the policy and value 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. We would expect this multi-task training to replicate the generalization benefits of training highly-expressive neural network models via supervised learning, but in the context of robotic RL. Once a policy is obtained via this generalist pre-training process, we adapt this policy for solving a new target task by utilizing a very small amount of target task data. We describe the two phases (pre-training and fine-tuning) below:

Phase 1: Generalist multi-task offline RL pre-training. In the first phase, our system 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) [97] 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 out-of-distribution actions, which can lead to poor offline RL performance [98]. Formally, the training objective for our multi-task Q-function, as prescribed by CQL, is given by:

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

$\bar{Q}$ denotes a target Q-network, which 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 finetuned to a new downstream task.

Phase 2: Fine-tuning π_θ^{off} and Q_ϕ^{off} to target tasks $\mathcal{T}_{\text{target}}$. In the second phase, our system 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 $\mathcal{D}^*$. Our method for adaptation 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, we propose to create minibatches that contain a higher proportion of data from the target task compared to the pre-training dataset during fine-tuning. Specifically, 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 sufficiently 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(s) without overfitting.

Handling task identifiers. 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 never train the network corresponding to $\mathcal{T}_{\text{target}}$. 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. PTRprovides the 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 5.5 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 system is shown in Figure 5.1. We use a value of $\alpha = 10.0$ in multi-task CQL and $\tau = 0.7$ 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. Additional details of our system can be found in Appendix D.4.

Crucial Design Choices and Practical Considerations

Beyond the components in Section 5.4, we need to make some important design decisions, including the reward functions for offline pre-training, neural network architectures to be able to learn from diverse data, and cross-validation metrics to identify policies we expect to be effective after finetuning. We present our design choices below, and present supporting analysis in Appendix D.5.

Reward specification. In this work, we pre-train on the bridge dataset [45], which consists of human-teleoperated demonstration data. The demonstration data, however, does not come annotated with rewards. Perhaps an obvious choice is to label the last transition of a trajectory as a success, and give it +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.3, this provided the best results.

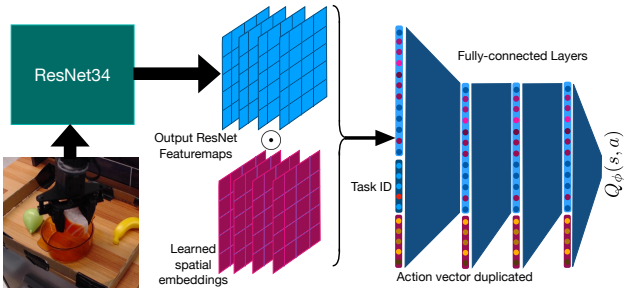


Figure 5.2: The Q-function architecture for PTR. The encoder is a ResNet34 with group normalization along with learned spatial embeddings (left). The decoder (right) is a multi-layer perceptron with the action vector duplicated and passed in at each layer. A one-hot task identifier is also passed into the input of the decoder.

Policy and Q-function architectures. To succeed at rapid finetuning, π^{off} and Q^{off} need to extract features useful for learning other tasks. This necessitates the use of high-capacity neural network models for representing the policy and the Q-function and requires us to stably train them. We experimented with a variety of standard (high-capacity) architectures for vision-based robotic RL.

This includes standard convolutional architectures [160] and IMPALA architectures [51]. However, we observed that these models were unable to effectively handle the diversity of the pre-training data. Then, we attempted to utilize standard ResNets [74] (ResNet-18, Resnet-34, and their adaptations to imitation problems from ebert2021bridge) to represent Q_ϕ, but faced divergence challenges similar to prior efforts [14, 13]. We found that by simply replacing batch normalization layers — known to be notoriously hard to train with TD-learning [13] — with group normalization layers [185], we were able to stably train ResNet Q-functions. We also observed that choosing an appropriate method for converting the 3-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. Simply computing global average pooling (as used in many classification architectures) performs similarly poorly as using a spatial softmax. 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. An illustration of this architecture is provided in Figure 5.2.

Next, we found that a Q-function $Q_\phi(\mathbf{s}, \mathbf{a})$ obtained by running naïve multi-task CQL tends to not use the action input $\mathbf{a}$ effectively, due to strong correlations between $\mathbf{s}$ and $\mathbf{a}$ in the offline data. 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 $\mathbf{a}$ as input at each fully connected layer, which (as shown in Figure 5.2 and Appendix D.5), greatly alleviates the issue.

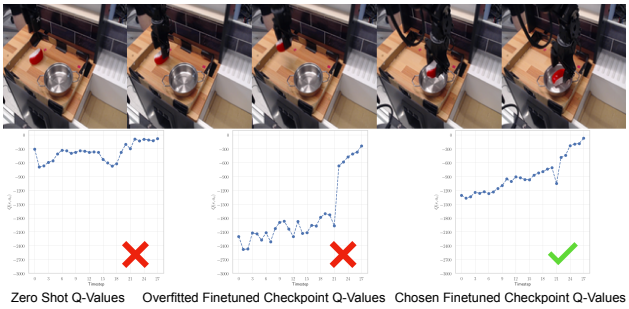


Figure 5.3: Top: PTR policy rollout of task "put sushi in pot" re-targeted to metal-pot. Bottom: left: Q-value over time for a target task trajectory before fine-tuning begins (zero-shot), middle: Q-values for a checkpoint that has started to overfit after being trained for long and exhibits drastic changes in Q-values over the course of a trajectory, and right: Q-values for a checkpoint that attains high performance.

Our policy is also represented via an identical architecture as the Q-function, except that it predicts $|\mathcal{A}|$-dimensional actions as outputs and does not have action inputs. Similar to prior works [72, 62], we rescale the action between $[-1, 1]^{|\mathcal{A}|}$, and apply a $\tanh(\cdot)$ squashing function at the output.

Cross-validation after finetuning. Since we wish to learn task-specific policies that do not overfit to small amounts of data, thereby losing their generalization ability, we must apply the right number of gradient steps during finetuning: 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 tradeoff, we use a simple heuristic: we

run finetuning for many iterations while also plotting the learned Q-values over a held-out dataset of trajectories from the target task. Then, we pick the checkpoint for which the learned Q-values are (roughly) monotonically increasing over the course of an held-out trajectory (see Figure 5.3 for an example). Empirically we find that this heuristic guides us to identify a good checkpoints (more examples in Appendix D.5).

5.5 Experimental Evaluation of PTR and Takeaways for Robotic RL

The goal of our experiments is to show that PTR can learn effective policies for new tasks by leveraging diverse robotic datasets alongside only a handful of user-provided demonstrations for a given target task. To this end, we evaluate PTR in a variety of robotic manipulation settings including scenarios where **(a)** the target task uses previously unseen objects that the robot must manipulate, **(b)** the target task requires performing a previously observed task but this time in a previously unseen domain, and **(c)** the target task requires retargeting the behavior of an existing skill, in this case changing the type of object types it interacts with, by using the target demonstrations. We will first describe our real-world setup, then discuss the methods we shall compare to, and finally present our empirical results. At the end, we will also present some diagnostic analysis in simulation that further corroborates takeaways from our real-world results. For more details, visuals and our video, please visit our anonymous website at `https://sites.google.com/view/ptrcorl`.

Real-world experimental setup. We directly utilize the publicly available *bridge dataset* [45] for pre-training, as it provides a large number of robot demonstrations for a diverse set of tasks in multiple domains. 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 of our setup can be found in Appendix D.1.

Comparisons. Since the datasets we use (both the bridge dataset and the newly collected door opening data) consist of human demonstrations, as indicated by prior work mandlekar2021what, 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. Next, to assess the importance of performing pre-training *followed* by fine-tuning, we compare PTRto **(i)** standard multi-task offline CQL (**CQL (joint)**) that jointly trains on the pre-training data and the target task data from scratch, and therefore skips Phase 1, and directly applies Phase 2 of PTR, and **(ii)** multi-task offline CQL (**CQL (0-shot)**) that does

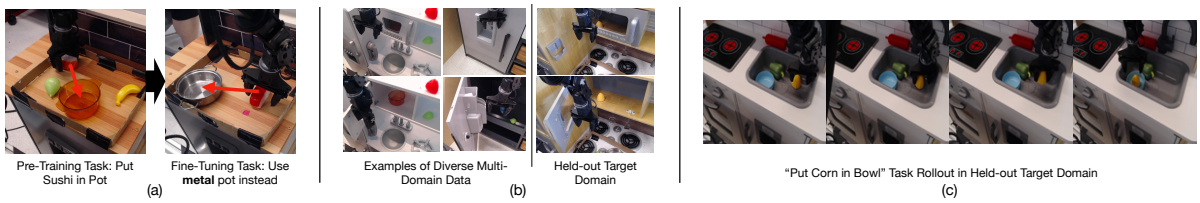


Figure 5.4: Illustrations of the three real-world experimental setups we evaluate PTRon: **(a)** the "put sushi in a metallic pot" task which requires retargeting, **(b)** the task of opening an unseen door, and **(c)** the "put corn in bowl in a sink" task in a new kitchen domain. Figure (c) shows a successful rollout from PTR.

not use the target demonstrations at all. We also make the analogous comparison for BC, jointly training BC on the pre-training and target task data from scratch (**BC (joint)**).

Scenario 1: Re-targeting skills for existing tasks to act on new objects during finetuning. We utilized the subset of the bridge with all pick-and-place tasks in one toykitchen for pre-training, and selected the "put sushi in pot" task as our target task. This task is demonstrated in the bridge dataset, but only using an orange transparent pot (see Figure 5.4 (a)). In order to pose 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.

This scenario is challenging since the metallic pot drastically differs from the orange transparent pot visually. Moreover, since the color of the tabletop is similar to that of the transparent pot, any learned model that does not fine-tune effective would be vulnerable to spuriously identifying a patch on the tabletop as the pot, when the transparent pot is absent. By pre-training on all pick-and-place tasks in this domain (32 tasks) and jointly 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, BC appears to be mistaking arbitrary patches on the tabletop with the pot. Quantitatively, we observe, as shwon in Table 5.1, that PTRis able to complete the task with reasonable accuracy, whereas 0-shot and fine-tuned BC are completely unable to solve the task. The fact that 0-shot CQL has great difficulty solving the task indicates that target demonstrations are necessary for solving this task, and PTRis able to make efficient use of these demonstrations (prior work [45] also found BC performs poorly with even 50 demonstrations).

Method	**Success rate**
BC (0-shot)	0%
BC (finetune)	0%
CQL (0-shot)	7%
PTR(Ours)	**47.0%**

Table 5.1: Performance of PTRfor "put sushi in metallic pot" in Scenario 1. PTRsubstantially outperforms BC (finetune), even though it is provided access to only demonstration data. We also show some examples comparing some trajectories of BC and PTRin Appendix D.3.

Scenario 2: Generalizing to previously unseen domains. Next, we study whether PTRcan 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 (video on the website: `https://sites.google.com/view/ptrcorl`). The target door (shown in Figure 5.4(b)) we wish to open and corresponding toykitchen domain is never seen previously in the pre-training data, and doors in the pre-training data exhibit different sizes, shapes, handle types and visual appearances. Due to the limited number of demonstrations and the associated task complexity, in order to succeed, an algorithm must effectively leverage the pre-training data. Concretely, for pre-training, we used a dataset of 800 door-opening demonstrations on 12 different doors in 4 different toykitchen domains, and we utilize 10 demonstrations on a held-out door for finetuning. Table 5.2 shows that PTRattains higher performance than both BC baselines and CQL (joint).

Method	**Success rate**
BC (0-shot)	0%
CQL (0-shot)	0%
BC (joint)	35%
CQL (joint)	25%
BC (finetune)	50%
PTR(Ours)	**60%**

Table 5.2: Performance of PTRfor opening a new target door in Scenario 2 averaged over 20 independently chosen evaluation trials. PTRoutperforms both BC (finetune) and BC (joint) given access to the same data. Note that jointly training on pre-training data and the target data is worse than finetuning from the pre-trained initialization.

Interestingly, Table 5.2 shows that while CQL (joint) 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 baseline despite using the very same CQL algorithm. 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 to first 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 PTRin learning to solve a new task in a new domain. Unlike the two scenarios studied above, where we attempted to solve the same task in a different domain ("open a different door"), or re-target task in the same domain ("put sushi in a metallic pot"), in this scenario, we attempt to solve a new task ("put corn in a bowl in sink") in a new kitchen scene. This task is represented via a unique task identifier, and we are not provided with any data for this task identifier during pre-training. 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 scene, and then fine-tune on demonstration data from this new kitchen. More details are in Appendix D.2.

Figure 5.4 (c) shows a rollout of the final policy found by PTRsuccessfully performing the "put corn in pot task," which is not included in the bridge data.

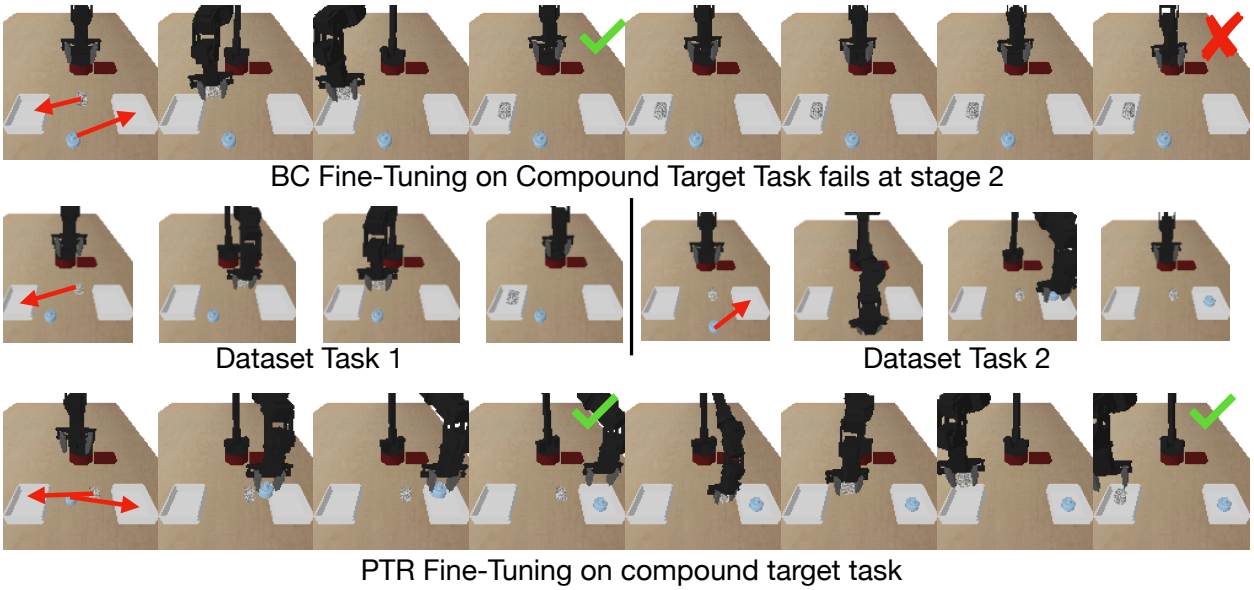


Figure 5.5: PTRand BC (finetune) on the simulated bin sorting task used for our diagnostic study. Note that while PTR is able to apply the skill of sort an object multiple times in a single trajectory, BC usually fails to learn this behavior.

A quantitative comparison between PTRand BC (finetune) (we only evaluated BC (finetune) in this scenario since this was the strongest BC baseline in other scenarios) shown in Table 5.2 indicates that PTRclearly outperforms fine-tuned BC on this task, demonstrating the capability of PTR for acquiring a new skill in new domains.

Method	**Success rate**
CQL (joint)	0%
BC (finetune)	40%
PTR(Ours)	**50%**

Table 5.3: Performance of PTRon the "put corn in bowl in sink" task in Scenario 3, averaged over 20 evaluation trials. PTRoutperforms BC (finetune) and CQL (joint), indicating that PTRcan generalize to new tasks in new domains better than BC.

Diagnostic study in simulation. Finally, 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 D.1). 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, that 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.

We also note that while BC (finetune) is able to only successfully sort one of the objects, PTR is able to successfully apply the skill of sorting an object multiple times. The absolute success rates attained by PTR and BC (finetune) are **8%**, and 2% respectively, indicating that PTR is better. Second, we observed in Scenario 2 that joint training with offline RL led to worse performance than PTR. This trend also holds in simulation (CQL (joint) attains only a 3% success rate, which is better than BC (finetune), but still worse than 8% for PTR), and this finding further supports the intuition that pre-training via offline RL on diverse data can give rise to a great initialization that can be fine-tuned.

5.6 Conclusion, Discussion, and Limitations

We presented a learning system that uses diverse prior data for general-purpose offline RL pretraining, 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 enables PTR to learn a generalist policy that 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. More broadly, our aim is to take a step toward large-scale general-purpose pre-training on diverse data in robotics, analogously to the kind of pre-training methods that have transformed vision/NLP [40, 146]. 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 by robotic practitioners as a starting point, similar to GPT3 [18].

Limitations. Our method requires the prior data and new tasks to be structurally similar. We expect that as the prior data becomes broader, the range of new tasks that could be tackled will increase. Additionally, our system must fine-tune on the same type of robot. Multi-robot pre-training could address this limitation, and studying such multi-robot training is an exciting direction.

Chapter 6

Conclusion

In the first part, we showed how model-based reinforcement learning that uses pixels as the state representation and learns a dynamics model over images can effectively leverage large amounts of data obtained from random scripted policies to perform complex object rearrangement tasks including pushing and grasping. We showed that pre-training on multi-robot data and fine-tuning on a small amount of data from a different, held-out target robot can greatly boost performance over training on the target robot alone.

In the second part, we showed how multi-domain, multi-task human teleoperated demonstration data can be used for joint training and fine-tuning with a target task, boosting generalization even if the target task uses new objects or is situated in new scene or domain. We also showed that fine-tuning performance can be further increased by using offline-reinforcement learning instead of imitation learning, and empirically confirmed that increasing model-capacity leads to better policy performance.

Both parts demonstrate the benefit and potential of accumulating and leveraging large and diverse robotic datasets for boosting generalization and adapatability of robotic agents, indicating that the lessons learned in large-scale deep learning indeed apply to robotics as well.

A promising direction that we have not been able to work on is whether video data from humans on the internet can be filtered and incorporated into the training process increasing the dataset size by orders of magnitudes. This could lead to even better generalization. For example since the diversity of objects in internet videos is far greater than our datasets, it is conceivable that better objects representations and affordances could be learned by incorporating such data. Furthermore, language annotations which often exist for in-the-wild videos from the internet could be added into the training process which could natural language grounding, and a way of specifying goals via natural language.

I hope that this work can serve as a motivation for further scaling up robot learning datasets and finding yet more general methods that allow leveraging a broader variety of robotic data, leading to ever greater generalization capabilities.

Bibliography

[1] A. Abdolmaleki et al. "Maximum a Posteriori Policy Optimisation". In: *International Conference on Learning Representations (ICLR)*. 2018.

[2] Pulkit Agrawal et al. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Neural Information Processing Systems*. 2016.

[3] Michael Ahn et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *arXiv preprint arXiv:2204.01691* (2022).

[4] Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. "Modular meta-learning". In: *arXiv:1806.10166* (2018).

[5] Haitham Bou Ammar et al. "Online multi-task learning for policy gradient methods". In: *International Conference on Machine Learning*. 2014, pp. 1206–1214.

[6] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Neural Information Processing Systems*. 2017, pp. 5048–5058.

[7] Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *arXiv:1808.00177* (2018).

[8] Mohammad Babaeizadeh et al. "Stochastic variational video prediction". In: *International Conference on Learning Representations (ICLR)* (2018).

[9] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. "Visual tracking with online multiple instance learning". In: *Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2009.

[10] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: *CoRR* abs/1812.03079 (2018). arXiv: 1812.03079.

[11] Marc Bellemare et al. "Unifying count-based exploration and intrinsic motivation". In: *Advances in Neural Information Processing Systems*. 2016.

[12] John T Betts. "Survey of numerical methods for trajectory optimization". In: *Journal of guidance, control, and dynamics* (1998).

[13] Aditya Bhatt et al. "CrossNorm: Normalization for Off-Policy TD Reinforcement Learning". In: *arXiv e-prints*, arXiv:1902.05605 (Feb. 2019), arXiv:1902.05605. arXiv: 1902.05605 [cs.LG].

[14] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. "Towards Deeper Deep Reinforcement Learning". In: *arXiv preprint arXiv:2106.01151* (2021).

[15] Byron Boots, Arunkumar Byravan, and Dieter Fox. "Learning predictive models of a depth camera & manipulator from raw execution traces". In: *International Conference on Robotics and Automation (ICRA)*. 2014.

[16] Konstantinos Bousmalis et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[17] Rodney Brooks. "A robust layered control system for a mobile robot". In: *IEEE journal on robotics and automation* 2.1 (1986), pp. 14–23.

[18] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[19] Andreja Bubic, D. Yves Von Cramon, and Ricarda Schubotz. "Prediction, cognition and the brain". In: *Frontiers in Human Neuroscience* (2010).

[20] Yuri Burda et al. "Exploration by random network distillation". In: *arXiv preprint arXiv:1810.12894* (2018).

[21] Arunkumar Byravan and Dieter Fox. "Se3-nets: Learning rigid body motion using deep neural networks". In: *arXiv:1606.02378* (2016).

[22] Arunkumar Byravan and Dieter Fox. "Se3-nets: Learning rigid body motion using deep neural networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 173–180.

[23] Roberto Calandra et al. "The feeling of success: Does touch sensing help predict grasp outcomes?" In: *arXiv:1710.05512* (2017).

[24] Yevgen Chebotar et al. "Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills". In: *arXiv preprint arXiv:2104.07749* (2021).

[25] Yevgen Chebotar et al. "Bigs: Biotac grasp stability dataset". In: *ICRA 2016 Workshop on Grasping and Manipulation Datasets*. 2016.

[26] Yevgen Chebotar et al. "Combining model-based and model-free updates for trajectory-centric reinforcement learning". In: *International Conference on Machine Learning*. 2017.

[27] Silvia Chiappa et al. "Recurrent Environment Simulators". In: *arXiv:1704.02254* (2017).

[28] Paul F. Christiano et al. "Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model". In: *CoRR* abs/1610.03518 (2016). arXiv: 1610.03518.

[29] Ignasi Clavera et al. "Learning to Adapt: Meta-Learning for Model-Based Control". In: *CoRR* abs/1803.11347 (2018). arXiv: 1803.11347.

[30] Sudeep Dasari et al. "RoboNet: Large-Scale Multi-Robot Learning". In: *Conference on Robot Learning*. PMLR. 2020, pp. 885–897.

[31] Sudeep Dasari et al. "Robonet: Large-scale multi-robot learning". In: *arXiv preprint arXiv:1910.11215* (2019).

[32] Bert De Brabandere et al. "Dynamic filter networks". In: *Neural Information Processing Systems (NIPS)*. 2016.

[33] Marc Deisenroth and Carl E Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *International Conference on Machine Learning (ICML)*. 2011.

[34] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian processes for data-efficient learning in robotics and control". In: *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2013), pp. 408–423.

[35] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. "A survey on policy search for robotics". In: *Foundations and Trends® in Robotics* (2013).

[36] Marc Peter Deisenroth et al. "Multi-task policy search for robotics". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3876–3881.

[37] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *computer vision and pattern recognition*. Ieee. 2009.

[38] Coline Devin et al. "Learning modular neural network policies for multi-task and multi-robot transfer". In: *International Conference on Robotics and Automation (ICRA)*. 2017.

[39] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[40] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[41] Ron Dorfman and Aviv Tamar. "Offline meta reinforcement learning". In: *arXiv e-prints* (2020), arXiv–2008.

[42] Alexey Dosovitskiy and Vladlen Koltun. "Learning to act by predicting the future". In: *International Conference on Learning Representations (ICLR)* (2017).

[43] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[44] Yan Duan et al. "One-shot imitation learning". In: *Advances in neural information processing systems*. 2017.

[45] Frederik Ebert et al. "Bridge Data: Boosting Generalization of Robotic Skills with Cross-Domain Datasets". In: *arXiv preprint arXiv:2109.13396* (2021).

[46] Frederik Ebert et al. "Robustness via Retrying: Closed-Loop Robotic Manipulation with Self-Supervised Learning". In: *Conference on Robot Learning (CoRL)* (2018).

[47] Frederik Ebert et al. "Self-Supervised Visual Planning with Temporal Skip Connections". In: *Conference on Robot Learning (CoRL)* (2017).

[48] Frederik Ebert et al. "Self-supervised visual planning with temporal skip connections". In: *arXiv:1710.05268* (2017).

[49] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv:1812.00568* (2018).

[50] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv preprint arXiv:1812.00568* (2018).

[51] Lasse Espeholt et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1407–1416.

[52] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML)* (2017).

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

[54] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Neural Information Processing Systems (NIPS)*. 2016.

[55] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in neural information processing systems*. 2016, pp. 64–72.

[56] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *International Conference on Robotics and Automation (ICRA)*. 2017.

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

[58] Chelsea Finn et al. "Deep spatial autoencoders for visuomotor learning". In: *International Conference on Robotics and Automation (ICRA)*. 2016.

[59] Chelsea Finn et al. "Deep spatial autoencoders for visuomotor learning". In: *International Conference on Robotics and Automation (ICRA)*. 2016.

[60] Chelsea Finn et al. "One-shot visual imitation learning via meta-learning". In: *arXiv:1709.04905* (2017).

[61] Scott Fujimoto and Shixiang Shane Gu. "A Minimalist Approach to Offline Reinforcement Learning". In: *arXiv preprint arXiv:2106.06860* (2021).

[62] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1587–1596.

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

[64] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to Fly by Crashing". In: *arXiv:1704.05588* (2017).

[65] Ali Ghadirzadeh et al. "Deep predictive policy training using reinforcement learning". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[66] Ali Ghadirzadeh et al. "Human-centered collaborative robots with deep reinforcement learning". In: *IEEE Robotics and Automation Letters* (2020).

[67] Erin Grant et al. "Concept acquisition via meta-learning: Few-shot learning from positive examples". In: *NIPS Workshop on Cognitively-Informed Artificial Intelligence*. 2017.

[68] Abhinav Gupta et al. "Robot learning in homes: Improving generalization and reducing dataset bias". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9112–9122.

[69] Abhishek Gupta et al. "Learning invariant feature spaces to transfer skills with reinforcement learning". In: *arXiv:1703.02949* (2017).

[70] Abhishek Gupta et al. "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning". In: *arXiv preprint arXiv:1910.11956* (2019).

[71] David Ha and Jürgen Schmidhuber. "World Models". In: *arXiv:1803.10122* (2018).

[72] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. Tech. rep. 2018.

[73] Kaiming He et al. "Deep residual learning for image recognition". In: *Conference on Computer Vision and Pattern Recognition*. 2016.

[74] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[75] Matteo Hessel et al. "Multi-task deep reinforcement learning with popart". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3796–3803.

[76] Daniel Ho et al. "Retinagan: An object-aware approach to sim-to-real transfer". In: (2021), pp. 10920–10926.

[77] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *arXiv preprint arXiv:1606.03476* (2016).

[78] Brody Huval et al. "An empirical evaluation of deep learning on highway driving". In: *arXiv:1504.01716* (2015).

[79] Stephen James, Michael Bloesch, and Andrew J Davison. "Task-embedded control networks for few-shot imitation learning". In: *arXiv:1810.03237* (2018).

[80] Stephen James et al. "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks". In: *Computer Vision and Pattern Recognition*. 2019.

[81] Natasha Jaques et al. "Way off-policy batch deep reinforcement learning of implicit human preferences in dialog". In: *arXiv preprint arXiv:1907.00456* (2019).

[82] Ryan Julian et al. "Never Stop Learning: The Effectiveness of Fine-Tuning in Robotic Reinforcement Learning". In: *arXiv e-prints* (2020), arXiv–2004.

[83] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[84] Gregory Kahn et al. "Uncertainty-Aware Reinforcement Learning for Collision Avoidance". In: *arXiv:1702.01182* (2017).

[85] Dmitry Kalashnikov et al. "MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale". In: *arXiv preprint arXiv:2104.08212* (2021).

[86] Dmitry Kalashnikov et al. "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Conference on Robot Learning*. 2018, pp. 651–673.

[87] Dmitry Kalashnikov et al. "Scalable deep reinforcement learning for vision-based robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2018, pp. 651–673.

[88] Nal Kalchbrenner et al. "Video pixel networks". In: *arXiv:1610.00527* (2016).

[89] Alexander Khazatsky et al. "What Can I Do Here? Learning New Skills by Imagining Visual Affordances". In: *arXiv preprint arXiv:2106.00671* (2021).

[90] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014).

[91] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[92] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. "Robot motor skill coordination with EM-based reinforcement learning". In: *2010 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2010, pp. 3232–3237.

[93] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. "Offline reinforcement learning with implicit q-learning". In: 2021.

[94] Ilya Kostrikov et al. "Offline reinforcement learning with fisher divergence critic regularization". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5774–5783.

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[96] Aviral Kumar et al. "A Workflow for Offline Model-Free Robotic Reinforcement Learning". In: *5th Annual Conference on Robot Learning*. 2021. URL: `https://openreview.net/forum?id=fy4ZBWxYbIo`.

[97] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *arXiv preprint arXiv:2006.04779* (2020).

[98] Aviral Kumar et al. "Stabilizing off-policy q-learning via bootstrapping error reduction". In: *Advances in Neural Information Processing Systems*. 2019, pp. 11761–11771.

[99] Thanard Kurutach et al. "Learning Plannable Representations with Causal InfoGAN". In: *arXiv:1807.09341* (2018).

[100] Thanard Kurutach et al. "Learning Plannable Representations with Causal InfoGAN". In: *CoRR* abs/1807.09341 (2018). arXiv: `1807.09341`.

[101] Alex X Lee et al. "How to Spend Your Robot Time: Bridging Kickstarting and Offline Reinforcement Learning for Vision-based Robotic Manipulation". In: *arXiv preprint arXiv:2205.03353* (2022).

[102] Alex X Lee et al. "Stochastic Adversarial Video Prediction". In: *arXiv:1804.01523* (2018).

[103] Alex X Lee et al. "Stochastic adversarial video prediction". In: *arXiv:1804.01523* (2018).

[104] Seunghyun Lee et al. "Offline-to-Online Reinforcement Learning via Balanced Replay and Pessimistic Q-Ensemble". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1702–1712.

[105] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.

[106] Ian Lenz and Ashutosh Saxena. "Deepmpc: Learning deep latent features for model predictive control". In: *In RSS*. Citeseer. 2015.

[107] Sergey Levine et al. "End-to-End Learning of Deep Visuomotor Policies". In: *Journal of Machine Learning Research (JMLR)* (2016).

[108] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[109] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *International Journal of Robotics Research (IJRR)* (2016).

[110] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* (2018).

[111] Jiachen Li et al. "Multi-task batch reinforcement learning with metric learning". In: *arXiv preprint arXiv:1909.11373* (2019).

[112] Sen Lin et al. "Model-Based Offline Meta-Reinforcement Learning with Regularization". In: *arXiv preprint arXiv:2202.02929* (2022).

[113] YuXuan Liu et al. "Imitation from observation: Learning to imitate behaviors from raw video via context translation". In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[114] William Lotter, Gabriel Kreiman, and David Cox. "Deep predictive coding networks for video prediction and unsupervised learning". In: *International Conference on Learning Representations (ICLR)* (2017).

[115] Corey Lynch et al. "Learning Latent Plans from Play". In: *arXiv preprint arXiv:1903.01973* (2019).

[116] Ajay Mandlekar et al. "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)*. IEEE. 2020, pp. 4414–4420.

[117] Ajay Mandlekar et al. "ROBOTURK: A Crowdsourcing Platform for Robotic Skill Learning through Imitation". In: *arXiv:1811.02790* (2018).

[118] Ajay Mandlekar et al. "Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity". In: *arXiv:1911.04052* (2019).

[119] Ajay Mandlekar et al. "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.

[120] Michael Mathieu, Camille Couprie, and Yann LeCun. "Deep multi-scale video prediction beyond mean square error". In: *International Conference on Learning Representations (ICLR)* (2016).

[121] Simon Meister, Junhwa Hur, and Stefan Roth. "UnFlow: Unsupervised Learning of Optical Flow with a Bidirectional Census Loss". In: *arXiv:1711.07837* (2017).

[122] Eric Mitchell et al. "Offline meta-reinforcement learning with advantage weighting". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7780–7791.

[123] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv:1312.5602* (2013).

[124] Michael E Moran. "The da Vinci robot". In: *Journal of endourology* 20.12 (2006), pp. 986–990.

[125] A. Nair et al. "Contextual Imagined Goals for Self-Supervised Robotic Learning". In: *Conference on Robot Learning (CoRL)*. 2019.

[126] Ashvin Nair et al. "Accelerating Online Reinforcement Learning with Offline Datasets". In: *arXiv preprint arXiv:2006.09359* (2020).

[127] Ashvin Nair et al. "Combining self-supervised learning and imitation for vision-based rope manipulation". In: *International Conference on Robotics and Automation (ICRA)*. 2017.

[128] Ashvin Nair et al. "Overcoming Exploration in Reinforcement Learning with Demonstrations. CoRR abs/1709.10089 (2017)". In: *arXiv preprint arXiv:1709.10089* (2017).

[129] Suraj Nair et al. "R3M: A Universal Visual Representation for Robot Manipulation". In: *arXiv preprint arXiv:2203.12601* (2022).

[130] Suraj Nair et al. "Time Reversal as Self-Supervision". In: *arXiv:1810.01128* (2018).

[131] Simon Niklaus, Long Mai, and Feng Liu. "Video Frame Interpolation via Adaptive Separable Convolution". In: *International Conference on Computer Vision (ICCV)*. 2017.

[132] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016).

[133] Junhyuk Oh et al. "Action-conditional video prediction using deep networks in atari games". In: *Neural Information Processing Systems*. 2015.

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

[135] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. "Self-Supervised Exploration via Disagreement". In: *arXiv preprint arXiv:1906.04161* (2019).

[136] Deepak Pathak et al. "Zero-shot visual imitation". In: *Conference on Computer Vision and Pattern Recognition Workshops*. 2018.

[137] Xue Bin Peng et al. "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning". In: *arXiv preprint arXiv:1910.00177* (2019).

[138] Xue Bin Peng et al. "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14.

[139] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[140] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697.

[141] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *International Conference on Robotics and Automation (ICRA)*. 2016.

[142] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *international conference on robotics and automation (ICRA)*. 2016.

[143] Lerrel Pinto et al. "The curious robot: Learning visual representations via physical interactions". In: *European Conference on Computer Vision*. 2016.

[144] Vitchyr H Pong et al. "Offline Meta-Reinforcement Learning with Online Self-Supervision". In: *arXiv preprint arXiv:2107.03974* (2021).

[145] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8748–8763.

[146] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *arXiv preprint arXiv:2103.00020* (2021).

[147] Rafael Rafailov et al. "Offline Reinforcement Learning from Images with Latent Space Models". In: *Learning for Decision Making and Control (L4DC)* (2021).

[148] Aravind Rajeswaran et al. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations". In: *arXiv preprint arXiv:1709.10087* (2017).

[149] Kanishka Rao et al. "RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real". In: (June 2020).

[150] Scott Reed et al. "A generalist agent". In: *arXiv preprint arXiv:2205.06175* (2022).

[151] Scott Reed et al. "Parallel Multiscale Autoregressive Density Estimation". In: *arXiv:1703.03664* (2017).

[152] Fereshteh Sadeghi and Sergey Levine. "Cad2rl: Real single-image flight without a single real image". In: *arXiv:1611.04201* (2016).

[153] Fereshteh Sadeghi et al. "Sim2real viewpoint invariant visual servoing by recurrent control". In: *Conference on Computer Vision and Pattern Recognition*. 2018.

[154] Pierre Sermanet et al. "Time-contrastive networks: Self-supervised learning from video". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1134–1141.

[155] Pratyusha Sharma et al. "Multiple interactions made easy (mime): Large scale demonstrations data for imitation". In: *arXiv:1810.07121* (2018).

[156] Michael Shea. "Karakuri: Subtle trickery in device art and robotics demonstrations at Miraikan". In: *Leonardo* 48.1 (2015), pp. 40–47.

[157] D. Sherer. "Fetal grasping at 16 weeks' gestation". In: *Journal of ultrasound in medicine* (1993).

[158] Noah Y Siegel et al. "Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning". In: *arXiv preprint arXiv:2002.08396* (2020).

[159] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv:1712.01815* (2017).

[160] Avi Singh et al. "COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning". In: *arXiv preprint arXiv:2010.14500* (2020).

[161] Avi Singh et al. "Parrot: Data-driven behavioral priors for reinforcement learning". In: *arXiv preprint arXiv:2011.10024* (2020).

[162] Shagun Sodhani, Amy Zhang, and Joelle Pineau. "Multi-Task Reinforcement Learning with Context-based Representations". In: *arXiv preprint arXiv:2102.06177* (2021).

[163] Yee Whye Teh et al. "Distral: Robust multitask reinforcement learning". In: *arXiv preprint arXiv:1707.04175* (2017).

[164] Gerald Tesauro. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68.

[165] Sebastian Thrun. "A lifelong learning perspective for mobile robot control". In: *Intelligent Robots and Systems*. 1995.

[166] Sebastian Thrun and Tom M Mitchell. "Lifelong robot learning". In: *Robotics and autonomous systems* (1995).

[167] Stephen Tian et al. "Manipulation by feel: Touch-based control with deep predictive models". In: *arXiv:1903.04128* (2019).

[168] Stephen Tian et al. "Model-Based Visual Planning with Self-Supervised Functional Distances". In: *arXiv preprint arXiv:2012.15373* (2020).

[169] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017.

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

[171] Antonio Torralba, Rob Fergus, and William T Freeman. "80 million tiny images: A large data set for nonparametric object and scene recognition". In: *transactions on pattern analysis and machine intelligence* (2008).

[172] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *arXiv:1607.08022* (2016).

[173] Mel Vecerik et al. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards". In: *arXiv preprint arXiv:1707.08817* (2017).

[174] Ruben Villegas et al. "High Fidelity Video Prediction with Large Neural Nets". In: ().

[175] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. "Generating videos with scene dynamics". In: *Neural Information Processing Systems*. 2016.

[176] Ziyu Wang et al. "Critic regularized regression". In: *arXiv preprint arXiv:2006.15134* (2020).

[177] Manuel Watter et al. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Neural Information Processing Systems*. 2015.

[178] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. "Scaling Autoregressive Video Models". In: *arXiv preprint arXiv:1906.02634* (2019).

[179] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. "Model Predictive Path Integral Control using Covariance Variable Importance Sampling". In: *CoRR* abs/1509.01149 (2015). arXiv: 1509.01149. URL: http://arxiv.org/abs/1509.01149.

[180] Grady Williams et al. "Aggressive driving with model predictive path integral control". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1433–1440.

[181] Aaron Wilson et al. "Multi-task reinforcement learning: a hierarchical bayesian approach". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 1015–1022.

[182] William Kurtz Wimsatt. "Poe and the chess automaton". In: *American Literature* 11.2 (1939), pp. 138–151.

[183] Bohan Wu et al. "Greedy hierarchical variational autoencoders for large-scale video prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2318–2328.

[184] Yifan Wu, George Tucker, and Ofir Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *arXiv preprint arXiv:1911.11361* (2019).

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

[186] Annie Xie and Chelsea Finn. "Lifelong robotic reinforcement learning by retaining experiences". In: *arXiv preprint arXiv:2109.09180* (2021).

[187] Annie Xie et al. "Few-Shot Goal Inference for Visuomotor Learning and Planning". In: *Conference on Robot Learning (CoRL)* (2018).

[188] Annie Xie et al. "Improvisation through Physical Understanding: Using Novel Objects as Tools with Visual Foresight". In: *CoRR* abs/1904.05538 (2019). arXiv: 1904.05538.

[189] SHI Xingjian et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Neural Information Processing Systems*. 2015.

[190] SHI Xingjian et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[191] Zhiyuan Xu et al. "Knowledge transfer in multi-task deep reinforcement learning for continuous control". In: *arXiv preprint arXiv:2010.07494* (2020).

[192] Mengjiao Yang, Sergey Levine, and Ofir Nachum. "TRAIL: Near-Optimal Imitation Learning with Suboptimal Data". In: *arXiv preprint arXiv:2110.14770* (2021).

[193] Mengjiao Yang and Ofir Nachum. "Representation matters: Offline pretraining for sequential decision making". In: *arXiv preprint arXiv:2102.05815* (2021).

[194] Ruihan Yang et al. "Multi-task reinforcement learning with soft modularization". In: *arXiv preprint arXiv:2003.13661* (2020).

[195] Sarah Young et al. "Visual imitation made easy". In: *arXiv e-prints* (2020), arXiv–2008.

[196] Kuan-Ting Yu et al. "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2016.

[197] Tianhe Yu et al. "Conservative data sharing for multi-task offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 34 (2021).

[198] Tianhe Yu et al. "Gradient surgery for multi-task learning". In: *arXiv preprint arXiv:2001.06782* (2020).

[199] Tianhe Yu et al. "How to Leverage Unlabeled Data in Offline Reinforcement Learning". In: *arXiv preprint arXiv:2202.01741* (2022).

[200] Tianhe Yu et al. "Mopo: Model-based offline policy optimization". In: *arXiv preprint arXiv:2005.13239* (2020).

[201] Tianhe Yu et al. "One-shot imitation from observing humans via domain-adaptive meta-learning". In: *arXiv preprint arXiv:1802.01557* (2018).

[202] Tianhe Yu et al. "Unsupervised Visuomotor Control through Distributional Planning Networks". In: *arXiv:1902.05542* (2019).

[203] Wenhao Yu, C. Karen Liu, and Greg Turk. "Preparing for the Unknown: Learning a Universal Policy with Online System Identification". In: *CoRR* abs/1702.02453 (2017). arXiv: 1702.02453.

[204] Andy Zeng et al. "Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning". In: *arXiv:1803.09956* (2018).

[205] Andy Zeng et al. "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics". In: *arXiv:1903.11239* (2019).

[206] Andy Zeng et al. "Tossingbot: Learning to throw arbitrary objects with residual physics". In: *IEEE Transactions on Robotics* 36.4 (2020), pp. 1307–1319.

[207] Marvin Zhang et al. "SOLAR: Deep Structured Latent Representations for Model-Based Reinforcement Learning". In: *arXiv:1808.09105* (2018).

[208] Tianhao Zhang et al. "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 5628–5635.

[209] Tinghui Zhou et al. "View synthesis by appearance flow". In: *European conference on computer vision*. 2016.

Appendix A

Visual Foresight

A.1 Skip Connection Neural Advection Model

Our video prediction model, shown in Figure 2.3, is inspired by the dynamic neural advection (DNA) model proposed by [54] and it is a variant of the SNA model proposed by [47]. The model uses a convolutional LSTM [189] to predict future frames. The prediction is initialized with an initial sequence of 2 ground truth frames, and predicts 13 futures frames. The model predicts these frames by iteratively making next-frame predictions and feeding those predictions back to itself. Each predicted frame, is given by a compositing layer, which composes intermediate frames with predicted compositing masks. The intermediate frames include the previous 2 frames and transformed versions them. To easily allow various transformations for different parts of the image, we predict 8 transformed frames, 4 of which are transformed from the previous frame, and the other 4 from the frame 2 steps in the past. These intermediate frames are then combined with compositing masks, which are also predicted by the convolutional LSTM. For simplicity, we collectively refer to these transformations as a single transformation $\hat{F}_{t+1 \leftarrow t}$ in Equation B.2. In addition, the first frame of the sequence is also given as one of the intermediate frames [47].

To enable action conditioning, the robot action at each time step is passed to all the convolutional layers of the main network, by concatenating it along the channel dimension of the inputs of these layers. Since they are vectors with no spatial dimensions, they are replicated spatially to match the spatial dimensions of the inputs.

The SNA variant that we use incorporate the architectural improvements proposed by [102]. Each convolutional layer is followed by instance normalization [172] and ReLU activations. We also use instance normalization on the LSTM pre-activations (i.e., the input, forget, and output gates, as well as the transformed and next cell of the LSTM). In addition, we modify the spatial downsampling and upsampling mechanisms. Standard subsampling and upsampling between convolutions is known to produce artifacts for dense image generation tasks [132, 131]. In the encoding layers, we reduce the spatial resolution of the feature maps by average pooling, and in the decoding layers, we increase the resolution by

using bilinear interpolation. All convolutions in the generator use a stride of 1. The actions are concatenated to the inputs of all the convolutional layers of the main network, as opposed to only the bottleneck.

We provide an example of the skip connection neural advection (SNA) model recovering from occlusion in Figure A.2. In this figure, the arm is predicted to move in front of the designated pixel, marked in blue in Figure A.1. The predictions of the DNA model, shown in figure Figure A.2(b), contain incorrect motion of the marked object, as shown in the heatmaps visualizing $\hat{P}_t$, although the arm actually passes in front of it. This is because the DNA model cannot recover information about an object that it has 'overwritten' during its predictions, causing the model to predict that the pixel *moves with the arm*. We identified this as one of the major causes of planning failure using the DNA model. By contrast our SNA model predicts that the occluded object will not move, shown in figure Figure A.2(a).

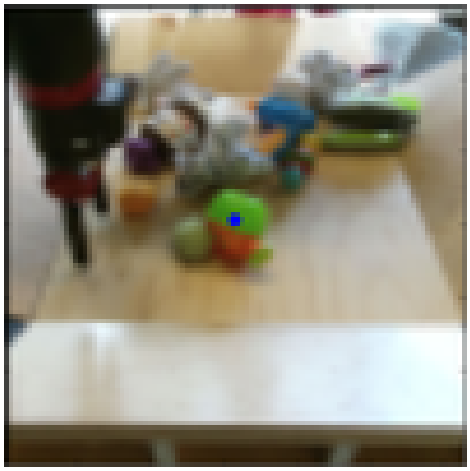


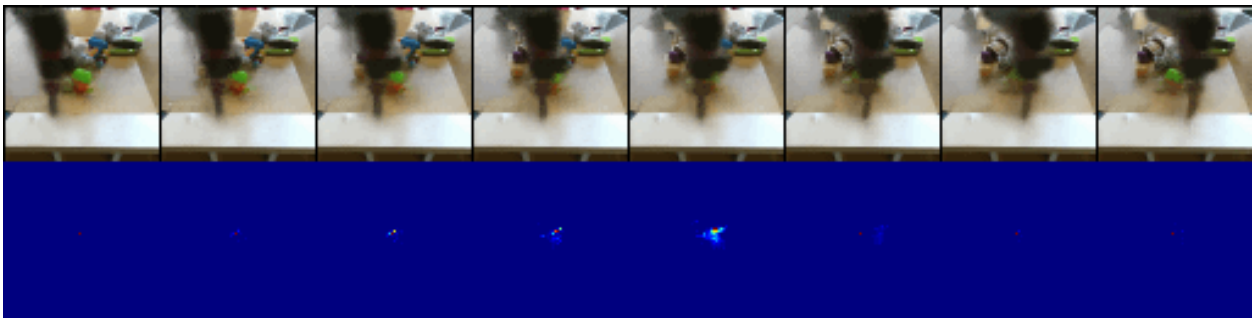
Figure A.1: The blue dot indicates the designated pixel

A.2 Improved Action Sampling Distributions for Data Collection

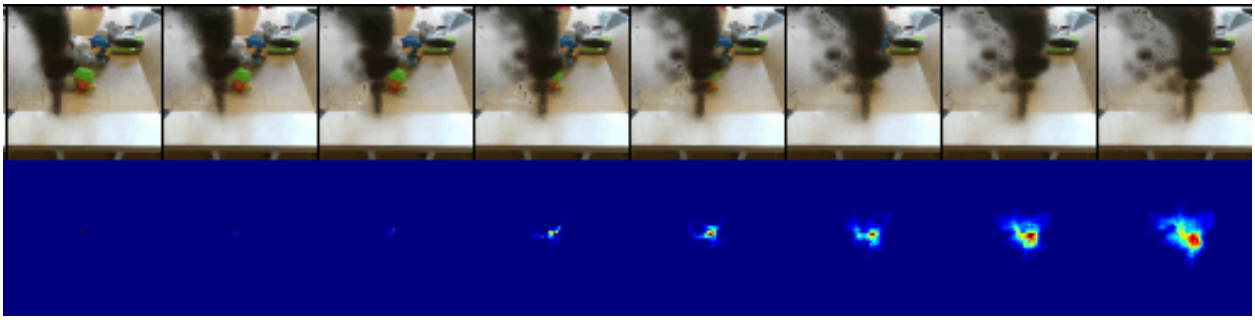
In order to collect meaningful interaction data for learning folding of deformable objects such as towels and cloth, we found that the grasping primitive can be slightly adapted to increase the likelihood of encountering states where cloths are actually folded. When using actions sampled from a simple distribution or the previously-described distribution, clothing would become tangled up. To improve the efficiency of folding cloths we use an action primitive similar to the grasping primitive, but additionally we reduce lateral motion of the end-effector when the gripper is close to the table, thus reducing events where cloths become tangled up.

A.3 Improvements of the CEM-Optimizer

In the model-predictive control setting, the action sequences found by the optimizer can be very different between execution real-world times steps. For example at one time step the optimizer might find a pushing action leading towards the goal and in the next time step it determines a grasping action to be optimal to reach the goal. Naïve replanning at every time step can then result in alternating between a pushing and a grasping attempt indefinitely causing the agent to get stuck and not making any progress towards to goal.



((a)) Skip connection neural advection (SNA) does not erase or move objects in the background



((b)) Standard DNA [56] exhibits undesirable movement of the distribution $P_d(t)$ and erases the background

Figure A.2: Top rows: Predicted images of arm moving *in front of* green object with designated pixel (as indicated in Figure A.1). Bottom rows: Predicted probability distributions $P_d(t)$ of designated pixel obtained by repeatedly applying transformations.

Figure A.3: Applying our method to a pushing task. In the first 3 time instants the object behaves unexpectedly, moving down. The tracking then allows the robot to retry, allowing it to eventually bring the object to the goal.

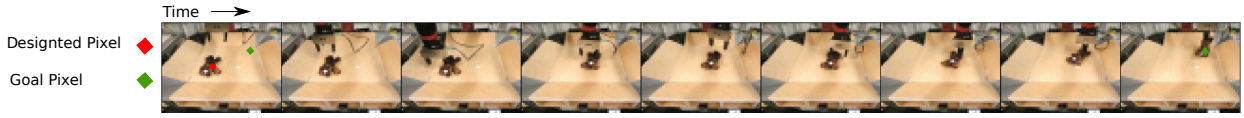


Figure A.4: Retrying behavior of our method combining prehensile and non-prehensile manipulation. In the first 4 time instants shown the robot pushes the object. It then loses the object, and decides to grasp it pulling it all the way to the goal. Retrying is enabled by applying the learned registration to both camera views (here we only show the front view).

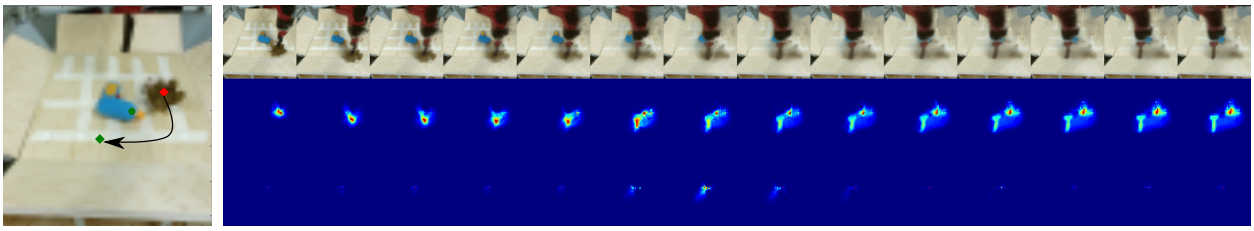


Figure A.5: Left: Task setup with green dot marking the obstacle. Right, first row: the predicted frames generated by SNA. Second row: the probability distribution of the designated pixel on the *moving* object (brown stuffed animal). Note that part of the distribution shifts down and left, which is the indicated goal. Third row: the probability distribution of the designated pixel on the obstacle-object (blue power drill). Although the distribution increases in entropy during the occlusion (in the middle), it then recovers and remains on its original position.

We can resolve this problem by modifying the sampling distribution of the first iteration of CEM so that the optimizer commits to the plan found in the previous time step. In the simplest version of CEM the sampling distribution at first iteration of CEM is chosen to be a Gaussian with diagonal covariance matrix and zero mean. We instead use the best action sequence found in the optimization of the *previous* real-world time step as the mean for sampling new actions in the *current* real-world time-step. Since this action sequence is optimized for the previous time step we only use the values $a_{2:T}$ and omit the first action. To sample actions close to the action sequence from the previous time step we reduce the entries of the diagonal covariance matrix for the first $T-1$ time steps. It is crucial that the last entry of the covariance matrix at the end of the horizon is not reduced otherwise no exploration could happen for the last time step causing poor performance at later time steps.

A.4 Experimental Setup

To train both our video-prediction and registration models, we collected 20,000 trajectories of pushing motions and 15,000 trajectories with gripper control, where the robot randomly picks and moves objects using the grasping reflex described in section 2.7. The data collection process is fully autonomous, requiring human intervention only to change out the objects in front of the robot. The action space consisted of relative movements of the end-effector in cartesian space along the x, y, and z axes, and for some parts of the dataset we also added azimuthal rotations of the gripper and a grasping action.

A.5 Experimental Evaluation

Figure A.5 shows an example of the SNA model successfully predicting the position of the obstacle through an occlusion and finding a trajectory that avoids the obstacle.

A.6 Simulated Experiments

In order to provide a more controlled comparison, we also set up a realistic simulation environment using MuJoCo [170], which includes a robotic manipulator controlled via Cartesian position control, similar to our real world setup, pushing randomly-generated L-shaped objects with random colors (see details in supplementary materials). We trained the same video prediction model in this environment, and set up 50 evaluation tasks where blocks must be pushed to target locations with maximum episode lengths of 120 steps. We compare our proposed registration-based method, "predictor propagation," and ground-truth registration obtained from the simulator, which provides an oracle upper bound on registration performance.

Appendix B

RoboNet

B.1 Visual Foresight Preliminaries

Here we give a brief introduction into the visual foresight algorithm used in this paper, see [57, 48, 46] for a more detailed treatment.

Action conditioned video-prediction model

The core of visual foresight is the action conditioned video-prediction model, which is a deterministic variant of the model described in [103]. The model is illustrated in Figure B.1 and implemented as a recurrent neural network using actions $a_{0:T}$, and images $I_{0:T}$ as inputs and outputting future predicted images $\hat{I}_{1:T}$. Instead of using a context of 1 as shown in Figure B.1, a longer context can be used which increases the model's ability to adapt to environment variation such as held-out view-points. In all experiments in this paper we used a context of 2 frames. Longer contexts can potentially help the model adapt to unseen conditions in the environment, however, this is left for future work. The RNN is unrolled according to the following equations:

$$[h_{t+1}, \hat{F}_{t+1\leftarrow t}] = g_\theta(a_t, h_t, I_t) \tag{B.1}$$

$$\hat{I}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t \tag{B.2}$$

Here $g_\theta(a_t, h_t, I_t)$ is a forward predictor parameterized by θ and $\hat{F}_{t+1\leftarrow t}$ is two-dimensional flow field with the same size as the image which is used to transform an image from one time-step into the next via bilinear-sampling.

The architecture of the RNN, which is illustrated in Figure B.2, uses a stack of convolutional LSTMs [190] interleaved with convolution layers, skip connection help the learning process.

Training details For pretraining all models are trained for 160k iterations using a batchsize of 16. For SGD we use the Adam optimizer. Finetuning is carried out for another

150k steps. The learning rate starts at 1e-3 and is annealed linearly to 0 after 200k steps until the end of training.

Sampling-based Planning

In visual foresight tasks are specified in terms of the motion of user-selected pixels. To predict where pixels move in response to a sequence of actions, we define a probability distribution P_0 over the locations of the designated pixel. At time step 0 this we use a one-hot-distribution with 1 a the user-selected pixel and 0 everywhere else. When then apply the same transformations to these distributions that we also apply to the images. This is summarized in the following equation:

$$\hat{P}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{P}_t \tag{B.3}$$

Here $\hat{P}_{t+1}$ denotes the predicted probability distribution of the designated pixel.

The planning cost is computed as the expected distance to the goal pixel position d_g under the predicted distribution $\hat{P}_t$, averaged over time:

$$c = \sum_{t=1,\ldots,T} c_t = \sum_{t=1,\ldots,T} \mathbb{E}_{\hat{d}_t \sim P_t} \left[\|\hat{d}_t - d_g\|_2 \right] \tag{B.4}$$

To find the optimal action sequence we apply the model-predictive path intregral (MPPI) [179] algorithm, since this allows us to find good actions sequences more efficiently than random

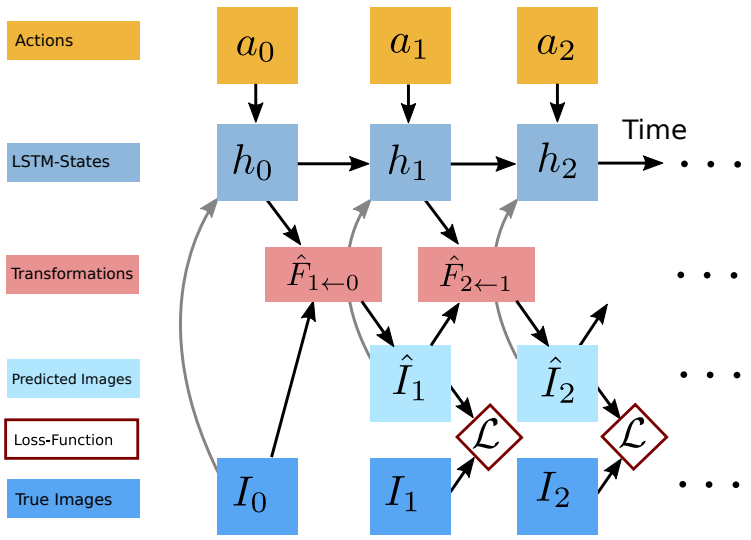


Figure B.1: Recurrent dynamics model for action-conditioned video-prediction based on flow transformations. (Used with permission from [49])

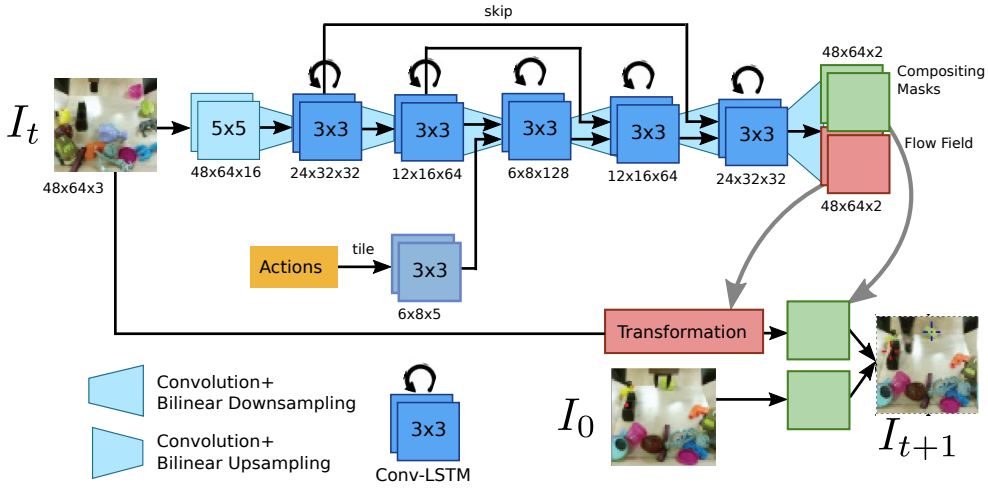


Figure B.2: Architecture of the recurrent video-prediction architecture. (Used with permission from [49])

shooting. In the first iteration the actions are sampled from a unit Gaussian, in subsequent iterations the mean action is computed as an exponential weighted average as follows:

$$\mu_t = \frac{\sum_{k=0}^{N} e^{-\gamma c_k} a_{k,0:T}}{\sum_{k=0}^{N} e^{-\gamma c_k}} \tag{B.5}$$

Here N is the number of samples, chosen to be 600. The prediction horizon is 15 steps. We found that a number of 3 MPPI iterations works best in our settings. We apply temporal smoothing to the action samples using a low-pass filter to achieve smoother control and better exploration of the state space.

After finding an action sequence, the first action of this sequence is applied to the robot and the planner is queried again, thus operating in an MPC-like fashion. In order to perform re-planning, it is required to know the current position of the designated pixel. In this work we use a simple method for obtaining an estimate for the designated pixel by using the model's prediction, i.e. the flow maps from the previous time-step, we call this predictor propagation. While this position estimate is noisy and more complex alternatives, such as hand-engineered trackers or self-supervised registration [46] exist, we opt for the simple approach in this work.

B.2 Data Collection Details

State and Action Space

Most of the robots in RoboNet (excluding Google R3 from [55]) employ the same Cartesian end-effector control action space with restricted rotation, and a gripper joint. At each time-

step, the state is a $\mathbb{R}^5$ vector containing the grippers XYZ position, the gripper's yaw angle (rest of orientation is locked, with the gripper pointed downwards), and the gripper joint-angle value. The user must specify safety bounds per-robot, which constrain the end-effector to operate within a "safe" region of space at all time-steps. Actions are specified as deltas between the current state and commanded state for the next time-step. Note that the gripper action is binarized to "open" or "close" for simplicity. Actions are blocking with a set time-out, so user defined policies only receive states and calculate actions once the robot has reached (or gotten as close as possible to) the commanded state. There are no "real-time" constraints on the user policy. As a result, the robot must come to a complete stop at each step. While this scheme can easily generalize to new robots, it does impose constraints on the final robot behavior. We hope to relax these constraints in future work.

Exploration Policy

During data collection, actions are either sampled from a simple diagonal Gaussian with one dimension per action-space dimension, or a more sophisticated distribution that biases the probability of grasping when the gripper is at the table height, increasing the chance that the robot will randomly grasp objects. This primitive is described further below. The variances in the diagonal Gaussians are hand-tuned per robot and differ between different action dimensions. The exact parameters are stored in inside the hdf5-files under the field `policy-description`.

While using a simple action distribution such as a diagonal Gaussian, the robot arm frequently pushes objects, however the arm quite rarely grasps objects. In order for a learning algorithm such as visual foresight to effectively model grasping, it must have seen a sufficient number of grasps in the dataset. By applying a grasping primitive, such as the one originally introduced in [46], the likelihood for these kinds of events can be increased. The grasping primitive is implemented as a hard-coded rule that closes the gripper when the z-level of the end-effector is less than a certain threshold, and opens the gripper if the arm is lifted above a threshold while not carrying an object.

There are, however, two robots in this dataset which employ significantly different exploration policies. The Google R3 robot samples random pushing motions instead of simply taking random Cartesian motions, and the Fetch robot data only contains random exploration in the x and y dimensions.

B.3 Database Implementation Details

The database stores every trajectory as a separate entity with a set of attributes that can be filtered. We provide code infrastructure that allows a user to filter certain subsets of attributes for training and testing. The database can be accessed using the Pandas python-API, a popular library for structuring large amounts of data. Data is stored in the widely

adopted `hdf5`-format, and videos are encoded via MP4 for efficiency reasons. New trajectory attributes can be added easily.

B.4 Description of Benchmarking Tasks

For all control benchmarks we used object relocation tasks from a set of fixed initial positions towards a set of fixed goal positions marked on a table. The experimental setups for each robot are depicted in Figure B.3. After executing the action sequences computed by the algorithm the remaining distance to the goal is measured using a tape, and success is determined by human judges.

Figure B.3: Experimental setups for benchmarking tasks on the Kuka, Franka, and Sawyer robots.

B.5 Experimental evaluation of adaptation to unseen gripper

We evaluate on a domain where a Sawyer robot is equipped with a new gripper that was not seen in the dataset. We collected 300 new trajectories with a Robotiq 2-finger gripper, which differs significantly in visual appearance and dimensions from the Weiss Robotics gripper used in all other Sawyer trajectories (see Figure 3.2), and used this data to evaluate four different models: zero-shot generalization for a model trained on RoboNet, a model trained only on the new data, a model pre-trained on only the Sawyer data in RoboNet and then finetuned with the new data, and a model pre-trained on all of RoboNet and finetuned with the new data. The results qualitative results of this evaluation are shown in Figure B.4 and the quantitative results are in Table B.1, averaging over 10 trajectories each. The best-performing model in this case is the one that is pretrained on only the Sawyer data, and it attains performance that is comparable to in-domain generalization (see, e.g., the seen viewpoint result in Table 3.2 for a comparison). The model pre-trained on the more diverse

RoboNet dataset actually performs worse, likely due to the limited capacity and underfitting issues discussed in Section 3.5. However, these results do demonstrate that visual foresight models can adapt to moderate morphological changes using a modest amount of data.

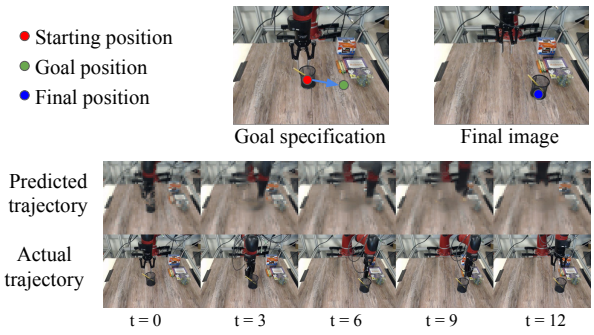


Figure B.4: Example task of pushing an object with an unseen gripper, in this case the Robotiq gripper.

	Avg. distance (cm)
zero-shot	15.5 ± 2.6
without pretraining	17 ± 1.8
pretraining on Sawyer-only	**9.8 ± 2.1**
pretraining on all of RoboNet	14.7 ± 2.1

Table B.1: Evaluation results for adaptation to Robotiq gripper with the Sawyer arm. The model trained on only Sawyer data performs the best when fine-tuned on 300 trajectories with a Robotiq gripper.

Appendix C

Bridge Data

Task	**Tar. Env**	**Joint Train**	**Single Task**	**Potential reason for no gain with bridge data**
turn faucet lever (1) appearance from the other faucets	tk2	0%	0%	[t]The faucet in tk2 has very different
Pickup pan from stove (2)	ts1	0%	N/A	Not enough target domain data and prior data for this task
Put spoon into pot (2)	tk2	0%	N/A	Not enough target domain data and prior data for this task
flip orange pot upright (3) only metal pots in prior dataset	tk2	50%	60%	[t]There is no orange pot in prior dataset,
open box flaps (3)	tk2	10%	10%	Boxes and pushing motions do not occur in prior data
take lid off pot (3)	ts3	60%	60%	Only 100 demos involving lids in prior dataset.
pick up screw driver (3) visually very different from prior data	toolchest	0%	0%	[t]The toolchest and screwdrivers are
put pot or pan in sink (1)	tk2	90%	50%	
put carrot on plate (2)	ts1	40%	N/A	
Wipe plate w/ sponge (3)	k1	70%	N/A	
put pear in bowl (3)	tk2	50%	10%	
put brush in pot (3)	ts3	90%	0%	
put detergent dry rack (3)	ts3	80%	10%	
lift bowl (3)	tk2	70%	50%	

Figure C.1: Comparison of scenarios where usage of the bridge data helps performance and where it does not. Scenarios where usage of bridge data does not help are marked in red font. The type of transfer setting is denoted by the number in brackets after the task description.

Appendix D

Pre-training For Robots

D.1 Details of Our Experimental Setup

Real-World Experimental Setup

A picture of our real-world experimental setup is shown in Figure D.1. The scenarios considered in our experiments (Section 5.5) 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 D.1) on two different WidowX250 robot arms. We use data from the bridge dataset ebert2021bridge 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.

Diagnostic Experimental Setup in Simulation

In simulation, we evaluate our approach in a simulated bin-sorting task on the simulated WidowX250 platform, aimed to mimic the setup we use for our real-world evaluations. This setup is designed in the PyBullet simulation framework provided by singh2020cog. A picture is shown in Figure D.2. 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 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 iden-

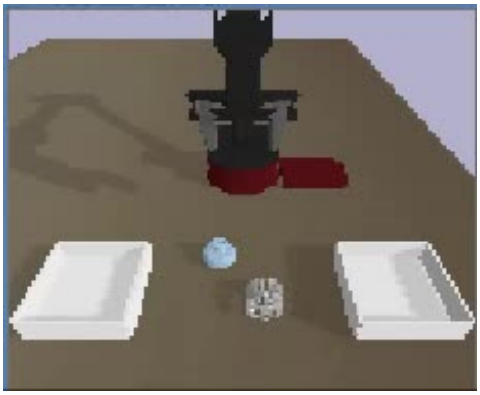


Figure D.2: Bin-Sorting task used for our simulated evaluations. The task requires sorting the cylinder into the left bin and the teapot into the right bin.

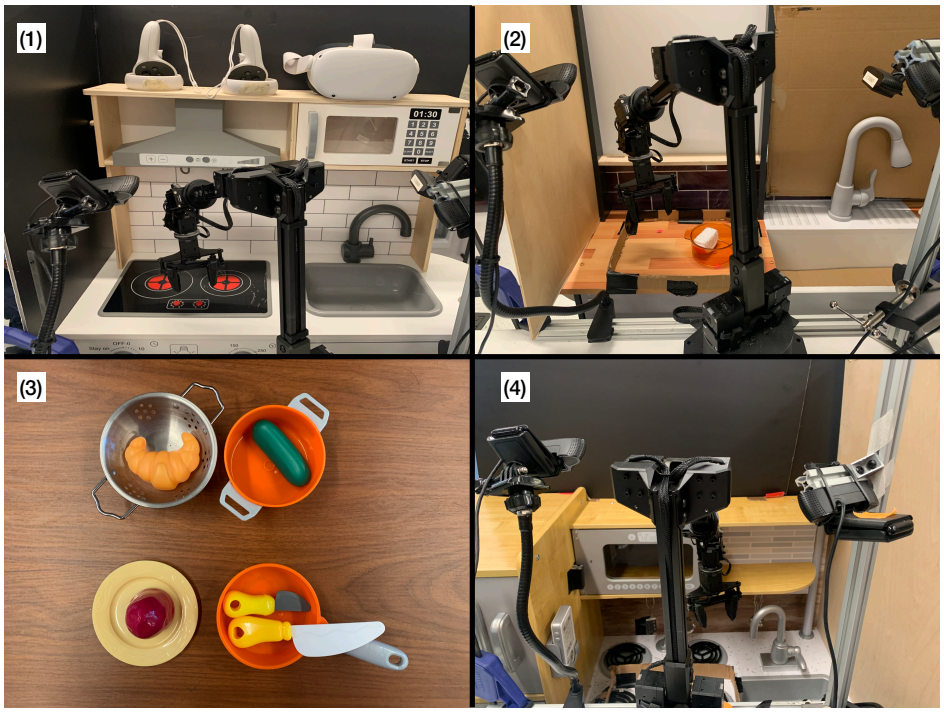


Figure D.1: Setup Overview: Following [45], we use a toykitchen setup described in that prior work for our experiments. This utilizes a 6-DoF WidowX 250 robot. (1): Held-out toykitchen used for experiments in Scenario 3 (denoted "toykitchen 6"), (2): Re-targeting toykitchen used for experiments in Scenario 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").

tifying the correct bin associated with each object. A 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 are shown in Figure D.3. While the downstream task only requires solving this sorting task with two specific objects (shown in Figure D.4), the pre-training data consists of a 10 unique objects (some shown in Figure D.3). 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 objects is placed one of the bins while the other one is not. This is what makes this task

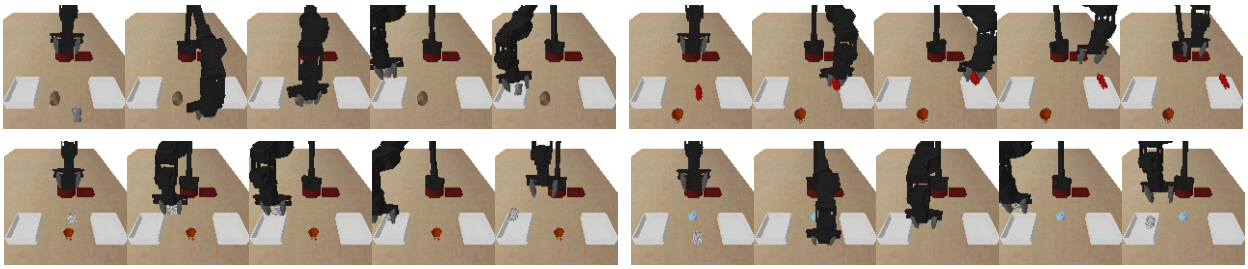


Figure D.3: Some trajectories from the pre-training data used in the simulated bin-sort task.

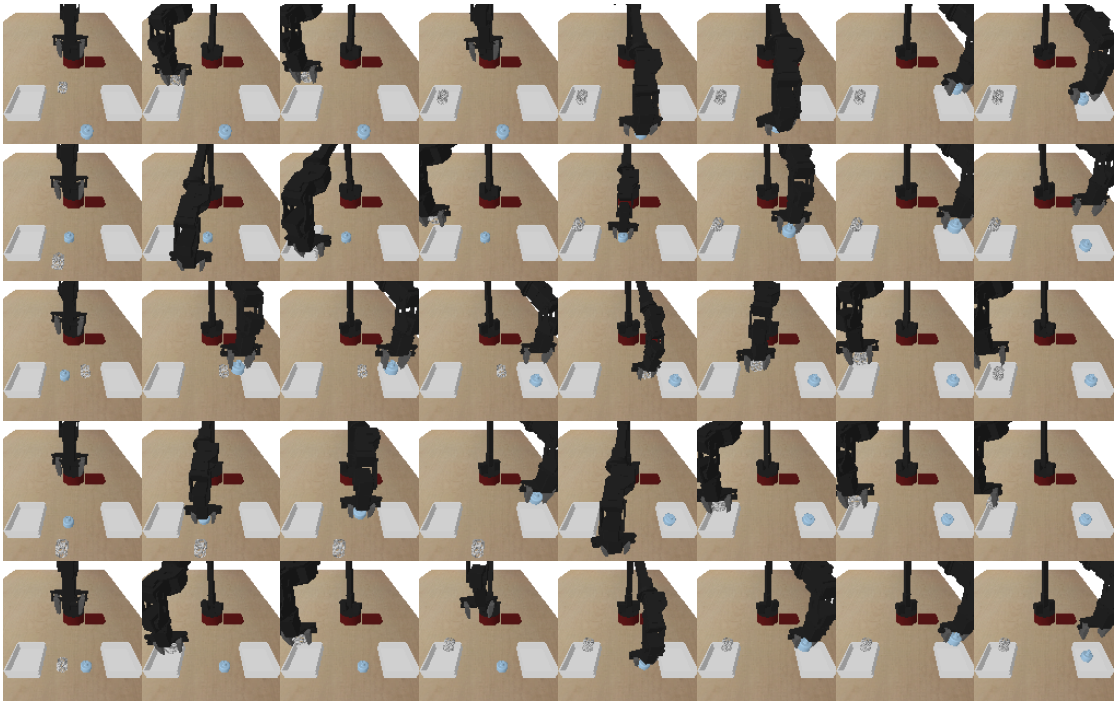


Figure D.4: The five demonstration trajectories used for Phase 2 of PTR.

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 D.4). 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 PTRattains a substantially better success rate than other baselines in this exact step.

Figure D.5: 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.

D.2 Description of the Real-World Evaluation Scenarios

In this section, we describe the real-world evaluation scenarios considered in Section 5.5. We additionally include a much more challenging version of Scenario 3, for which we present results in Appendix D.3. These harder test cases evaluate the finetuning 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, nomenclature for the toy kitchens is drawn from ebert2021bridge and as described in the caption of Figure D.1.

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

Pre-training data. The pre-training data comprises of all of the pick and place data from the bridge dataset ebert2021bridge from toykitchen 2. This includes data corresponding to the task of putting the sushi in the transparent orange pot (Figure D.5).
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 D.5).
Quantitative evaluation protocol. For our quantitative evaluations in Table 5.1, we run 10 controlled evaluation rollouts that place the sushi and the metallic pot in different

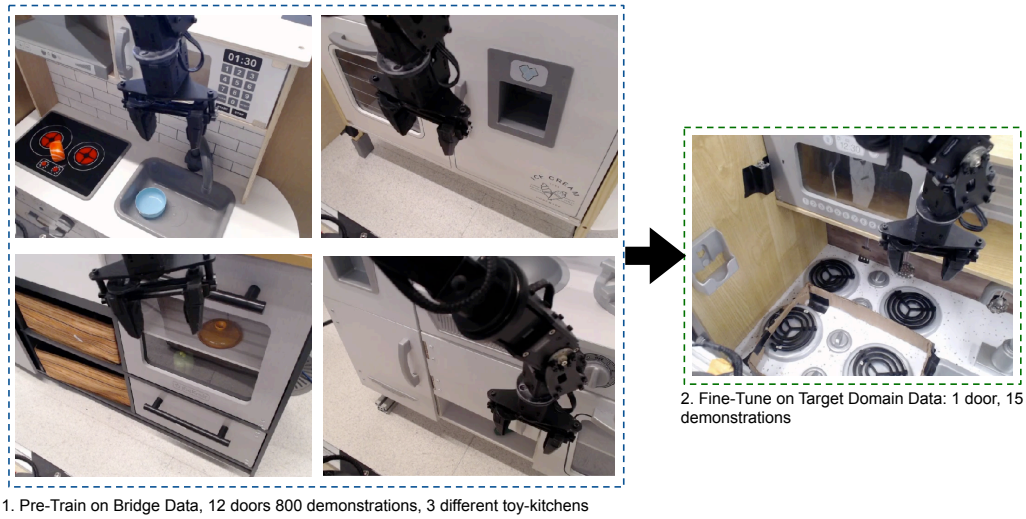


Figure D.6: Illustration of pre-training data and fine-tuning data used for Scenario 2 (door opening): transferring a behavior to a held-out domain.

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.

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 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.

Scenario 3: Learning to Solve New Tasks in New Domains

Pre-training data. All pick-and-place data in the bridge dataset ebert2021bridge except any demonstration data collected in toykitchen 6, where our evaluations are performed.
Target task and data. The target task requires placing a 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.

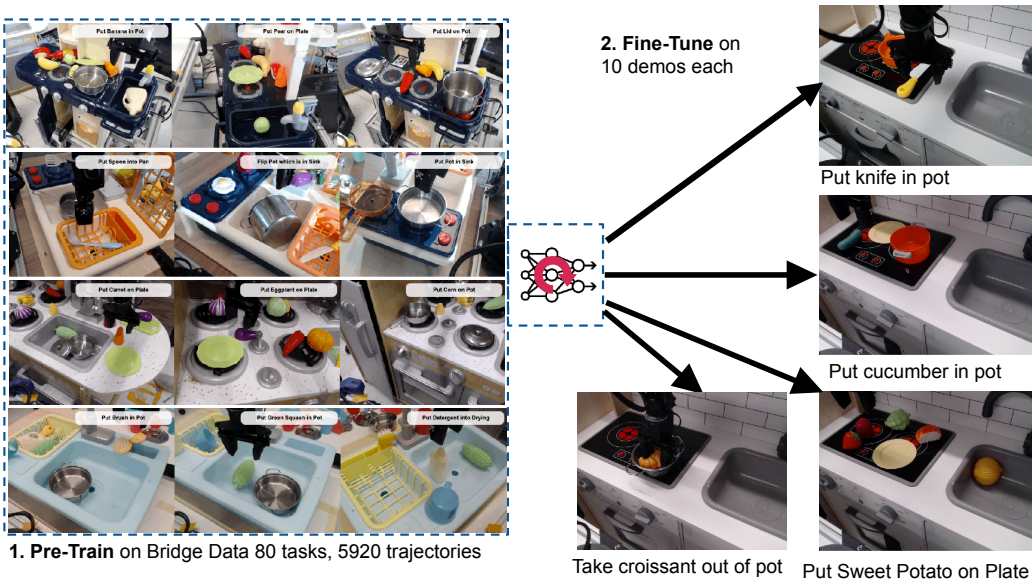


Figure D.7: Illustration of pre-training data and fine-tuning data used for the new tasks we have added in Scenario 3. The goal is to learn to solve new tasks in new domains starting from the same pre-trained initialization and when fine-tuning is only performed using 10-20 demonstrations of the target task.

Quantitative evaluation protocol. During 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 pose for each method. We attempted to match the positions of objects across methods as closely as possible.

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

In Appendix D.3, 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 ebert2021bridge except data from toykitchen 6.

Target task and data. We consider four downstream tasks: take croissant from a metallic bowl, put sweet potato on plate, place knife in pot, and put cucumber in bowl, and collected

10 target demonstrations for 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 D.7.

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 PTRstarting 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 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 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 present some unique challenges, and evaluating both of them allows us to guage how robust the learned policy is.
- **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 D.1).
- **Put cucumber in bowl:** We run 10 evaluation rollouts starting from 10 randomly sampled initial poses of the robot for our evaluations.

We will discuss the results obtained on these new tasks in Appendix D.3.

D.3 Additional Experimental Results

In this section we present additional experiments for each of the three scenarios, further corroborating the results introduced in the main experimental section of this paper. We will then present some ablations of the key practical design decisions in Appendix D.5.

Additional Baseline Comparisons in Scenario 2

In this section, we present results for some additional baselines from the open door task in a new toykitchen domain (Scenario 2 in the main paper). The results are shown in Table D.1.

Observe that not only does PTRoutperform BC (finetuning), it also outperforms joint training for both CQL and BC, and also just training on only the target data from scratch with either method. This indicates that PTRis quite effective compared to other baselines, including the best BC method, even with demonstration data, where BC is expected to be optimal.

			0-shot		**joint**		**Target data only**	
Task	**PTR(Ours)**	**BC (fine.)**	**CQL**	**BC**	**CQL**	**BC**	**CQL**	**BC**
Open Door	**60%**	50%	0%	0%	25%	35%	20%	35%

Table D.1: Performance of PTR and baselines on scenario 2, generalizing to previously unseen domains (opening new door in new domain). PTR attains the highest success rate. 0-shot, i.e. only using bridge data, does not work at all. Jointly training with target averaged over 20 independently chosen evaluation trials also performs much worse than PTR.

More interestingly, note that while PTRoutperforms the best BC baseline, joint training with CQL does not quite outperform BC (joint). This is interesting for two reasons: first, the fact that BC outperforms CQL with joint training is expected and shows that our BC baseline is functioning as expected in this regime. Second, it shows that while offline RL methods naïvely may not quite outperform BC with demonstration data, running multi-task offline RL on diverse demonstration data can still be useful because fine-tuning from the resulting initialization on a target task can actually lead to better performance compared to BC (either joint trained or fine-tuned).

More Evaluations of PTRin the Additional Tasks in Scenario 3

As discussed in Appendix D.2, the new tasks in Scenario 3 are designed to test the efficacy of PTRin solving new tasks in new domains. The four downstream tasks we consider are: take croissant from metallic bowl, put sweet potato on plate, keep knife on pot, and put cucumber in bowl. All of these are setup in the same new toy kitchen environment (called "toykitchen6") that was held out from the pre-training dataset. The details of this setup including the number of demonstrations in the target data, the evaluation protocol for obtaining quantitative results, and illustrations of the tasks are presented in Appendix D.2. Here we discuss the results.

The results for these tasks are presented in Table D.2. Observe that PTRattains the highest success rate on every task, and in many tasks outperforms the next best baseline by about **2x**. Observe that while **CQL (joint)** outperforms **BC (joint)** generally, it still does not attain as high success rates as PTR. Completely skipping pre-training and just training on 10-20 target task demonstrations only can be effective in some cases with CQL (see **CQL (Target data only)** on place knife in pot) but is substantially worse with both CQL and

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

Table D.2: Performance of PTRand other baseline methods for new tasks in Scenario 3. Note that PTRoutperforms all other baselines including BC (finetune), training on the target demonstration data only with no pre-training ("Target data only") and jointly training on all the pre-training data and the target demonstration data ("Joint training").

BC in general. Finally, we find **BC (finetune)** attains similar success rates as **BC (joint)**, and is also worse than PTR.
Again note that the gap in performance between PTRand BC (finetune) is much larger than the gap between **CQL (joint)** and **BC (joint)** indicating that applying the fine-tuning procedure does lead to a larger boost for multi-task offline RL compared to BC.

Why Does PTROutperform 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 PTRoutperform 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 ebert2021bridge for tuning BC as we discuss in Appendix D.4. 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 ebert2021bridge, 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 PTRto 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 PTRoutperforms the baseline.
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 PTRas film strips in Figure D.8. Specifically, we visualize evaluation rollouts that present a challenging initial state. For example, for the rollout from the take crossiant out of metallic pot task, the robot must

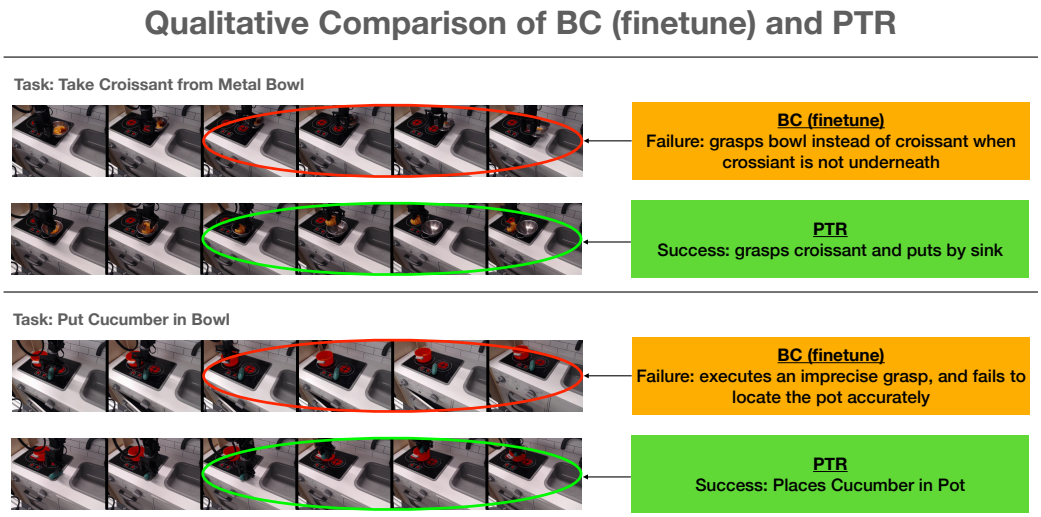


Figure D.8: Qualitative successes of PTRvisualized alongside failures of BC (finetune). As an example, observe that while PTRis accurately able to reach to the croissant and grasp it to solve the task, BC (finetune) is imprecise and grasps the bowl instead of the croissant resulting in failure.

first accurately position itself over the crossiant 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 D.8 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, PTRis 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 kumar2022should. Intuitively, this argument suggests that in tasks where the robot must precisely accomplish actions at only 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 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 at these states. In such scenarios, the argument of kumar2022should 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 succeed.

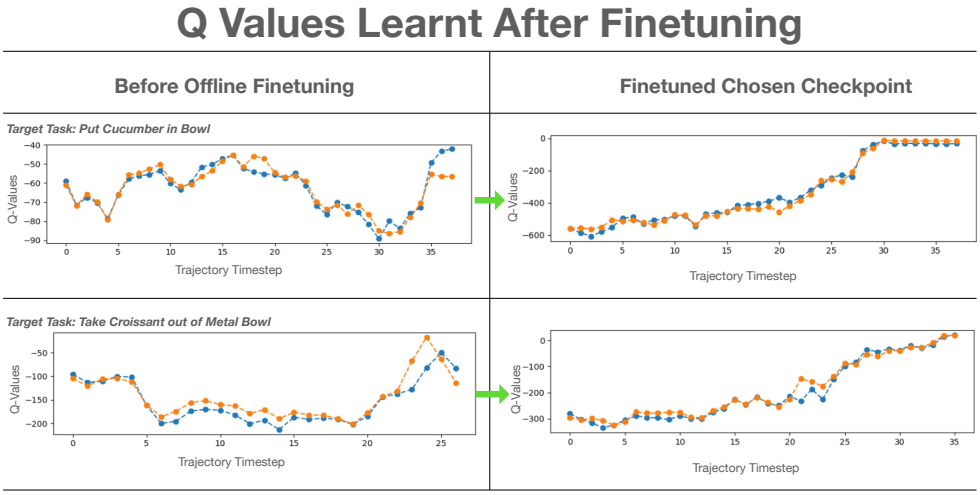


Figure D.9: Evolution of Q-values on the target task over the process of fine-tuning with PTR. Observe that while the learned Q-values on *held-out* trajectories from the dataset just at the beginning of Phase 2 (finetuning) do not exhibit a roughly increasing trend, the checkpoint of PTRwe choose to 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 PTRduring fine-tuning in Figure D.3. 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 to 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.

Complete Results In Simulation

Finally, we present some additional results comparing BC-based methods and CQL to PTRin our simulated bin-sorting task. Recall that goal of this task was to solve a two-stage, compound task with only *five* target demonstrations as discussed in Figure D.2. Most importantly, the pre-training data does not show any instance of the robot attempting to solve this two-stage task.

The performance numbers (along with 95%-confidence intervals) are shown in Table D.3. Observe that PTRimproves upon prior methods in a statistically significant manner, outperforming the BC baselines by a significant margin. This validates the efficacy of PTRin simulation, and corroborates our real-world results.

Method	**Success rate**
BC (joint training)	7.00 ± 0.00 %
CQL (joint training)	8.00 ± 1.00 %
BC (finetune)	4.88 ± 4.07 %
PTR(Ours)	**17.41 ± 1.77 %**

Table D.3: Performance of PTRin 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 PTRsubstantially 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, and so we do not report it in this table. Since the 95%-confidence intervals do not overlap between PTRand other methods, it indicates that PTRimproves upon baselines in a statistically significant manner.

D.4 Hyperparameters for PTRand Baseline Methods

In this section, we will present the hyperparameters we use in our experiments and explain how we tune the other hyperparameters for both our method PTRand the baselines we consider.

PTR. Since PTRutilizes 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 singh2020cog, and are presented in Table D.4 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 D.5.

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

Table D.4: Main hyperparameters for CQL training in our real-world experiments. In simulation, we utilize a smaller α for CQL, $\alpha = 1.0$ and a larger discount $\gamma = 0.98$ since trajectories in simulation are about 60-70 timesteps in length.

The only other hyperparameter used by PTRis 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 PTRin 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 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 5.4. We did not tune any other algorithmic hyperparameters for CQL, as these were taken directly from singh2020cog.

BC (finetune). We trained BC in a similar manner as ebert2021bridge, 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 ebert2021bridge. Following the recommendations of ebert2021bridge 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 ebert2021bridge,emmons2021rvs 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 PTRdiscussed 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 PTRwas 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.

D.5 Validating the Design Choices from Section 5.4 via Ablations

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 PTRscale 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 PTRwith varying sizes and architectures on three tasks: the open door task from Scenario 2, and the put cucumber in 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, kostrikov2021offline), **(b)** an IMPALA espeholt2018impala 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 D.10. To obtain more accurate scaling trends, we plot the trend in the av-

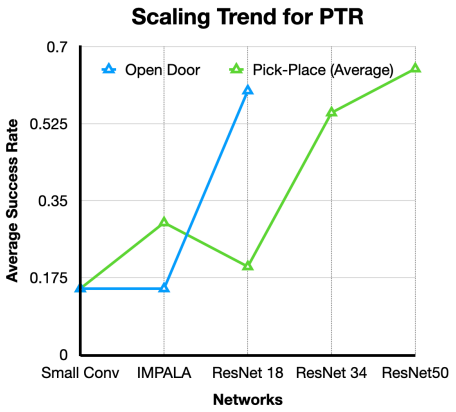


Figure D.10: Scaling trends for PTRon the open door task from Scenario 2, and average over two pick and place tasks (take croissant out of metallic pot and put cucumber in bowl) from Scenario 3. Note that more high capacity and expressive function approximators lead to the best results.

erage 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 possiblity 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 lee2022multi). 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 ebert2021bridge, 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 D.5: Ablation of PTRwith spatial softmax and GAP on the croissant task. Observe that PTRwith learned spatial embeddings performs significantly better than using a spatial softmax or global average pooling.

As shown in Table D.5 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 PTRis required for attaining good performance.

Concatenating actions at each layer is crucial for performance. Next, we run PTRwithout 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 PTRwhich passes in actions at each layer and succeeds in **7/10** evaluation rollouts (Table D.6).

Group normalization is more consistent than batch normalization. Next, we ablate

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 D.6: Ablation of PTRwith actions passed in at each layer. Observe that passing in actions at each fully-connected layer does lead to quite good performance.

the usage of group normalization over batch normalization in the ResNet 34 Q-functions that PTRuses. 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 D.7, and they demonstrate that batch normalization may not be as consistent and reliable with PTRas group normalization.

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 D.7: Relative performance of PTRwith batch normalization with respect to PTRwith group normalization. Observe that while utilizing batch normalization in PTRcan be sometimes more effective than using batch 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 → corresponds to the performance of PTRwith group normalization and the performance to the right of → is the performance with batch normalization.

Choice of reward function. Finally, we present some results that ablate the choice of the reward function utilized for training PTRfrom 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.