“Good Robot! Now Watch This!”: Building Generalizable Embeddings via Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Modern Reinforcement Learning (RL) algorithms are not sample-efficient to train on multi-step tasks in complex domains, impeding their wider deployment in the real world. We address this problem by leveraging the insight that RL models trained to complete one set of tasks can be re-purposed to complete related tasks when given just a handful of demonstrations. Based upon this insight, we propose See-SPOT-Run (SSR), a new computational approach to robot learning that enables a robot to complete a variety of real robot tasks in novel problem domains *without* task-specific training. SSR uses pretrained RL models to create vectors that represent model, task, and action relevance in demonstration and test scenes. SSR then compares these vectors via our Cycle Consistency Distance (CCD) metric to determine the next action to take. SSR completes 58% more task steps and 20% more trials than a baseline few-shot learning method that requires task-specific training. SSR also achieves a *four order of magnitude improvement in compute efficiency* and a *20% to three order of magnitude improvement in sample efficiency* compared to the baseline and to training RL models from scratch. To our knowledge, we are the first to address multi-step tasks from demonstration on a real robot without task-specific training, where both the visual input and action space output are high dimensional. Code will be made available.

1 Introduction

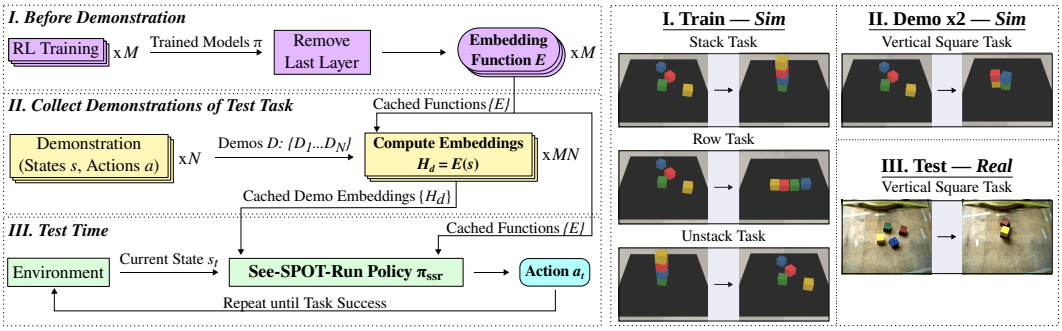

Figure 1: **Left:** Reinforcement Learning Before Demonstration (RLBD) paradigm. **Right:** One of four folds for leave-one-out cross-validation with sim to real transfer and a novel test task.

The ability to quickly adapt deep Reinforcement Learning (RL) models to new tasks with just a handful of demonstrations would dramatically enhance their practicality and applicability. Repurposing previously-trained models for new tasks consumes several orders of magnitude fewer computational resources than training from scratch. For our approach to repurposing RL models, we draw inspiration from social imitation and learning in animals. Certain animals, dogs, for example, have a strong capacity to imitate human performance of a novel task when trained with the “Do as I Do” method [1, 2]. Imagine a dog named Spot that can be been trained with positive reinforcement (“Good Spot!”) to imitate a new action shown to her, such as fetching a specific object from a container, when prompted by a command (“Watch This!”). When Spot imitates, “it is the orientation

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

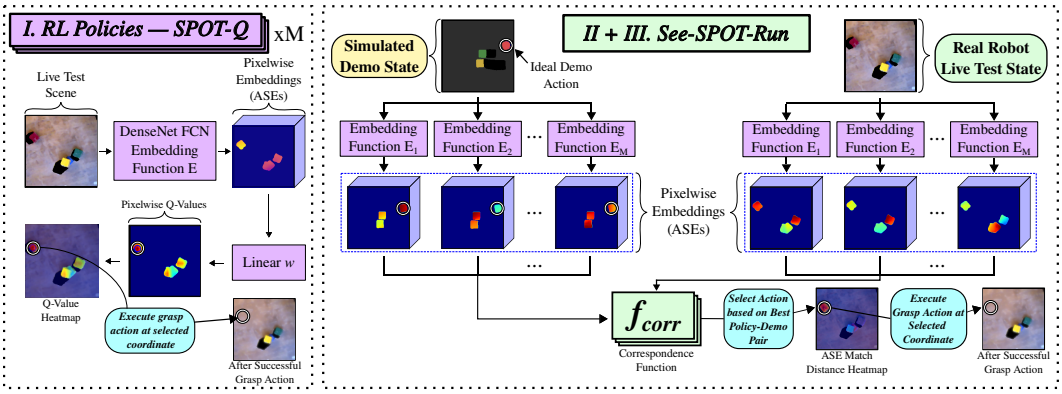

Figure 2: **Left:** An outline of our pretrained RL policies trained with SPOT-Q [4]. Each RL policy contains an Embedding Function E that outputs pixel-wise (per action) embeddings (16x224x224x64 or $H^{\Theta \times X \times Y}$ ASEs). These embeddings are turned into Q-values (16x224x224x1) by a linear layer. The action space A has the same dimension as the Q-values (16x224x224x1, or $\Theta \times X \times Y$) for each `grasp` and `place` action. **Right:** See-SPOT-Run (SSR) Framework architecture executing a row task, the grasp action is selected from the vertical square policy. SSR successfully executes novel tasks using a correspondence function f_{corr} (Eqs. 4-8) that matches the demonstrated action to the live test scene using the embeddings (ASEs) derived from each pre-trained policy.

and the shape of the action [that is] attended to and reproduced" [2, 3]. Spot encodes what she sees in an internal representation, which she may then map onto her trained skillset in order to successfully imitate a new task.

We observe that some deep RL models, like Spot the dog, *already* encode meaningful representations of a scene that can be mapped to a skillset. We hypothesize that we can leverage these encodings to solve new tasks from demonstration *without task-specific learning*, a concept we call **Reinforcement Learning *Before* Demonstration** (RLBD, Fig. 1). To evaluate our hypothesis, we propose the See-SPOT-Run Framework for RLBD.

See-SPOT-Run (SSR, Fig. 2) rapidly adapts an ensemble of existing RL policies to new tasks by imitating demonstrations. Intuitively, existing fully-trained and validated RL models are already invariant to certain changes in the scene within the tested circumstances. We pass a known state and a novel state to a preexisting model to create and then compare latent vectors in a search for embedded task-invariant knowledge to use in novel circumstances. When the known state is from a novel task demonstration, and the novel state is from a live test scene, we can pinpoint latent vectors that map to actions in the test environment that are likely to progress towards completing the novel demonstrated task. Therefore, we apply See-SPOT-Run to imitate demonstrations of a new multi-step task ("Watch This!") by matching demonstrated actions to candidate actions in a test environment. In doing so, we not only enable few-shot imitation of unseen tasks but also provide the potential to vastly expand robotic skill sets at dramatically lower costs.

In our few-shot imitation experiments, we demonstrate See-SPOT-Run's ability to complete a set of multi-step tasks, including block stacking, row making, block unstacking, and building a vertical square structure (see Fig. 1). These tasks are an order of magnitude more challenging than typical reach or limited grasp and place tasks [4] (see Sec 3). To evaluate each task, we use RL policies trained to complete different, and even conflicting (stacking vs unstacking) tasks, and provide two expert demonstrations collected in simulation. In summary, our contributions are as follows:

1. Reinforcement Learning Before Demonstration (RLBD): We propose a concept to use pretrained RL policies to accelerate robotic imitation learning. We show that these policies generate Action-State Embeddings (ASEs) from robot state observations, and that these embeddings can be re-purposed to execute unseen tasks.
2. See-SPOT-Run (SSR): Our RLBD implementation that accomplishes both few-shot imitation and zero-shot sim-to-real domain transfer by matching successful demonstrated actions to the test environment with a novel *Cycle Consistency Distance (CCD)* metric.
3. SSR achieves competitive task performance without the task-specific training required by prior work, leading to a *four order of magnitude improvement in compute efficiency* and a *20% to*

three order of magnitude improvement in sample efficiency for novel tasks when compared to the baseline and to training RL models from scratch.

2 Related Work

Deep learning approaches have successfully completed a range of robotic tasks in visually-guided manipulation [5, 6] and navigation with high sample complexity [7, 8, 9, 10, 11, 12, 13]. VPG [6] achieved particularly efficient RL training for decluttering tasks. "Good Robot!" [4] improved on VPG's RL sample efficiency and showed zero-shot sim-to-real transfer for multi-step block tasks with a risk of *progress reversal* [4], *e.g.* toppling a partially completed tower (see Sec. 3); however, it requires from-scratch task-specific training.

Another approach to tackling deep RL's high sample complexity is through multi-task and meta-learning [14, 15, 16, 17, 18, 19, 20] of general purpose task representations or policies that allow an agent to quickly adapt to novel tasks by fine-tuning previously learned policies. James *et. al.* [16] approach the few-shot learning problem by first meta-learning a task embedding network from a set of training tasks, then computing a test task embedding from a few demonstrations; this task embedding is provided as input to a control policy. Meanwhile, Finn *et. al.* [14] perform one-shot imitation on tasks seen during meta-training but evaluated on unseen objects. While these approaches achieve good performance on one-step tasks, they do not consider multi-step tasks or *progress reversal*. Alternately, few-shot inverse RL methods [21, 22, 23] aim to increase RL sample efficiency by learning a reward function from one or a few demonstrations, then using this reward function to supervise RL training. Yet, Liu *et. al.* [21] and Singh *et. al* [22] do not address learning multi-step tasks, which can require dense reward functions to learn effectively [4, 10]. Goo *et. al.* [23], in turn, propose an IRL approach for multi-step tasks that requires time and compute-intensive training.

Cycle Consistency is a mechanism for comparing representations across domains [24, 25, 26]. Zhang *et. al.*[27] utilizes Cycle Consistency for cross domain dynamics in simulated locomotion tasks and sim-to-real transfer of a basic robot reaching task, showing that Cycle Consistency can mitigate domain shift for the same, single-step, task. Our See-SPOT-Run framework, by contrast, completes both sim-to-real *and* known-to-novel task transfer with more complex multi-step tasks via our own cycle consistency concept.

Behavior Cloning (BC), Imitation Learning (IL) [28, 29, 30, 31, 32, 33, 34], and IL from Observation (ILO) [30], are approaches to mimic the behaviors of agents. Recent reviews cover IL [35, 36] and ILO [37]. Jung *et. al.* [38] takes a hybrid BC and IL approach to grasp and place one block on another block. GAIL [39] learns to imitate with inspiration from generative adversarial networks. Others pursue distribution matching approaches [32, 24, 29]. However, these BC, IL, and ILO papers variously require training on novel tasks, are evaluated only on simulated tasks, or do not consider multi-step tasks with a risk of *progress reversal* [4]. NTP [40] and NTG [41] learn task plans in a blocks environment by imitating videos, but require a perception pipeline to act.

TransporterNets [28] models multi-step IL tasks characterized by *progress reversal* as a sequence of displacements via behavior cloning, with impressive results that scale from 1-1k demonstrations per task. However, TransporterNets makes no attempt at task-to-task transfer for different tasks. Instead, it requires real-world, task-specific demonstrations and training for every unique task, so their approach is simultaneously much easier to solve (direct task transfer) and much more difficult to adapt to new scenarios (due to task-specific training). In comparison, SSR addresses the problem of few-shot imitation with no task-specific training. Even with these limitations, TransporterNets is the closest available method to SSR and thus we chose it as our baseline for comparison.

3 Preliminaries

As in "Good Robot!" [4] and VPG [6], we investigate multi-step tasks by framing our problem as a Markov Decision Process (MDP) (S, A, P, R), with state space S; action space A; transition probability function $P : S \times S \times A \to \mathbb{R}$; and reward function $R : S \times A \to \mathbb{R}$. We make an MDP-simplifying assumption where states equal observations, and that there is a discrete action space. At time step t, the agent observes state s_t and chooses an action a_t according to its policy $\pi : S \to A$, leading to a new state s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$. Q-Learning is an RL method that seeks a Q function that maximizes R over time. Q-Learning policies π select an action a_t at time t by first

estimating the reward R with a function $Q : S \times A \rightarrow R$, then selecting the action that maximizes the expected reward: $a_t = \pi(s_t) = \text{argmax}_{a \in A} Q(s_t, a)$.

Our See-SPOT-Run Framework experiments are pretrained with the Q-Learning-based SPOT-Q framework introduced by "Good Robot!" [4]. SPOT-Q demonstrates the importance of prioritizing *progress reversal* during RL training, and also effectively integrates 'common-sense' exploration constraints into Q-learning. Our RL policies are pretrained with the same exploration constraints and reward functions $\mathbb{R}_{\text{trial}}$, $\mathrm{R}_{\mathcal{P}}$ as the best performing models in [4]. In our development, we make use of a task progress measure, $\mathcal{P} : S \rightarrow \mathbb{Z}^{[0, p_{\max}]}$ as an integer indicating the number of successful task steps; the task is complete when $\mathcal{P}(s_t) = p_{\max}$. Consequently, a *progress reversal* [4] occurs when an action a_t at a timestep t undoes previous task progress *i.e.* there exists an $i \in [0, t]$ such that $\mathcal{P}(s_{t-i}) > \mathcal{P}(s_{t+1})$. Next, we will introduce our proposed methods.

4 Methods

SSR has three phases: **(1) Embed**, **(2) Correspond**, and **(3) Act**.

In phase **(1) Embed**, SSR takes a set of fully-trained policies, such as Q-functions, and converts them into *embedding functions* by extracting the latent per-action feature vectors embedded in the final network layer immediately before Q-values are created (Fig. 2). SSR runs these *embedding functions* on every demonstration and test state to make a unique latent space for each `(policy, demonstration)` pair.

Phase **(2) Correspond** has two steps: (2a) Match and (2b) Verify. In (2a), SSR compares the embedded demonstration state to the embedded test state of each latent space, searching for the test action that most closely matches the successful demonstration action. It finds this closest match by minimizing the euclidean distance between the demonstration action's embedding (feature vector) and each potential test action's embedding. Since matches are made in the embedding space, the matched action should mirror the qualities of the demonstrated action for the new task, and need not bear any resemblance to the action the original preexisting policy would choose for its original task. Intuitively, the high dimensionality of latent vectors ensures dissimilar actions do not collide, in a manner analogous to hash functions. For example, we successfully choose reasonable action matches within policies and across states (Sec. 5), but matches across policies were random and unsuccessful in small tests. Ultimately, this matching process yields a set of candidate actions, one for each `(policy, demonstration)` pair.

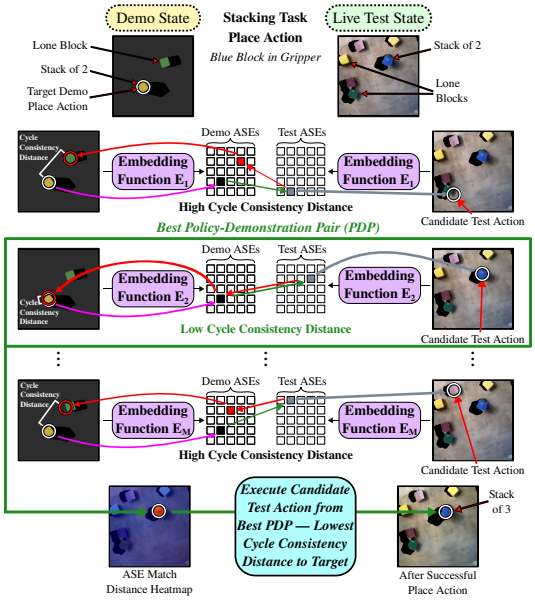

Figure 3: An illustration of cycle consistency correspondence for 3 policy-demonstration pairs (each row is a pair) as See-SPOT-Run (SSR, Fig. 2, Eq. 8) chooses to place a hidden blue block already in the gripper onto the blue block visible in the test state image.

In (2b) Verify, SSR reverses the process, starting with each test candidate action, retrieving its embedding, and performing the match process in the demo scene to see how that action would perform in the demo world. The physical cartesian distance between the known ideal demo action and our new rematch demo action in the action space is our *cycle consistency distance* metric (Fig. 3); intuitively, actions that minimize this distance are more likely to accomplish the demonstrated behavior.

Finally, in phase **(3) Act**, SSR selects the candidate action that minimizes the cycle consistency distance and acts. It repeats this entire process until task completion. SSR thus conducts a comprehensive search over all `(policy, demonstration)` pairs, jointly selecting the latent space and demonstration that are most relevant to the novel task as well as the action that will most likely advance task progress. In Section 4.1, we will describe how we construct the embedding functions, then in Section 4.2, we will elaborate on how SSR uses these embedding functions to imitate demonstrations, and provide additional intuition for our cycle consistency distance metric.

4.1 Generating Embedding Functions

We begin by pretraining task-specific SPOT-Q [4] policies $\pi_m : m = 1...M$ on M known tasks: Recall that Q functions are defined $v_a = Q(s, a)$; we reframe this as $V = Q_m(s)$, where V contains scores for every action $a \in A$ pertaining to task m. Thus, the Q-value of taking action a in state s is $Q_m(s)[a]$. Finally, from each Q_m, we strip off the last linear layer w_m, a linear projection (dense layer) that maps a feature vector to a scalar Q-value, to obtain as many **Embedding Functions**[1] E_m:

$$\pi_m(s) = \underset{a \in A}{\operatorname{argmax}}\, Q_m(s, a) = \underset{a \in A}{\operatorname{argmax}}\, Q_m(s)[a] = \underset{a \in A}{\operatorname{argmax}}\, E_m(s)[a] \cdot w_m \tag{1}$$

Evaluating each E_m on an arbitrary state yields a latent space array H_m, composed of one **Action-State Embedding (ASE)** $h_{m,a} = H_m[a]$ for each action space coordinate $a \in A$, as in Fig. 2.

4.2 Test-Time See-SPOT-Run Policy

In this section, we will define our test-time SSR policy, π_t^{ssr}, which observes the test state s_t and selects an action a_t for the robot to execute, as it attempts to complete the demonstrated test task. It solves the *imitation problem*: approximating an unknown optimal live test task action a_t^* that will complete the next step of a novel, demonstrated task. SSR uses information from N demonstrations $D : \{D_1, ..., D_N\}$. Each demo $D_n : \{(s_{\text{demo}}, a_{\text{demo}})_1, ..., (s_{\text{demo}}, a_{\text{demo}})_T\}$ is a sequence of State-Action pairs which maximize R at each task step $t \in T$.

(1) Embed: We begin by observing the test state s_t and progress $p = \mathcal{P}(s_t)$ and we consider the N state-action pairs $(s_{\text{demo},n}, a_{\text{demo},n})$ with progress p from the N demonstrations[2]. We use these to compute latent space representations of each demonstration state and the test state with each embedding function E_m, yielding $M \times N$ demo latent space arrays $H_{m,n}$, and M test latent space arrays[3] $H_{m,t}$:

$$H_{m,n} = E_m(s_{\text{demo},n}) \tag{2} \qquad H_{m,t} = E_m(s_t) \tag{3}$$

(2) Correspond: We find candidate test actions (2a), then verify these candidate actions with our cycle consistency distance metric (2b). To find a candidate action, we match the demo action to the test state by selecting the minimum euclidean distance between the demo action embedding (ASE) and all ASE vectors in the test latent space. We repeat this comparison for each latent space, yielding $M \times N$ total candidate actions $F_{m,n}$, one for each `(policy, demonstration)` pair. Candidate actions are expressed as coordinates in the action space A, and can be used to index quantities such as the latent spaces H, which have the same dimension as A, as detailed in the Fig. 2 caption.

$$L_{m,n} = \min_{a \in A} \|H_{m,t} - H_{m,n}[a_{\text{demo},n}]\|_2 \tag{4} \qquad F_{m,n} = \underset{a \in A}{\operatorname{argmin}} \|H_{m,t} - H_{m,n}[a_{\text{demo},n}]\|_2 \tag{5}$$

We then verify each matched candidate test action in the demo scene using our *cycle consistency distance* metric. We verify, or match in reverse, the candidate action's ASE $H_{m,t}[F_{m,n}]$ to the corresponding demonstration latent space $H_{m,n}$, minimizing euclidean distance in the latent space as above. We expect the verification action a_{rematch} to be physically close to the original demonstrated action $a_{\text{demo},n}$ in the action space A. Therefore, we define the cycle consistency distance $C_{m,n}$ as the Cartesian distance between $a_{\text{demo},n}$ and a_{rematch}.

$$a_{\text{rematch}} = \underset{a \in A}{\operatorname{argmin}} \|H_{m,t}[F_{m,n}] - H_{m,n}[a]\|_2 \tag{6}$$

$$C_{m,n} = \|a_{\text{demo}} - a_{\text{rematch}}\|_2 \tag{7}$$

The purpose of the cycle consistency distance $C_{m,n}$ is to consider other ASE vectors in the latent space $H_{m,n}$ that might indicate that a candidate action $F_{m,n}$ is not an authentic match. Alternatively, the rematch step can be interpreted as an assessment of how relevant a given latent space is to the test state and task at hand.

[1] In equations, parenthesis ( ) are for function arguments and brackets [ ] index arrays or discrete coordinates.

[2] We note that, because each target demonstration action a_{demo} is optimal, demos satisfy the property that they have monotonically increasing progress. Our method is trivially extended to suboptimal demonstrations by considering all state-action pairs with progress p.

[3] While Section 4.1 refers to latent spaces as H_m, we extend this notation here, using $H_{m,n}$ to designate the state from the n-th demonstration embedded with E_m. Similarly, $H_{m,t}$ is the test state embedded with E_m.

(3) Act: We select and execute the candidate action with minimal Cycle Consistency Distance (Eq 8). We also evaluate a baseline SSR without cycle consistency by skipping the rematch phase and directly minimizing the *L2 Consistency Distance* (Eq. 9) $L_{m,n}$.

$$\hat{m}, \hat{n} = \underset{m,n}{\operatorname{argmin}}\, C_{m,n} \qquad a_t^{\text{ssr}} = F_{\hat{m},\hat{n}} \tag{8}$$

$$\hat{m}, \hat{n} = \underset{m,n}{\operatorname{argmin}}\, L_{m,n} \qquad a_t^{\text{ssr}} = F_{\hat{m},\hat{n}} \tag{9}$$

This completes our definition of the SSR policy $a_t^{\text{ssr}} = \pi_{\text{ssr}}(s_t)$. Every time an action a_t^{ssr} completes we collect a new observation s_{t+1}, running the SSR policy π_{ssr} repeatedly until task progress $\mathcal{P}(s_T)$ reaches its maximum, p_{max}, indicating that the task is complete. We evaluate See-SPOT-Run with L2 consistency distance (**SSR L2CD** in Tables 1, 2, and 3) and cycle consistency distance (**SSR CCD** in Tables 1, 2, and 3), as well as prior work, with few-shot experiments in Sec. 5. We will also discuss how our SSR Framework with CCD surpasses the other methods.

5 Experiments

We outline our assessment metrics in Sec. 5.1, provide simulation results in Sec. 5.2, then cover our sim-to-real transfer results in Sec. 5.3. We pretrain models with SPOT-Q [4] (Sec. 3) with the workspace, commands and action space defined in "Good Robot!" [4]. Implementation and Robot details are in Fig. 1 and Appendix C.1.

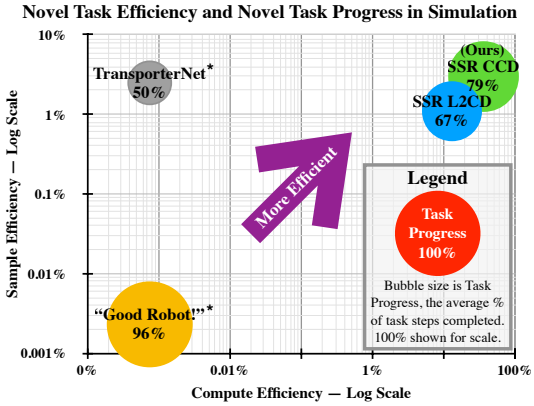

Figure 4: Higher is better on all metrics (Sec 5.1). Log Scale. Larger bubbles are better. See Tab. 1.

5.1 Evaluation Metrics

Our metrics quantify the broad improvements in task performance and reductions in the resources necessary to perform novel tasks. Most important, we consider critical efficiency measures that have not previously been evaluated in the baselines, to motivate the broad range of useful applications for both RBLD and SSR.

Test Metrics evaluate how effectively the robot completes the test tasks, and higher is better: (1) **Trial Success Rate (Trials)** is the percentage of multi-step tasks completed 100% successfully, and in many applications completing a task is a prerequisite to moving on to the next task. (2) **Action Efficiency (Eff.)** is the $\frac{\text{ideal}}{\text{actual}}$ number of actions per trial, and more efficient models will complete tasks with fewer actions. [6] Our ideal is 6 actions for all tasks except for rows, which is 4 actions. [4] (3) **Progress (Prog.)** is each trial's maximum proportional within-task progress averaged over all trials, *e.g.* a 3 of 4 block stack is 75%, to show capabilities that nearly complete trials. (4) **Recoveries (Recov.)** is the percentage of trials *in which there was a mistake such as a progress reversal* that the agent was able to complete with a trial success, *i.e.* $\frac{\text{trial successes containing } \textit{progress reversal}}{\text{trials containing } \textit{progress reversal}}$. Higher recovery rates reflect better robustness to perturbations and uncommon situations.

Cost Metrics delineate resources spent, and lower is better: (5) **Train Steps** is the number of neural network batch steps performed prior to executing on a novel test task. Each individual experiment is run on one NVIDIA GeForce RTX 2080Ti GPU. (6) **Annotated Actions (Ann. Actions)** is the number of robot actions a_t that have been annotated by either a human or scripted observer, which are each fairly high cost activities in robot or human time [42, 43].

Efficiency Metrics evaluate trial success benefits with respect to the cost metrics, and higher is better. (7) **Trial Success Compute Efficiency** $\frac{\text{Trials}}{\text{Train Steps}+1}$ is the amortized percentage of trials that can be completed for every training batch step when completing a novel task. We add one to the train steps denominator to prevent dividing by 0 with SSR. (8) **Trial Success per Annotated Action** $\frac{\text{Trials}}{\text{Ann. Actions}}$ measures the increase in trial success rate amortized over annotated actions on a test task.

5.2 Simulation Experiments

We pretrain RL policies on four tasks as in Fig. 1: stacking, row-making, unstacking, and 2x2 vertical square. At test time we perform four-fold cross-validation where the test task is given two

demonstrations but no pretrained model, then the remaining three models predict actions for the untrained test task. To obtain our TransporterNet [28] baseline on its less challenging scenario (Sec. 2), we collect two demonstrations of each test task in their *ravens* framework, then train TransporterNet on these demonstrations for 40k iterations. We will examine overall performance on the test metrics, task-specific test metrics of particular interest, and then our efficiency metrics.

Simulation Task	**Average Test Metrics**				**Costs**		**Efficiency**	
	Trials	Action Efficiency	Prog.	Recov.	Train Steps	Annotated Actions	Compute	Sample
TransporterNet [28]*	30%	35%	50±3%	8%	40k	12	0.00075%	2.5%
SSR L2CD Eq. 9	13%	30%	67±1%	5%	0	12	13%	1.1%
SSR CCD Eq. 8 (ours)	**36%**	**41%**	**79±1%**	**19%**	**0**	12	**36%**	**3.0%**
"Good Robot!" [4] *	91%*	57%*	96±1%*	90%*	120k	40k	0.00076%	0.0023%

Table 1: Simulation task performance on the metrics detailed in Sec. 5.1, averaged over all four folds of leave-one-model-out cross-validation. "Average Test Metrics" averages Table 2 values. Bold indicates the best performing model. Higher is better for all metrics except costs. The progress range, *e.g.* in 50±3%, the 3 is standard error. * Starred methods address the simpler problems described in Sec. 2, so comparisons should carefully consider this context. TransporterNets [28] trains on robot demos for each novel task with no task-to-task transfer. SPOT-Q [4], the SSR pretraining step, tests on the train task, provides a cost and efficiency baseline plus a test metrics ceiling.

Simulation Task	**Stack**				**Unstack**				**Row**				**Vertical Square**			
	Trials	Action Eff.	Prog.	Recov.	Trials	Action Eff.	Prog.	Recov.	Trials	Action Eff.	Prog.	Recov.	Trials	Action Eff.	Prog.	Recov.
TransporterNet [28]	12%	27%	51%	12%	**86%**	63%	**94%**	–	2%	25%	24%	2%	20%	29%	32%	9%
SSR L2CD Eq. 9	0%	–	51%	0%	38%	60%	76%	–	6%	27%	73%	7%	8%	34%	67%	7%
SSR CCD Eq. 8 (ours)	**24%**	22%	**75%**	16%	66%	77%	82%	–	**30%**	38%	**81%**	28%	**24%**	28%	**79%**	14%

Table 2: Leave-one-model-out cross-validation of our See-SPOT-Run (SSR) framework for 50 simulation trials. Bold indicates key best metrics (Sec. 5.1). Unstacking has no notion of recovery.

Our results on the overall **Test Metrics** in Table 1 show SSR with CCD achieves 36% trial completion, 79% average progress, 41% action efficiency, and 29% recovery rate; L2 Correspondence L2CD gets 13%, 67%, 30%, and 5%, respectively, which demonstrates the benefit of evaluating demo embeddings H_d with Cycle Consistency Correspondence CCD (Eq. 4) ; and TransporterNet gets 30%, 50%, 35%, and 8%, respectively, which highlights our method's overall improvement over prior few-shot imitation work in our more challenging scenario with no task-specific training.

Our task-specific simulation results in Table 2 show SSR with CCD outperforms L2CD in all cases, and better than TransporterNets in all cases except unstacking. The SSR worst case trial success rate of 24% is 12x better than the TransporterNets worst case of 2%. SSR completes 66% of unstacking trials vs 86% for TransporterNets, and SSR completes 30% of rows vs 2% for TransporterNets. This highlights a shortcoming of TransporterNets' feature template displacement matching approach which is not present in SSR. A more detailed per-task simulation breakdown is in Appendix C.3.

Efficiency Metrics: With CCD, **SSR achieves 36% compute efficiency, which is roughly four orders of magnitude better** than TransporterNet and "Good Robot!", which both have 8e-4% compute efficiency. This is triple the 13% compute efficiency of L2CD, further underlining the effectiveness of our CCD metric. **SSR also achieves three orders of magnitude better sample efficiency** with CCD than 'Good Robot!", at 3% vs. 2e-3%, while maintaining a much smaller lead over TransporterNet at 2.5% sample efficiency, and L2CD at 1.1% sample efficiency. The smaller gap is because, as an imitation method, TransporterNets is able to replicate behaviors with a few annotated actions. Even so, SSR maintains its lead over TransporterNets due to its superior trial completion rate. Next we will examine our performance on the real robot experiments.

5.3 Real Robot Experiments

We transfer SSR to a real robot using our models pretrained in simulation from Section 5.2 with results in Tab. 3. All other aspects of the method remain the same except task progress is recorded by a different observer, since in simulation we read internal simulator states. TransporterNets is not designed for sim to real transfer, and is thus not included here.

In real experiments SSR has an average of 40% of trials complete, 42% action efficiency, 84% progress, and 35% rate of recoveries; which is very similar to our results in simulation at to 36%, 41%, 79%, and 29%, respectively. For reference, "Good Robot" completed 100% of trials after 20k RL training actions and 60k training steps for both stacks and rows; but they should not be compared directly to SSR since it trains with the aid of a task specific reward function, it is scored on the train task, and it is neither designed for nor capable of completing new test tasks on its own. We examine a per-task breakdown of real SSR performance metrics below.

Real Stack: The SSR real stacking model completes 30% of trials and makes 80% progress which is slightly better than the 24% of trials and 75% progress of SSR in simulation. For efficiency, the SSR $\frac{\text{Trials}}{\text{Ann. Actions}}$ of 2.5% and $\frac{\text{Trials}}{\text{Train Steps}+1}$ of 30% is a striking 500x and 18,000x better than "Good Robot" real stack trials at 0.005% and 0.0017%, respectively.

Real Task	Trials	Action Efficiency	Prog.	Recov.
Stack	**30%**	25%	**80±5%**	22%
Unstack	**90%**	86%	**97±3%**	–
Row	**30%**	28%	**77±7%**	66%
Square	**10%**	27%	**75±4%**	10%
Average	**40%**	42%	**82±3%**	33%

Table 3: Real See-SPOT-Run framework with Cycle Consistency, SSR CCD Eq. 8, performance on Sim-To-Real transfer to novel tasks from simulated demos during leave-one-model-out cross-validation. Bold entries are for ease of reading and progress range (±) is standard error.

Real Unstack: The real unstacking model is the best performing model by far, completing 90% of trials successfully, with 86% efficiency, a value exceeding the 66% trial success rate of simulated unstacking. The reason for this difference is because the simulated fully actuated gripper can forcefully press blocks off-center in the vertical z direction upon closing, toppling the tower, while the real underactuated gripper tip physically flexes up to several centimeters out of the way, so the stack typically remains standing.

Real Row: The real row making SSR model performs similarly to simulation with an equal 30% trial success; 77% progress, a 4% drop; 28% action efficiency, a 10% drop. For efficiency, as with stacks, the SSR $\frac{\text{Trials}}{\text{Ann. Actions}}$ of 2.5% and $\frac{\text{Trials}}{\text{Train Steps}+1}$ of 30% is a striking 500x and 18,000x better than "Good Robot" real row trials at 0.005% and 0.0017%, respectively.

Real Vertical Square: The vertical square task proved to be the most challenging with just 10% of trials completed. In the real trials it is able to complete the first 3 steps reliably, but struggles with the fourth step to complete the square, which is evident in the average progress of 75%.

6 Conclusion

See-SPOT-Run's three to four order of magnitude sample efficiency and compute efficiency improvements show it can *immediately* and *effectively* begin making progress on a novel task in a novel domain with scarce demonstration examples. By contrast, RL models such as "Good Robot!" typically start out with highly randomized exploration and no ability to complete a novel task, requiring inefficient sampling and compute-intensive training to first match and then beat SSR on a given task. Furthermore, although SSR performs 12x better than TransporterNets [28] in the worst case sim trials, TransporterNets may scale better than SSR on some metrics as demonstrations are added. Even so, it would ultimately be more efficient in the long run to develop methods where demonstrations on one multi-step task also improve performance and efficiency on the next task.

For these reasons, we suggest that future research investigate how RLBD, Embeddings, and See-SPOT-Run might guide the exploration and augment the training of other learning methods such as Reinforcement Learning [4, 6], Behavior Cloning, Imitation Learning, and Meta-Learning. Success in these avenues would substantially improve multi-task learning efficiency. Further improvements to RLBD itself might include better correspondence metrics, automatically assessing task progress, or otherwise mitigating the need for a dense reward [23, 25].

In summary, we have introduced the concept of Reinforcement Learning Before Demonstration, implemented via the See-SPOT-Run framework. Our framework achieves simultaneous few-shot imitation and sim-to-real transfer for multi-step tasks, such as block stacking and row-making, that are prone to *progress reversal*. See-SPOT-Run improves *compute and sample efficiency by three to four orders of magnitude* vs. prior work by leveraging pretrained RL models to solve the *imitation problem* on novel tasks.

References

[1] C. Fugazza and Ádám Miklósi. Should old dog trainers learn new tricks? the efficiency of the do as i do method and shaping/clicker training method to train dogs. *Applied Animal Behaviour Science*, 153:53–61, 2014. ISSN 0168-1591. doi:https://doi.org/10.1016/j.applanim.2014.01.009. URL https://www.sciencedirect.com/science/article/pii/S0168159114000264.

[2] C. Fugazza, E. Petro, Ádám Miklósi, and Ákos Pogány. Social learning of goal-directed actions in dogs (canis familiaris): Imitation or emulation? *Journal of Comparative Psychology*, 133 (2):244–251, 2019.

[3] V. Horner and A. Whiten. Causal knowledge and imitation/emulation switching in chimpanzees (pan troglodytes) and children (homo sapiens). *Animal Cognition*, 8(3):164–181, 2005.

[4] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager. "Good Robot!": Efficient Reinforcement Learning for Multi-Step Visual Tasks with Sim to Real Transfer. *IEEE Robotics and Automation Letters (RA-L)*, 5(4):6724–6731, 2020. URL https://arxiv.org/abs/1909.11730.

[5] O. Kroemer, S. Niekum, and G. D. Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of Machine Learning Research*, 22(30): 1–82, 2021.

[6] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *IROS*, pages 4238–4245. IEEE, 2018.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[8] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, 2018.

[9] J. Mahler and K. Goldberg. Learning deep policies for robot bin picking by simulating robust grasping sequences. *Conference on Robot Learning*, pages 515–524, 2017.

[10] J. F. Hernandez-Garcia and R. S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target. *arXiv preprint arXiv:1901.07510*, 2019.

[11] M. Q. Mohammed, K. L. Chung, and C. S. Chyi. Review of deep reinforcement learning-based object grasping: Techniques, open challenges, and recommendations. *IEEE Access*, 8: 178450–178481, 2020.

[12] C. R. Garrett, C. Paxton, T. Lozano-Perez, L. P. Kaelbling, and D. Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684, 2020.

[13] A. K. Tanwani. Domain invariant representation learning for sim-to-real transfer. In *Conference on Robot Learning (CoRL)*, 2020.

[14] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. 2017.

[15] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML'17 Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pages 1126–1135, 2017.

[16] S. James, M. Bloesch, and A. J. Davison. Task-embedded control networks for few-shot imitation learning. *Conference on Robot Learning*, pages 783–795, 2018.

[17] R. Rahmatizadeh, P. Abolghasemi, L. Boloni, and S. Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3758–3765, 2018.

[18] A. Zhou, E. Jang, D. Kappler, A. Herzog, M. Khansari, P. Wohlhart, Y. Bai, M. Kalakrishnan, S. Levine, and C. Finn. Watch, try, learn: Meta-learning from demonstrations and rewards. In *ICLR 2020 : Eighth International Conference on Learning Representations*, 2020.

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

[20] D. Kalashnkov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv*, 2021.

[21] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125, 2018.

[22] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[23] W. Goo and S. Niekum. One-shot learning of multi-step tasks from observation via activity localization in auxiliary video. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7755–7761, 2019.

[24] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017.

[25] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas. Playing hard exploration games by watching youtube. In *Advances in NeurIPS*, pages 2935–2945, 2018.

[26] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11157–11166, 2020.

[27] Q. Zhang, T. Xiao, A. A. Efros, L. Pinto, and X. Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QIRlze3I6hX.

[28] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

[29] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *International Conference on Learning Representations*, 2018.

[30] Z. Zhu, K. Lin, B. Dai, and J. Zhou. Off-policy imitation learning from observations. In *Advances in Neural Information Processing Systems*, volume 33, pages 12402–12413, 2020.

[31] S. Reddy, A. D. Dragan, and S. Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. In *ICLR 2020 : Eighth International Conference on Learning Representations*, 2020.

[32] I. Kostrikov, O. Nachum, and J. Tompson. Imitation learning via off-policy distribution matching. In *ICLR 2020 : Eighth International Conference on Learning Representations*, 2020.

[33] R. Jena, C. Liu, and K. Sycara. Augmenting gail with bc for sample efficient imitation learning. In *Conference on Robot Learning (CoRL)*, 2021.

[34] A. Z. Ren, S. Veer, and A. Majumdar. Generalization guarantees for imitation learning. *Conference on Robot Learning (CoRL)*, 2020.

[35] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun. Survey of imitation learning for robotic manipulation. In *International Journal of Intelligent Robotics and Applications*, volume 3, pages 362–369, 2019.

[36] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50(2):21, 2017.

[37] F. Torabi, G. Warnell, and P. Stone. Recent advances in imitation learning from observation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 6325–6331, 2019.

[38] E. Jung and I. Kim. Hybrid imitation learning framework for robotic manipulation tasks. *Sensors*, 21(10):3409, 2021.

[39] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, volume 29, pages 4565–4573, 2016.

[40] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3795–3802, 2018. doi:10.1109/ICRA.2018.8460689.

[41] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8557–8566, 2019. doi:10.1109/CVPR.2019.00876.

[42] M. L. Gray and S. Suri. *Ghost Work: How to Stop Silicon Valley from Building a New Global Underclass*. 2019.

[43] K. Hara, A. Adams, K. Milland, S. Savage, C. Callison-Burch, and J. P. Bigham. A data-driven analysis of workers' earnings on amazon mechanical turk. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 449, 2018.