VI. Background

a) Reinforcement Learning (RL): We study RL as a discounted infinite-horizon Markov Decision Process (MDP). For pixel observations, the agent's state is approximated as a stack of consecutive RGB frames. The MDP is of the form $(\mathcal{O}, \mathcal{A}, P, R, \gamma, d_0)$ where $\mathcal{O}$ is the observation space, $\mathcal{A}$ is the action space, $P : \mathcal{O} \times \mathcal{A} \rightarrow \Delta(\mathcal{O})$ is the transition function that defines the probability distribution over the next state given the current state and action, $R : \mathcal{O} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, γ is the discount factor and d_0 is the initial state distribution. The goal is to find a policy $\pi : \mathcal{O} \rightarrow \Delta(\mathcal{A})$ that maximizes the expected discounted sum of rewards $\mathbb{E}_\pi[\Sigma_{t=0}^{\infty} \gamma^t R(\boldsymbol{o}_t, \boldsymbol{a}_t)]$, where $\boldsymbol{o}_0 \sim d_0$, $a_t \sim \pi(\boldsymbol{o}_t)$ and $\boldsymbol{o}_{t+1} \sim P(.|\boldsymbol{o}_t, \boldsymbol{a}_t)$.

b) Imitation Learning (IL): The goal of imitation learning is to learn a behavior policy π^b given access to either the expert policy π^e or trajectories derived from the expert policy $\mathcal{T}^e$. While there is a multitude of settings with differing levels of access to the expert, this work operates in the setting where the agent only has access to observation-based trajectories, i.e. $\mathcal{T}^e \equiv \{(o_t, a_t)_{t=0}^T\}_{n=0}^N$. Here N and T denotes the number of trajectory rollouts and episode timesteps respectively. We choose this specific setting since obtaining observations and actions from expert or near-expert demonstrators is feasible in real-world settings and falls in line with recent work in this area [13, 12, 11].

c) Optimal Transport for Imitation Learning (OT): In Sec. III-D and Sec. III-E, we used optimal transport (OT) based inverse reinforcement learning (IRL) to learn robot policies in the real world. This is in line with an array of OT-based approaches that have recently been proposed for policy learning [13, 14, 43]. Intuitively, the closeness between expert trajectories $\mathcal{T}^e$ and behavior trajectories $\mathcal{T}^b$ can be computed by measuring the optimal transport of probability mass from $\mathcal{T}^b \rightarrow \mathcal{T}^e$. During policy learning, the policy π_ϕ encompasses a feature preprocessor f_ϕ which transforms observations into informative state representations. Some examples of a preprocessor function f_ϕ are an identity function, a mean-variance scaling function and a parametric neural network. In this work, we use a parametric neural network as f_ϕ. Given a cost function $c : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}$ defined in the preprocessor's output space and an OT objective g, the optimal alignment between an expert trajectory $\mathbf{o}^e$ and a behavior trajectory $\mathbf{o}^b$ can be computed as

$$\mu^* \in \arg\min_{\mu \in \mathcal{M}} g(\mu, f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e), c) \tag{3}$$

where $\mathcal{M} = \{\mu \in \mathbb{R}^{T \times T} : \mu \mathbf{1} = \mu^T \mathbf{1} = \frac{1}{T}\mathbf{1}\}$ is the set of coupling matrices and the cost c can be the Euclidean or Cosine distance. In this work, inspired by [43], we use the entropic Wasserstein distance with cosine cost as our OT metric, which is given by the equation

$$\begin{aligned} g(\mu, f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e), c) &= \mathcal{W}^2(f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e)) \\ &= \sum_{t,t'=1}^{T} C_{t,t'} \mu_{t,t'} \end{aligned} \tag{4}$$

where the cost matrix $C_{t,t'} = c(f_\phi(\mathbf{o}^b), f_\phi(\mathbf{o}^e))$. Using Eq. 4 and the optimal alignment μ^* obtained by optimizing Eq. 3, a reward signal can be computed for each observation using the equation

$$r^{OT}(o_t^b) = -\sum_{t'=1}^{T} C_{t,t'} \mu^*_{t,t'} \tag{5}$$

Intuitively, maximizing this reward encourages the imitating agent to produce trajectories that closely match demonstrated trajectories. Since solving Eq. 3 is computationally expensive, approximate solutions such as the Sinkhorn algorithm are used instead.

Algorithm 1 Demonstration-guided RL

Require:
Expert Demonstrations $\mathcal{D}^e \equiv \{(\mathbf{o}_t, \mathbf{a}_t, \mathbf{g}_t)_{t=0}^T\}_{n=0}^N$
Pretrained policy π^{BC}
Replay buffer $\mathcal{D}$, Training steps T, Episode Length L
Task environment env
Parametric networks for RL backbone (e.g., the encoder, policy, and critic function for DrQ-v2)

Algorithm:
$\pi \leftarrow \pi^{BC}$ ▷ Initialize with pretrained policy **for** *each timestep* t = $1...T$ **do**
  end
  done
Add episode to $\mathcal{D}$
$\mathbf{o}_t = env.reset(),$ done = False, episode = [ ]

$\mathbf{a}_t \sim \pi(\mathbf{o}_t, \mathbf{g}_t)$
$\mathbf{o}_{t+1}, r_t,$ done $= env.step(\mathbf{a}_t)$
episode.append($[\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}, r_t, \mathbf{g}_t]$)
Update backbone-specific networks and
reward-specific networks (for inverse RL) using $\mathcal{D}$

Algorithm 2 Behavior distillation for multi-task policy learning

Require:
Replay Buffers $\mathcal{D}_k^\beta \equiv \{(\mathbf{o}_t, \mathbf{g}_t)_{t=0}^T\}_{n=0}^N \forall k \in \{1, \cdots, K\}$
Trained task-specific policy $\pi^k \forall k \in \{1, \cdots, K\}$
Parametric networks for behavior distillation (i.e., the encoder and the policy)

Algorithm:
$\Pi \leftarrow$ randomly initialized **for** *each gradient step* t = $1...T$ **do**
  end
  $k \sim \{1, \cdots, K\}$ ▷ Sample task
$\mathbf{o}_k, \mathbf{g}_k \sim \mathcal{D}_k^\beta$
$\mathbf{a}_k = \pi^k(\mathbf{o}_k, \mathbf{g}_k)$
$\hat{\mathbf{a}}_k = \Pi(\mathbf{o}_k, \mathbf{g}_k)$
$\mathcal{L} = \|\mathbf{a}_k - \hat{\mathbf{a}}_k\|^2$ ▷ Compute distillation loss (Eq. 2)
Update Π using loss $\mathcal{L}$

VII. Algorithmic Details

PolyTask utilizes efficient demonstration-guided RL with multi-task distillation to enable both multi-task and lifelong learning. Algorithm 1 and Algorithm 2 describe the algorithm for demonstration-guided RL and behavior distillation for multi-task policy learning. Further implementation details are as follows:

a) Task expert training procedure: Our model consists of 3 primary neural networks - the encoder, the actor, and the critic. During the BC pretraining phase in Sec. II-B, the encoder and the actor are trained using a mean squared error (MSE) on the

expert demonstrations. Next, for finetuning, weights of the pretrained encoder and actor are loaded from memory and the critic is initialized randomly. We observed that the performance of the algorithm is not very sensitive to the value of α (in Eq. 1) and we set it to 0.3 for all experiments in this paper. Also, the weight λ controlling the relative contribution of the two losses in Eq. 1 is set to 0.25 for all the experiments in the paper.

b) Behavior distillation training procedure: Following the notation used in Sec. II-C, the expert policies $\pi_{1:k}$ are unified into a multitask policy Π using Eq. 2. For lifelong learning, each unified policy $\Pi_i \forall i \in \{2, \cdots, N\}$ is obtained by distilling $\pi_{1:i}$ into a single policy using Eq. 2.

c) Actor-critic based reward maximization: We use a recent n-step DDPG proposed by **(author?)** [29] as our RL backbone. The deterministic actor is trained using deterministic policy gradients (DPG) given by Eq. 6.

$$\begin{aligned}\nabla_\phi J &\approx \mathbb{E}_{s_t \sim \rho_\beta}\left[\nabla_\phi\ Q_\theta(s,a)|_{s=s_t, a=\pi_\phi(s_t)}\right] \\ &= \mathbb{E}_{s_t \sim \rho_\beta}\left[\nabla_a\ Q_\theta(s,a)|_{s=s_t, a=\pi_\phi(s_t)}\ \nabla_\phi\ \pi_\phi(s)|_{s=s_t}\right]\end{aligned} \tag{6}$$

The critic is trained using clipped double Q-learning similar to **(author?)** [29] in order to reduce the overestimation bias in the target value. This is done using two Q-functions, $Q_{\theta 1}$ and $Q_{\theta 2}$. The critic loss for each critic is given by the equation

$$\mathcal{L}_{\theta_k} = \mathbb{E}_{(s,a)\sim D_\beta}\left[(Q_{\theta_k}(s,a) - y)^2\right] \forall\ k \in \{1,2\} \tag{7}$$

where $\mathcal{D}_\beta$ is the replay buffer for online rollouts and y is the target value for n-step DDPG given by

$$y = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \min_{k=1,2} Q_{\bar{\theta}_k}(s_{t+n}, a_{t+n}) \tag{8}$$

Here, γ is the discount factor, r is the reward, and $\bar{\theta}_1$, $\bar{\theta}_2$ are the slow moving weights of target Q-networks.

d) Target feature processor to stabilize OT rewards: In Sec. III-E, we show that in the absence of environment rewards, we can use optimal transport (OT) based trajectory matching rewards for optimizing the task-specific expert during the finetuning phase of Sec. II-B. The OT rewards are computed on the output of the feature processor f_ϕ which is initialized with a parametric neural network. Hence, as the weights of f_ϕ change during training, the rewards become non-stationary resulting in unstable training. In order to increase the stability of training, the OT rewards are computed using a target feature processor $f_{\phi'}$ [43] which is updated with the weights of f_ϕ every T_{update} environment steps.

A. Hyperparameters

The list of hyperparameters for demonstration-guided RL is provided in Table IV and for behavior distillation and lifelong learning have been provided in Table V.

VIII. Environments

Table VI lists the different tasks that we experiment with from the DeepMind Control suite [37, 38], the Meta-World suite [41] and the Franka kitchen environment [42] along with the number of training steps and the number of demonstrations used. The episode length for all tasks in DeepMind Control is 1000 steps, for Meta-world is 125 steps (except bin picking which runs for 175 steps), and for Franka kitchen is 280 steps. A visualization of all the tasks in DeepMind Control, Meta-World, and Franka kitchen has been provided in Fig. 6, Fig. 7 and Fig. 8 respectively.

IX. Demonstrations

The details about collecting demonstrations on the 3 simulated environment suites have been mentioned in Sec. III-A. Here we provide some additional details about splitting the 566 demonstrations for the Franka kitchen environment provided in the relay policy learning dataset [42] into task-specific snippets. Each demonstration carries out 4 tasks in the environment. We roll out each demonstration in the environment to figure out the time step at which a task is completed and record the task that was completed. After doing this for all demonstrations, we divide each trajectory into task-specific snippets by considering the frames from after the previous task was completed till the current one is completed. Each such snippet is appended to the task-specific demonstration set. We reiterate that since this is play data, the task-specific snippets are suboptimal in the sense that each snippet may start from where the previous task was completed. These positions might be very different from where the Franka robot arm is initialized in the environment when we want to perform a specific task.

For the real robot experiments, up to 2 demonstrations are collected per task by a human operator using a joystick.

X. Robot tasks

We evaluate PolyTask on 10 real-world tasks by adding 4 variations of tasks in addition to the 6 real-world tasks depicted in Fig. 1. In this section, we describe the suite of manipulation experiments carried out on a real robot in this paper. A visualization of a trajectory rollout of each task has been provided in Fig. 9.

(a) **Insert peg in green cup:** The robot arm is supposed to insert a peg, hanging by a string, into a green cup placed on the table.
(b) **Insert peg in yellow cup:** The robot arm is supposed to insert a peg, hanging by a string, into a yellow cup placed on the table.
(c) **Insert peg in blue cup:** The robot arm is supposed to insert a peg, hanging by a string, into a blue cup placed on the table.
(d) **Reach:** The robot arm is required to reach a specific goal after being initialized at a random position.
(e) **Pouring grains (position 1):** Given a cup with some item placed inside (in our case, grains), the robot arm is supposed to move towards a cup placed on the table and pour the item into the cup.
(f) **Pouring grains (position 2):** Given a cup with some item placed inside (in our case, grains), the robot arm is supposed to move towards a cup placed on the table and pour the item into the cup.
(g) **Open a box (left):** In this task, the robot arm is supposed to open the lid of a box placed on the left of the table by lifting a handle provided in the front of the box.
(h) **Open a box (right):** In this task, the robot arm is supposed to open the lid of a box placed on the right of the table by lifting a handle provided in the front of the box.
(i) **Drawer Close (left):** Here, the robot arm is supposed to close an open drawer placed towards the left of the table by pushing it to the target.
(j) **Drawer Close (right):** Here, the robot arm is supposed to close an open drawer placed towards the right of the table by pushing it to the target.

a) Evaluation procedure: For each task, we obtained a set of 10 random initializations and evaluate 4 methods (shown in Table II, namely single task BC, single task expert, and PolyTask in a multi-task and lifelong setting) over 10 trajectories from the same set of initializations. These initializations are different for each task based on the limits of the observation space for the task.

XI. Baseline methods

Throughout the paper, we compare PolyTask with several prominent imitation learning and reinforcement learning methods. Here,

TABLE IV: List of hyperparameters for demonstration-guided RL.

Parameter	Value
Replay buffer size	150000
Learning rate	$1e^{-4}$
Discount γ	0.99
n-step returns	3 (for Meta-World, DM control), 1 (for Franka kitchen, real robot)
Action repeat	2
Seed frames	12000
Mini-batch size	256
Agent update frequency	2
Critic soft-update rate	0.01
Hidden dim	1024
Optimizer	Adam
Exploration steps	2000
DDPG exploration schedule	0.1
Fixed weight α	0.3
Regularization weight λ	0.25
Target feature processor update frequency(steps) for OT	20000
Reward scale factor for OT	10

TABLE V: List of hyperparameters for multi-task learning and lifelong learning using Behavior Distillation.

Parameter	Value
Replay buffer size for each task	30000
Learning rate	$5e^{-5}$
Action repeat	2
Mini-batch size	256
Hidden dim	1024
Optimizer	Adam

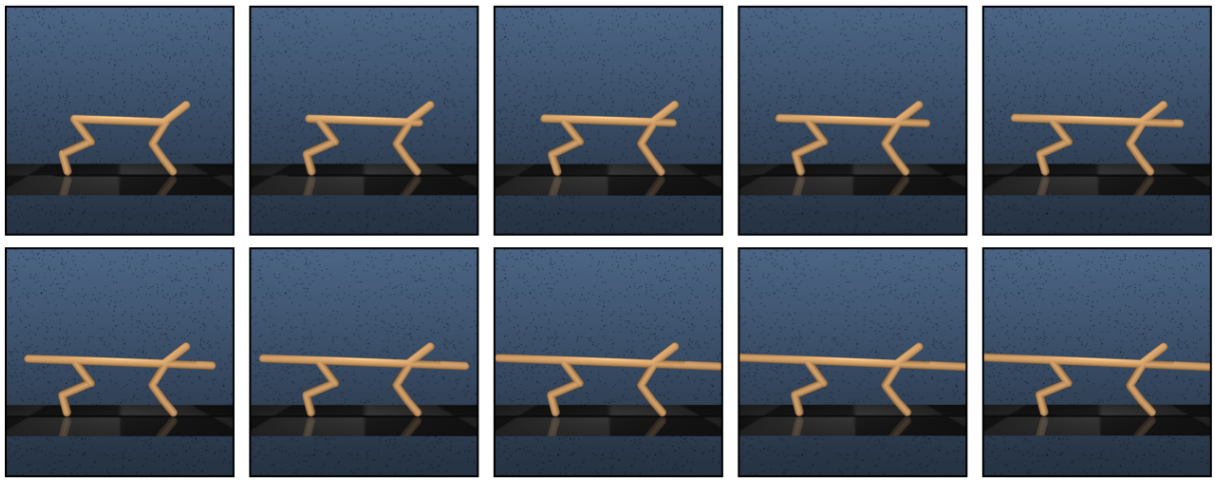


Fig. 6: A visualization of variations of the tasks in the cheetah run environment in the DeepMind Control suite. The torso length of the cheetah is varied across tasks.

we give a brief description of each of the baseline models that have been used.

TABLE VI: List of tasks used for evaluation.

Suite	Tasks	Allowed Steps	# Demonstrations
DeepMind Control	Cheetah Run	5×10^5	10
Meta-World	Door Unlock	5×10^4	1
	Door Open	1×10^5	
	Plate Slide	5×10^4	
	Plate Slide Back	5×10^4	
	Plate Slide Side	5×10^4	
	Plate Slide Side Back	5×10^4	
	Coffee Button	5×10^4	
	Coffee Pull	1.5×10^5	
	Button Press	5×10^4	
	Button Press Wall	5×10^4	
	Button Press Topdown	1×10^5	
	Button Press Wall Topdown	1.5×10^5	
	Bin Picking	1.5×10^5	
	Hammer	1×10^5	
	Drawer Close	5×10^4	
	Drawer Open	5×10^4	
Franka Kitchen	Bottom Burner	4×10^5	100
	Top Burner		
	Light Switch		
	Slide Cabinet		
	Microwave		
	Kettle		
xArm Robot	Put Peg in Green Cup	6×10^3	2
	Open a Box		1
	Pouring Grains		1
	Put Peg in Yellow Cup		2
	Reach		1
	Drawer Close		1

(a) **Task-specific expert:** In DeepMind Control, 10 expert demonstrations per task-variant are collected by training expert policies using DrQ-v2 (as mentioned in Sec. 3.1). These demonstrations are used to learn task-specific expert policies using demonstration-guided RL. Similarly, the demonstrations collected using the scripted policies for Meta-World are then used to learn task-specific expert policies using demonstration-guided RL. For Franka Kitchen, as mentioned in Section 3.1, we use an offline dataset comprising 566 demonstrations. However, since these demonstrations comprise randomly ordered play data, we segment each trajectory into task-specific snippets and use 100 demonstrations per task to learn task-specific experts using demonstration-guided RL. These task-specific expert policies are the highest-performing policies that we are able to learn from the collected demonstrations and online demonstration-guided RL. The specialist policies are unified into a generalist agent through knowledge distillation to obtain the PolyTask policy.

(b) **Single-task behavior cloning (BC):** A supervised learning framework where a single-task policy $\pi(\cdot|o)$ is learned given a dataset of (observation, action) tuples (o, a).

(c) **Goal conditioned BC (GCBC)** [27, 28]: A supervised learning framework where a goal conditioned multi-task policy $\pi(\cdot|o, g)$ is learned given a dataset of (observation, action, goal) tuples (o, a, g).

(d) **Multi-task RL (MTRL)** [20, 19, 17, 44]: A framework where a policy $\pi(\cdot|o, g)$ is jointly learned on all tasks or environments using reinforcement learning. In MTRL, the agent is assumed to have access to all tasks, or environments simultaneously. Here, we consider a DrQ-v2 policy [29] and jointly optimize it across multiple tasks by randomly sampling a task for each environment rollout and learning a goal-conditioned policy on randomly sample training samples across all tasks from a replay buffer.

(e) **Distral** [19]: A MTRL algorithm that jointly trains separate task-specific policies and a distilled policy that captures common behavior across tasks.

(f) **MTRL-PCGrad** [18]: A variant of MTRL with projecting-conflicting-gradients (PCGrad) style gradient optimization to mitigate task interference during gradient updates. PCGrad requires computing the loss on all tasks before each gradient update which increases the time per gradient update. This is why we could not get the result for MTRL-PCGrad on the Franka kitchen environment in Table I.

(g) **MTRL-Demo**: A demo-guided variant of MTRL where we adapt the strategy proposed in Sec. II-B to a multi-task setting. A

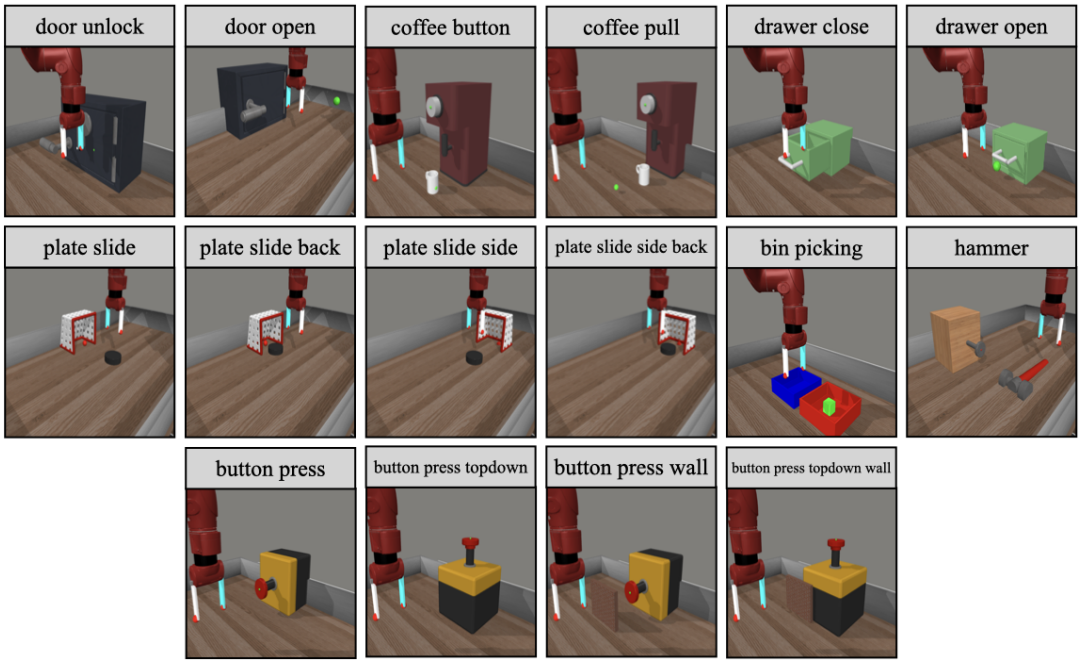


Fig. 7: A visualization of the 16 different tasks from the Meta-World suite used for our experiments.

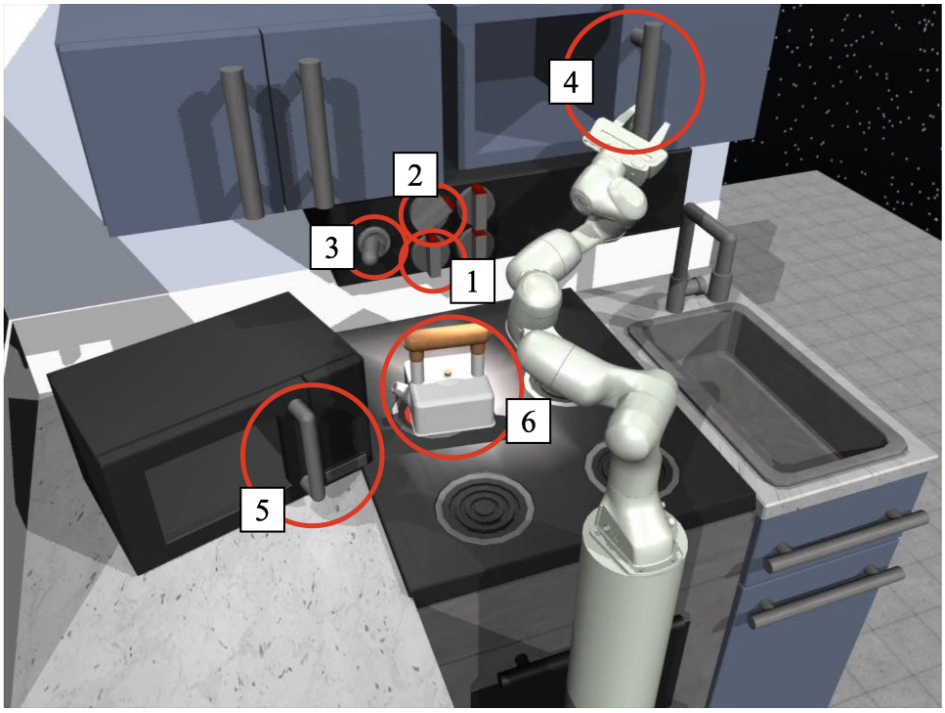


Fig. 8: A visualization of the Franka Kitchen environment. We experiment with 6 tasks in this environment - (1) activate bottom burner, (2) activate top burner, (3) turn on the light switch, (4) open the slide cabinet, (5) open the microwave door, and (6) move the kettle to the top left burner.

goal-conditioned multi-task BC policy is first trained jointly on all task demonstrations and this policy is used for initialization and regularization during the MTRL training. Though MTRL performs poorly on the harder tasks in the Meta-World suite and Franka kitchen environment in Table I, we observe that MTRL-demo does significantly better thus highlighting the importance of demonstration-based guidance for policy learning using RL.

(h) **Fine-tuning** [23]: A framework where a single policy is initialized and keeps getting finetuned on each new task. The parameter count of the policy remains constant throughout training. Even if such a policy does well on the most recent task, it tends to forget previous tasks when being trained on future ones.

(i) **Progressive Nets** [25]: A framework that deals with catas-

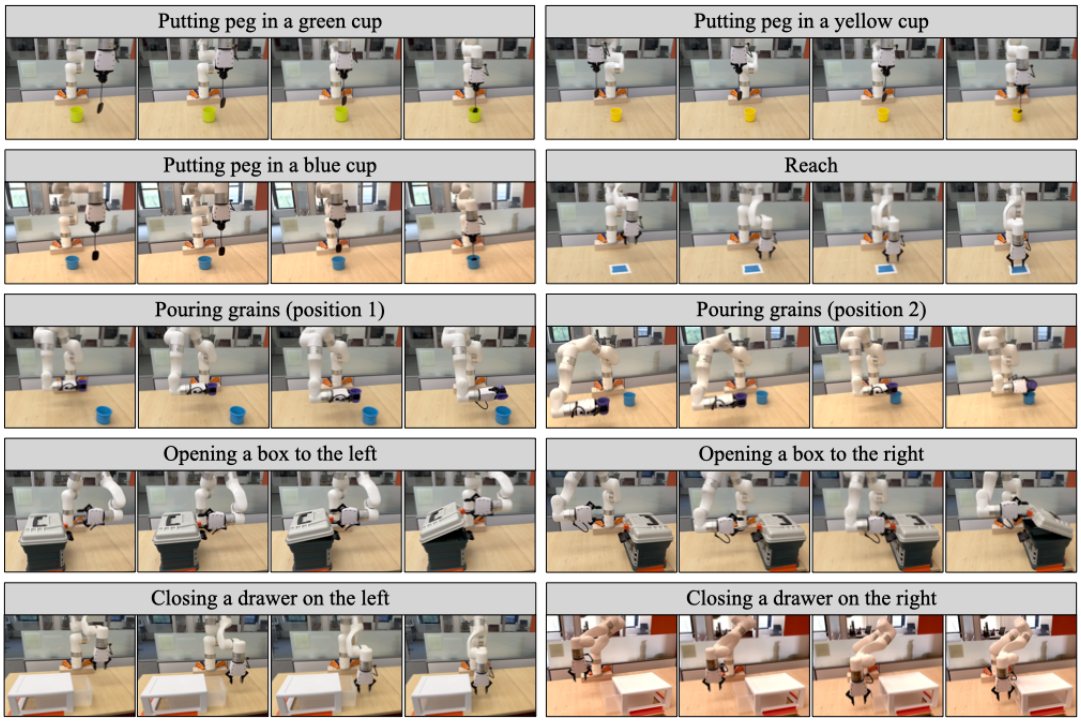


Fig. 9: A visualization of trajectory rollouts of the 10 real-world tasks used for evaluations.

trophic forgetting by adding a new model for each task with lateral connections to all previous models to allow for forward transfer. The parameter count of the policy increases with each new task. In this work, we use progressive networks in an RL setting as done in prior work [25, 45].

XII. Behavior Distillation

In Sec. II-C, we introduce behavior distillation, a technique used for combining multiple expert policies into a single unified policy with the use of knowledge distillation. During the process, we randomly initialize a policy Π having the same architecture as the expert policies. We randomly sample observations o and corresponding goals g from the replay buffer for each expert policy π_k, compute the corresponding expert actions $\pi_k(o, g)$ and use $(s, \pi_k(o, g))$ to train Π using the mean squared error (MSE) loss shown in Eq. 2. It must be noted that though we initialize the distillation policy Π to have the same architecture as the task expert, our framework is compatible with having a different architecture and model size for the distilled policy and the expert policies. The hyperparameters used for multi-task policy learning using behavior distillation have been provided in Table V.

XIII. Lifelong Learning

In this work, we claim that the offline training procedure of behavior distillation makes it suitable for lifelong learning. Fig. 10 and Fig. 11 show a comparison between two of our baselines - finetune and PolyTask - in a lifelong learning scenario on the Meta-World suite and the Franka kitchen environment respectively. We observe that PolyTask exhibits a significantly better ability to tackle catastrophic forgetting than just finetuning the policy. The hyperparameters used for lifelong learning using behavior distillation have been provided in Table V.

XIV. Additional Analysis and Ablations

A. Multi-task distillation on real-robot tasks

In addition to results shown in Sec. III-D, Table VII depicts the performance of PolyTask for multi-task policy learning on the suite of 10 real-world tasks. We observe that the multi-task policy learned using PolyTask is able to effectively succeed on 9.1 out of 10 tasks and outperforms the single-task experts which effectively succeed on 8.9 out of 10 tasks. This highlights the potential advantage of training a single policy for multiple tasks.

B. How important is adding BC regularization?

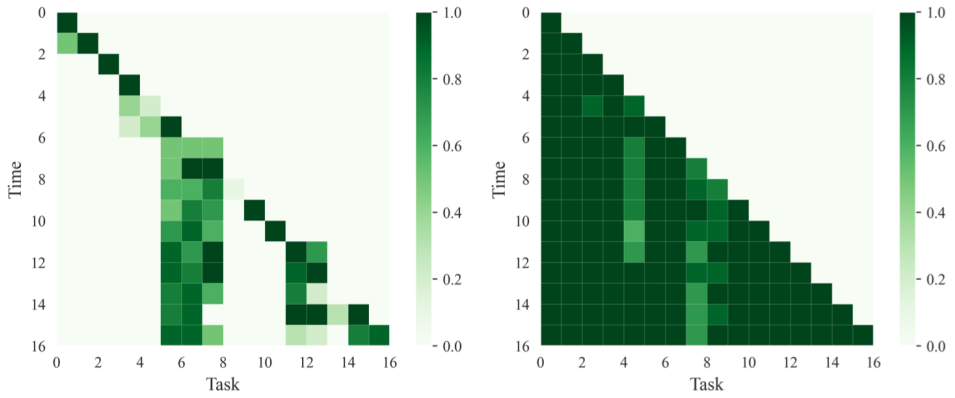
In Sec. III-C, we add a behavior regularization term to the learning objective while training finetuning and progressive networks. Behavior regularization is aimed at accelerating the learning of the current task. Fig. 12 justifies this choice by highlighting the importance of behavior regularization in learning skills sequentially using a finetuning network. We observe that without behavior regularization, most of the skills are not learned within the given interaction budget. However, in the absence of behavior regularization, once a skill is learned, the policy tends to remember it. This can be attributed to the fact that the policy is always trained on replay buffer data from all previous tasks without any interference from the regularization loss. Thus, to summarize, behavior regularization can enhance the sample efficiency of RL at the cost of forgetting prior experiences.

C. Variants of PolyTask for lifelong learning

In this section, we study two variants of distillation strategies for PolyTask. We study two variants of the distillation procedure - *online* and *offline*. A description of the variants has been provided below:

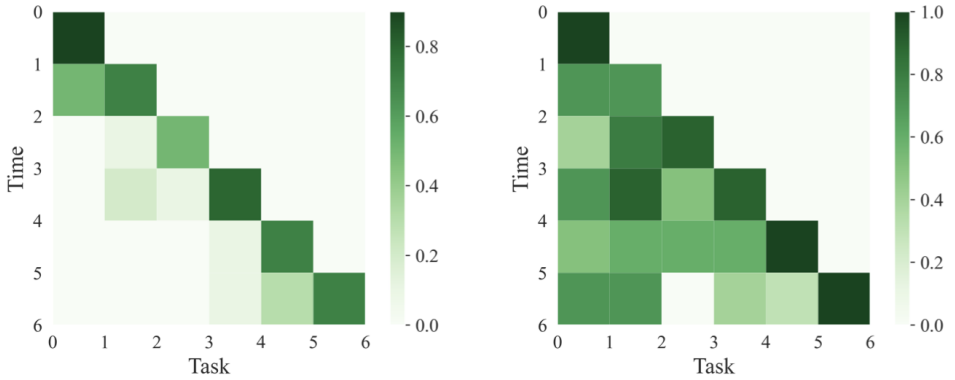
a) Online: In this variant, the policy is trained in a similar way as fine-tuning. However, at the end of each task, the actions in the replay buffer corresponding to the task are relabeled using the current lifelong policy. Then while training on a new task, a distillation loss is used for all previous tasks while an RL is used loss for the new task. The performance of the described method has been shown in Fig. 13(b).

b) Offline: In this variant, a single-task expert policy is obtained for each task using demonstration-guided RL (Eq. 1). At the end of each task, the actions in the replay buffer corresponding to the task are relabeled using the task-specific expert policy. Then



(a) Finetune (b) PolyTask

Fig. 10: A comparison between the performance of Fine-tuning and PolyTask on the Meta-World suite in a lifelong learning setting. We observe that PolyTask exhibits a significantly better ability to tackle catastrophic forgetting.



(a) Finetune (b) PolyTask

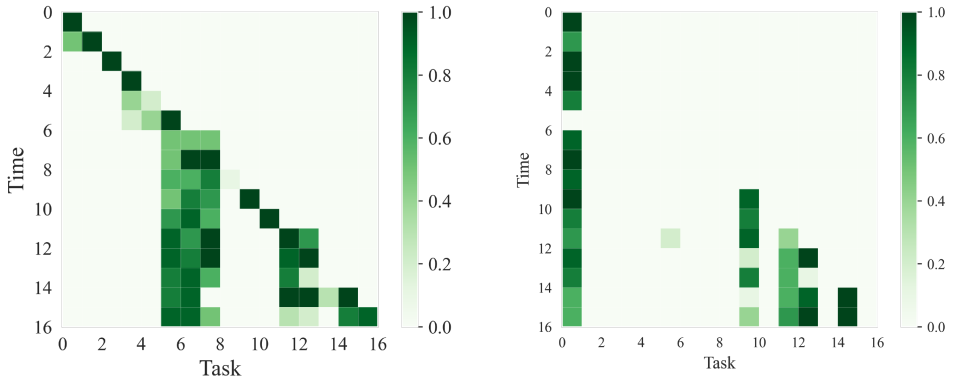
Fig. 11: A comparison between the performance of Fine-tuning and PolyTask on the Franka kitchen environment in a lifelong learning setting. We observe that PolyTask exhibits a significantly better ability to tackle catastrophic forgetting.

TABLE VII: PolyTask results for multi-task learning on 10 real-world tasks.

Task	Single-task BC	Single-task expert	Polytask
Insert Peg in Green Cup	4/10	5/10	7/10
Insert Peg in Yellow Cup	5/10	9/10	10/10
Insert Peg in Blue Cup	4/10	10/10	10/10
Reach	3/10	10/10	10/10
Pouring grains (position 1)	3/10	6/10	6/10
Pouring grains (position 2)	3/10	9/10	9/10
Open box (left)	0/10	10/10	10/10
Open box (right)	1/10	10/10	9/10
Drawer Close (left)	5/10	10/10	10/10
Drawer Close (right)	4/10	10/10	10/10

the lifelong learning policy Π_N is learned by distilling the task experts (using the task-specific state and expert action tuples) into a single policy. The performance of the described method has been shown in Fig. 13(c).

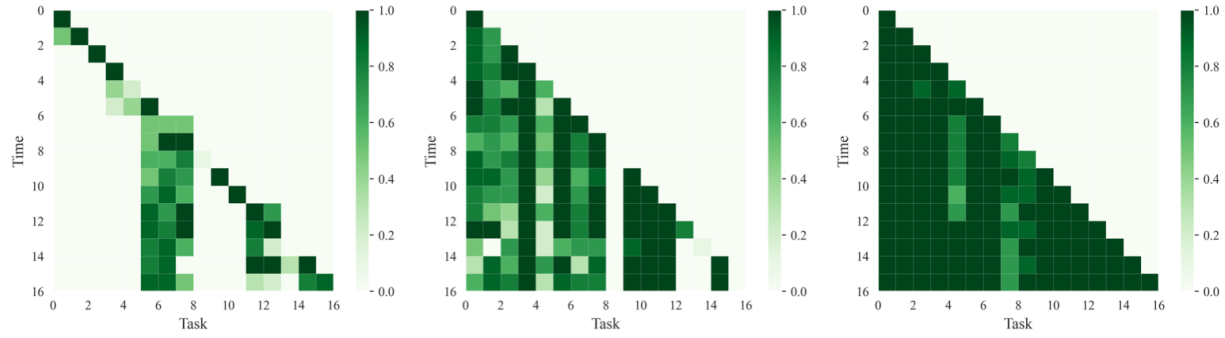
From the results provided in Fig. 13, we conclude that both variants of PolyTask significantly reduce catastrophic forgetting lifelong learning as compared to finetuning. Further, the offline variant of PolyTask exhibits better performance than its online counterpart.



(a) Finetune with behavior regularization

(b) Finetune without behavior regularization

Fig. 12: An illustration of the importance of behavior regularization for training a multi-task policy on the Meta-World suite when the tasks are presented sequentially. We observe that without behavior regularization, most of the skills are not learned within the given interaction budget.



(a) Finetune

(b) PolyTask with stored states and expert actions (online)

(c) PolyTask with stored states and expert actions (offline)

Fig. 13: An illustration of the performance of PolyTask in a lifelong learning scenario on the Meta-World suite when utilizing relabeled expert actions in the task-specific replay buffers. We observe that though there's an extra cost of iterating through the entire replay buffer and storing expert actions, the described variants can significantly reduce catastrophic forgetting.