A Figures

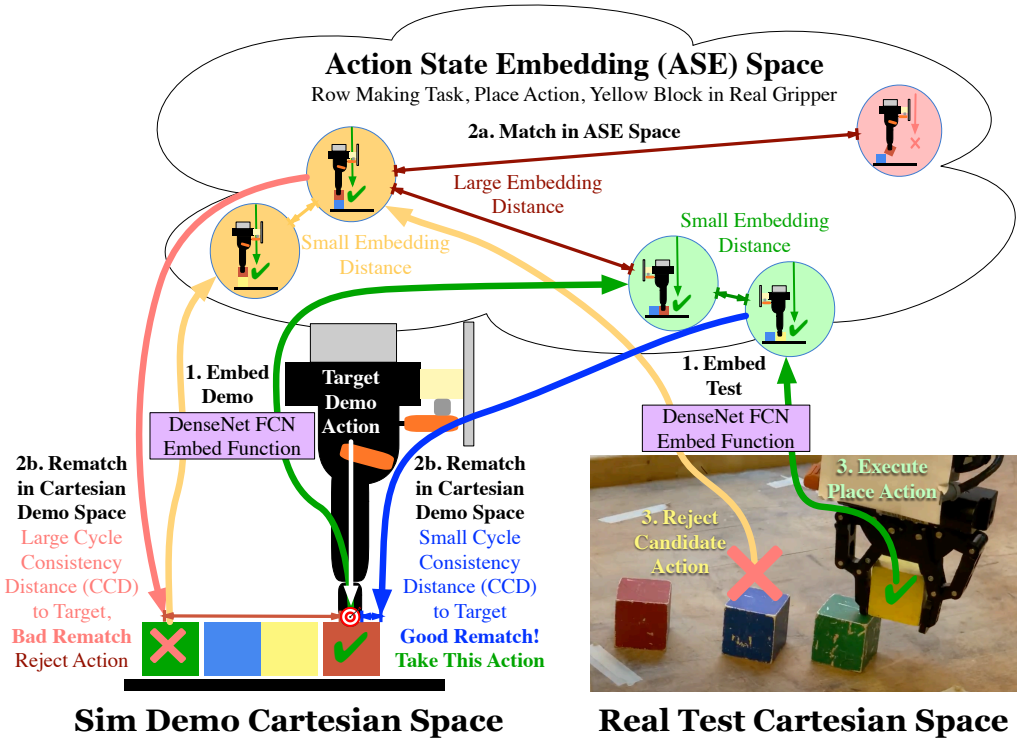

Figure 5: Illustration of the three phases of See-Spot-Run (SSR, Sec. B.3, Fig. 2, Alg. 1) execution, as well as a depiction of the best Policy-Demonstration pair succeeding at placing a real yellow block to create a row. The physical cartesian distance from the original target demo action to the rematch of the test ASE in the demo scene (arrow) is the Cycle Consistency Distance. The closer the rematch is to the target, the better. Typically, the demo and real scenes will have identical task progress measurements, but variable physical positions of objects. The green and blue arrows follow SSR as it considers, selects, and successfully executes placing the yellow block at the end of the row. The yellow and pink arrows show another possible action which is ultimately rejected due to a large Cycle Consistency Distance.

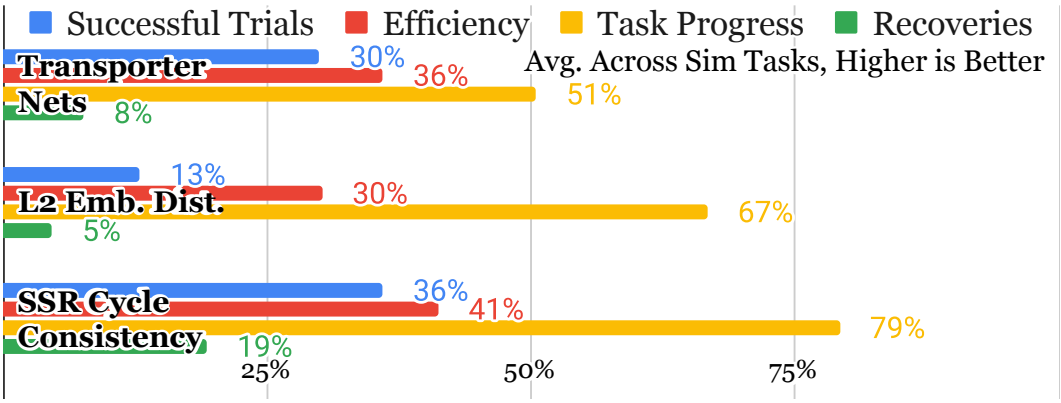

Figure 6: Summary comparison of algorithm performance (Table 2) over all four folds of cross validation of simulation tasks on the evaluation metrics (Sec. C.2). "Efficiency" refers to Action Efficiency.

B Methods — Restated in Imperative Form

In this work, we develop a few-shot Reinforcement Learning Before Demonstration (RLBD) method, detailed in Fig. 1, that leverages a set of SPOT-Q trained RL policies to complete a novel task with no task-specific training and just a few demonstrations. Our RLBD implementation, See-SPOT-Run (SSR, detailed in Fig. 2), identifies the Policy-Demonstration Pair (PDP) that is most relevant to the test task and state and then selects an action using this PDP to advance task progress. We will first summarize RLBD, then elaborate SSR and its key components.

B.1 Reinforcement Learning Before Demonstration (RLBD)

RLBD (Fig. 1) has three steps:

Step **I. Before Demonstration** is to pretrain task-specific RL policies π_m, on M known tasks. We then strip the final linear layer from the model for each policy $\Pi : \{\pi_1 ... \pi_M\}$ to create Embedding Functions $E : \{E_1 ... E_M\}$ (Fig. 2, Sec. B.2).

Step **II. Collect Demonstrations of Test Task** (Fig 1, 2) is to gather and save N varied demonstrations $D : \{D_1, ..., D_N\}$ for the novel task. Multiple demonstrations can show varying approaches to completing a task, *e.g.* vertical squares (Fig. 1) might be constructed row first or stack first.

Step **III. Test Time** is to observe the test state and select an action a_t for the robot to execute, as it attempts to complete the demonstrated test task. A test time policy attempts to solve the *imitation problem*: approximating an unknown optimal live test task action a_t^* which would complete the next step of the novel demonstration task by finding the most relevant action in the test scene.

We will describe the to use of Embedding Functions to create ASEs in Sec. B.2, then our solution to imitation, the See Spot Run Framework (SSR) in Sec. B.3.

B.2 Generating Action-State Embeddings (ASEs)

Each of the M tasks has an RL policy $\pi_m : m = 1...M$; we strip off the last linear layer of each policy to obtain as many **Embedding Functions** $E_m(s_t) \rightarrow H_m$, where the output of each embedding function is a set of **Action-State Embeddings (ASEs)** H_m, composed of one ASE $h_{m,a}$ (a vector) for each action space coordinate $a \in A$. Recall that Q functions are defined $v_a = Q(s_t, a_t)$; we reframe this as $V_A = Q(s_t)$, where V_A contains scores for every action; thus, $\pi_m(s_t) = \text{argmax}_{a \in A}\, Q_m(s_t)[a] = \text{argmax}_{a \in A}\, E_m(s_t)[a] \cdot w_m$, where w_m is a linear projection (dense layer) that maps an ASE to a scalar Q-value $q_{m,a}$ such that $q_{m,a} = h_{m,a} \cdot w_m$.

Our imitation methods also use information from a Demonstration $D_n : ((s_1, a_{d,1}) ... (s_T, a_{d,T}))$, i.e. a sequence of State-Action pairs which maximize R at each time step $t \in T$. We note that, because each Target Demonstration Action a_d is optimal, demonstrations satisfy the property that they have monotonically increasing progress; in what follows we write $D_n(p)$ to denote the state-action pair at progress p. We now define a **Policy Demonstration Pair (PDP)** $PDP_{m,n} = (E_m, D_n)$ where D_n is the n^{th} demonstration. Given M pre-trained tasks and N demonstrations, there are thus $M \times N$ PDPs.

B.3 Test-Time See-Spot-Run Policy

Our test-time SSR Policy, π_{ssr}, imitates demonstrations to select an action a_t^{ssr} by comparing and scoring the similarity between demo ASEs and test scene ASEs, as per Fig. 2 and Alg. 1. π_{ssr} has three phases: (1) Embed, (2) Correspond consisting of (2a) Match with the L2 Consistency Distance (L2CD), (2b) Verify (rematch) with the Cycle Consistency Distance (CCD), then (3) Select Best Policy Demonstration Pair (PDP) and Act.

In phase 1, **Embed**, referring to Alg. 1, lines 6-8, we evaluate each of the M embedding functions E_m on the the single test state s_t (line 6) and on the N demonstrations that correspond to the current discretized task progress p_t (line 8). This yields M test ASEs H_l, each associated with N PDPs; and $M \times N$ demo ASEs H_d, each associated with exactly one PDP.

In phase 2, **Correspond**, referring to Alg. 1 line 9, and all of Alg. 2,

Algorithm 1 Test-time See-SPOT-Run Policy π_{ssr}

Input Embed Functions $E_{1..M}$, Demos $D_{1..N}$
while TASK_INCOMPLETE() **do**
  $s_t, p_t \leftarrow$ OBSERVE() ▷ Get state, progress
  $\mathcal{A}, \mathcal{C} \leftarrow \{\}, \{\}$ ▷ Action index, CCD container
  for $m \in M, n \in N$ **do** ▷ Visit each PDP
    $H_l \leftarrow E_m(s_t)$ ▷ Get live test ASEs
    $s_d, a_d \leftarrow D[n, p_t]$ ▷ Get demo state, action
    $H_d \leftarrow E_m(s_d)$ ▷ Get demo ASEs
    $\mathcal{A}[m,n], \mathcal{C}[m,n] \leftarrow f_{corr}(H_d, a_d, H_l)$
  end for
  $a_t^{\text{ssr}} \leftarrow \mathcal{A}[\text{argmin}_{m,n}(\mathcal{C})]$ ▷ min dist. action
  ACT(a_t^{ssr}) ▷ Run the agent
end while

Algorithm 2 f_{corr} Correspondence

Input Demo ASEs H_d, Action a_d, Live test ASEs H_l, Mode $b_{L2} \leftarrow 0$
$h_d \leftarrow H_d[a_d]$ ▷ Get Target Demo ASE
▷ Find test action, L2 match metric
$a_l \leftarrow \text{argmin}_{a \in A} ||h_d - H_l[a]||_2$
$\Delta_{\text{L2CD}} \leftarrow ||h_d - H_l[a_l]||_2$
if b_{L2} **then**
  return $a_l, \Delta_{\text{L2CD}}$
$h_l \leftarrow H_l[a_l]$ ▷ Get candidate test ASE
▷ Find action and dist. to demo action
$a_{\text{rematch}} \leftarrow \text{argmin}_{a \in A} ||H_d[a] - h_l||_2$
$\Delta_{\text{CCD}} \leftarrow ||a_d - a_{\text{rematch}}||_2$
return a_l, Δ_{CCD} ▷ action, CCD

we use Target Demo Actions a_d to approximate an ideal test action a_t^* by evaluating a correspondence function f_{corr} once for each PDP, with two purposes: (2a) to identify each Candidate Test Action (CTA) a_l in the live scene that most closely matches the Target Demo Action a_d, and (2b) to find the relevance metric Δ of said PDP to the test scenario, a proxy measurement for the distance between a_t^* and a_l. The call to f_{corr} in Alg. 1 line 9 made for each PDP $\{E_m, D_n\}$ selects the Target Demo Action a_d's best-match test action a_l and stores it in $\mathcal{A}_{M \times N}[m,n]$, then finds a task relevance distance Δ between action coordinates a_l and the PDP's target demo action a_d and stores it in $\mathcal{C}_{M \times N}[m,n]$. We introduce two approaches to computing Δ: our baseline L2 Consistency Distance (L2CD) and our Cycle Consistency Distance (CCD), which we compare in Sec. 5. The key to their design is that they leverage each Target Demo Action a_d (Fig. 3)'s physical cartesian coordinate in the demo scene, which is part of the action space with the same dimensions as the Demo ASE H_d, and thus $H_d[a_d] \rightarrow h_d$, *i.e.* a target demo action corresponds to exactly one Target Demo ASE h_d.

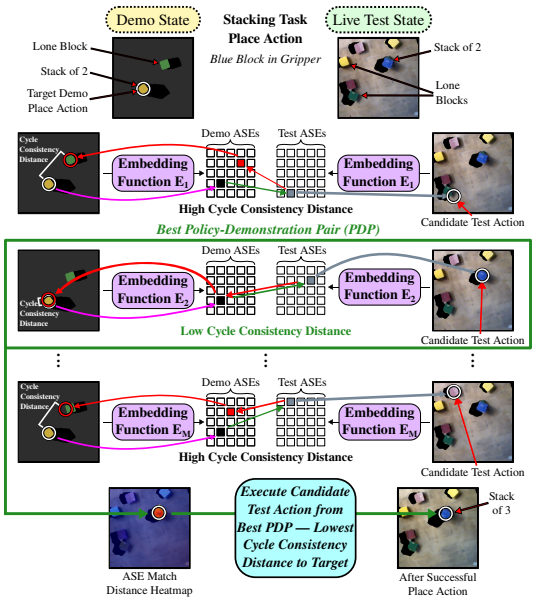

Figure 7: An illustration of cycle consistency correspondence for 3 policy-demonstration pairs (each row is a pair) as See-Spot-Run (SSR, Fig. 2, Alg. 1) chooses to place a hidden blue block already in the gripper onto the blue block visible in the test state image.

In phase 2a we find the **L2 Consistency Distance** (L2CD), referring to Alg. 2, lines 4-7, defined as the Euclidean distance between the Target Demo ASE h_d and its nearest neighbor out of all the test ASEs H_l. L2CD assumes that a CTA a_l is best represented by the closest match in ASE space to the Target Demo Action a_d embedding in H_l, where a lower L2CD is better, indicating that a_l is more relevant to the ideal action a_t^*.

While L2CD compares h_d and H_l, other demo ASEs in H_d might contain information indicating a_l is not related to a_t^*, so in phase 2b we find the **Cycle Consistency Distance** (CCD), referring to Fig. 3 and Alg. 2, lines 8-11. Starting with CTA a_l from the L2CD phase 2a, CCD verifies if a_l is a spurious match by comparing the euclidean distance between the test scene ASE $h_l \leftarrow H_l[a_l]$ (Alg. 2 line 8) and demo scene ASEs H_d to find the closest rematch coordinate a_{rematch} (Alg. 2 line 10). Measuring the physical Cartesian distance in the demo space between the Target Demo Action a_d and a_{rematch} (Alg. 2 line 11) gives us Δ_{CCD}, where a lower CCD is better, indicating that a_l is more relevant to the ideal action a_t^*. The underlying reasoning is that if a rematch is a large physical distance from a_d in the demo scene, the CTA a_l represents a poor fit for the novel demo task. Once

all calls to f_{corr} in Alg 1 have filled the action and correspondence containers $\mathcal{A}$ and $\mathcal{C}$, we proceed to phase 3.

Finally, in phase 3, **Select Best PDP and Act**, referring to Alg. 1, lines 11-12, we select the PDP with the minimal match distance, $\hat{m}, \hat{n} = \text{argmin}\, \mathcal{C}_{M \times N}$, then execute the final corresponding matched action $a_t^{\text{ssr}} = \mathcal{A}[\hat{m}, \hat{n}]$. The purpose of selecting the PDP is to compare f_{corr} distances Δ_l across all PDPs to select the most relevant policy with the best final action for the novel task, and then act, so that the agent carries a_t^{ssr} out.

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 π_{ssr} repeatedly until $\mathcal{P}(s_T) = p_{\text{max}}$, indicating that the task is complete. We evaluate See-SPOT-Run with L2CD and CCD, as well as prior work, with few-shot experiments in Sec. 5. We will also discuss how RLBD and the SSR Framework with CCD surpasses the other methods.

C Experiments — Restated in With Additional Details

Here we retrace much of the information from Sec. 5, with additional details. We will cover the robot implementation details; our evaluation metrics, which include two additional task progress efficiency metrics; an extended discussion of the simulation experiments, with a per-task breakdown; and finally we note that we have encountered restrictions due to the COVID-19 pandemic.

C.1 Robot Implementation Details

Our workspace, commands and action space are as defined in "Good Robot!" [4]. We consider a robot that is able to move to specific arm pose and gripper state given an action coordinate in our action space A (action type $k \in (grasp, place)$, x, y, and gripper rotation θ), and employ a traditional trajectory planner to compute robot arm trajectories for each action. The state s is captured with a fixed RGB-D camera which we project so that the z-axis of the camera is perpendicular to the workspace, with color heightmaps as shown in Fig. 2. We also provide depth heightmaps as a distance from the surface in meters, as pictured in "Good Robot!" [4], except, instead of repeating the most recent depth heightmap on each channel, we pack the depth heightmap from the three most recent timesteps into the input depth image's three channels. Each pixel in a state heightmap s represents a 4 mm^2 area, and each discrete gripper rotation $b \in [0, 16)$ represents a rotation of $\frac{b\pi}{4}$ radians.

In our RLBD experiments, progress for each task is determined by an observer as referred to in Alg. 1. In simulation, a scripted observer computes task progress using the known positions and orientations of blocks. In real experiments, a scripted observer can use depth measurements to compute task progress. Progress can also optionally be provided via a user interface. For step I. Before Demonstration, we perform a run of SPOT-Q training from "Good Robot!" [4] for 40k actions for each known task (row, stack, unstack, vertical square), shown in Fig. 1. We leave one resulting model out for each fold of cross validation. To generate each demonstration for Step II. Collect Demonstrations of Test Task, a human manually clicks once on each simulated image state s_i to define exact action a_i that leads to progress for untrained test task $\mathcal{T}$. The user repeats this process for each step of a multi-step task (in our case 4 to 6 total actions) to generate a demonstration D_n (Sec. B.2), with task progress $p_t = \mathcal{P}_t(s_t)$ discretized s.t. $p_t \in \mathbb{N}^{[0 \ldots 4]}$ (Sec. 3). See-SPOT-Run is agnostic to the total number of demonstrations N. With these details in place, we will next discuss our evaluation metrics.

C.2 Evaluation Metrics

Our extended set of metrics are as defined in Sec 5.1 with two added efficiency metrics. Performance metrics fall under four categories, including: four test metrics, two cost measures, two compute efficiency metrics, and two sample efficiency metrics. We describe each category and metric below.

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 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 in fewer actions. Our ideal is 6 actions for all tasks except for rows, which is 4 actions. (3) **Progress (Prog.)** is each trial's maximum proportional progress towards completing a task averaged over all trials. For instance, a stacking test trial in which the agent completed a stack of 3 blocks but could not complete the task is assigned a progress of 75%. Tasks that aren't completed 100% might be very close, so progress measures valuable capabilities that could otherwise go unnoticed when only considering the trial success rate. (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.* $\frac{\text{trial successes containing } \textit{progress reversal}}{\text{trials containing } \textit{progress reversal}}$. Better recovery rates mean tasks are more likely to be completed if the robot makes a mistake or if an outside actor interferes with the scene.

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. This metric is associated with (a) high cost human time required to write observer scripts or perform annotations [43, 44] and (b) with computer simulation or real robot execution time.

Compute Efficiency Metrics evaluate test metric benefits with respect to the cost measures when completing a novel, previously unseen task. Higher is better. We add one to the train steps denominator to prevent dividing by 0 with SSR: (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. (8) **Progress Compute Efficiency** $\frac{\text{Prog.}}{\text{Train Steps}+1}$ is the amortized percentage of proportional trial progress that can be completed for every training batch step.

Sample Efficiency Metrics measure test metric performance gains amortized over the annotated actions that need to be collected on the test task. Higher is better: (9) **Trial Success per Annotated Action** $\frac{\text{Trials}}{\text{Ann. Actions}}$ measures the increase in percentage of trials that can be completed amortized over annotated actions on a test task. (10) **Progress per Annotated Action** $\frac{\text{Prog.}}{\text{Ann. Actions}}$ measures the increase in the average proportion of task steps that can be completed amortized over annotated actions on a test task.

Altogether, our metrics quantify the broad improvements in task performance and reductions in the resources necessary to perform novel tasks. Most importantly, we consider critical efficiency measures that have not previously been evaluated in the baselines that we compare to, which helps to motivate the broad range of useful applications for both RLBD and SSR.

C.3 Simulation Experiments

Here we cover the fine points of our main Simulation Experiments Section 5.2. We pretrain RL policies for four tasks: stacking, row-making, unstacking, and 2x2 vertical square (see Fig. 1). For our experiments we collect two different demonstrations of each task, the minimum necessary to demonstrate robustness to diverging states and demonstrations, *e.g.* making a vertical square both rows first and stacks first[4]. We then evaluate the methods specified in Section 4 by conducting four fold cross validation 'Leave-One-Out' experiments. Namely, for each task, we use the remaining three models as the policies, *e.g.* row, unstack, and vertical square policies for stack evaluation, from which we generate ASEs for our See-SPOT-Run approach, as shown in Fig. 1.

Our TransporterNet [29] baseline is run in their *ravens* framework with no rotation limitations; however they assume high friction (bullet sim friction 1.0 vs our value of 0.5), and use a suction cup gripper with perfect point-to-point grasp place capabilities. To provide a fair comparison we adjust the amount of object slippage they permit from 1.5 cm to 1 cm, which is based on the distance a block can shift with our Robotiq 2f-85 gripper, and eliminate their object rotation restriction by increasing their limit from 30°to 180°. The key results of our simulation experiments from Sec 5 and Tab. 1, are also outlined in Tab. 4, with the addition of our extended efficiency metrics.

The "Good Robot!" [4] metrics in Tab. 1 and 4 are present to accentuate the efficiency improvements we discuss and to provide a test metric ceiling. The performance of "Good Robot!" [4] on the test metrics is not a basis for comparison to the other methods because the policies are trained from

[4]Actions taken based on different demonstrations can be seen in our supplementary video, where both stack-first and row-first vertical square task progress is visible on the real robot.

scratch on the test task; the models would reliably fail if scored on a novel test task since there is no demonstration mechanism. Such an experiment is out of scope since it would not be informative about the underlying methods.

Simulation Task	Average Test Metrics				Costs		Compute Efficiency		Sample Efficiency	
	Trials	Action Efficiency	Prog.	Recov.	Train Steps	Annotated Actions	Trials / Train Steps+1	Prog. / Train Steps+1	Trials / Ann. Actions	Prog. / Ann. Actions
TransporterNet [29]	30%	35%	50±3%	8%	40k	12	0.00075%	0.0013%	2.5%	4%
L2 Consistency Dist.	13%	30%	67±1%	5%	0	12	13%	67%	1.1%	6%
See-Spot-Run (ours)	**36%**	**41%**	**79±1%**	**19%**	**0**	12	**36%**	**79%**	**3.0%**	**7%**
"Good Robot" [4] *	91%*	57%*	96±1%*	90%*	120k	40k	0.00076%	0.0008%	0.0023%	0.0024%

Table 4: Simulation task performance on the metrics detailed in Sec. C.2, 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 [29] 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.

As described in Sec. 5.2, our See-Spot-Run with Cycle Consistency (labeled See-Spot-Run in Tab. 4, and SSR CCD in Tab. 1) significantly outperforms both comparable methods on the overall test metrics, with 36% of trials completed, 79% average progress, 41% action efficiency, and a 29% recovery rate; L2CD gets 13%, 67%, 30%, and 5%, respectively, which demonstrates the benefits of the rematch verify Cycle Consistency step in Alg. 2; and TransporterNet gets 30%, 50%, 35%, and 8%, respectively, which shows the overall improvements in performance compared to similar prior work for few shot tasks.

Table 4 provides additional performance metrics about the three methods we evaluate in simulation. The relative progress efficiency improvement of SSR with CCD over TransporterNet [29] is even larger than for trial efficiency. SSR has a 75% improvement in progress per annotated action and over TransporterNet, and a 61,000x improvement in progress per train step. Comparing SSR with CCD vs. L2CD, at 79% vs. 67% progress per train step and 7% vs. 6% progress per annotated action, respectively, shows CCD continues to outperform L2CD on the progress efficiency metrics.

The rate of recovery in trials with a *progress reversal* is a notable aspect across all trials, as our 19% recovery rate for SSR with CCD is substantially better than the 8% recovery rate for TransporterNet. This might be attributable to the strengths of SSR over pure imitation learning methods, where our pretraining of RL models explicitly incorporates failures and exploration data into ASEs. See-Spot-Run models may recover where the TransporterNet models cannot because the former can recognize the scene after making an error in action choice, while the latter enters an unseen domain. Comparing the recovery rates of SSR with CCD at 19% vs L2CD at 5% shows the value of evaluating not just the test scene ASEs, but also the demo scene data contained in Demo ASEs.

This completes our examination of the summary results. Next, we will discuss the task-specific breakdown of the simulation experiments found in Sec. 5, Table 2:

Sim Stack: SSR (Ours), L2CD, and TransporterNet complete 24%, 0%, and 12% of trials, respectively, while the average trial progress is 75%, 51%, and 51%, respectively. The rate of successful recoveries is 24%, 0%, and 12% for SSR, L2CD and TransporterNet, respectively. SSR is substantially better at stacking compared to both baselines with respect to each of these metrics.

Sim Unstack: Our SSR with cycle consistency completes 66% of trials, compared to 38% for L2CD and 86% for TransporterNet. While unstacking is the easiest task, it is also particularly informative because it requires extracting and acting on behaviors which are explicitly low scoring actions in the trained stacking model. This means SSR correctly avoids choosing stacking actions associated with a high Q-Value. This is also the only case where TransporterNet outperforms SSR, which is due to its propensity to match a base object with its point-to-point "transport" design, but the row task will illustrate the shortcomings that are a tradeoff of their approach. Progress is significant for SSR, L2CD, and TransporterNet with 82%, 76%, and 94%, respectively. Unstacking has no notion of recovery since knocking the stack over would make this a single step task, and thus toppling the stack is a failure. An example of a relevant application is unstacking fragile pallets in a warehouse.

Sim Row: Our SSR with cycle consistency completes 30% of trials, L2CD models complete 6% of trials, and TransporterNet completes 2% of trials; with respective progress rates of 81%, 73%, and 24%; and recovery rates of 28%, 7%, and 2%. Here, TransporterNets rarely manages to place the fourth block. This case highlights TransporterNet's propensity to match specific locations anchored by an object, a design component which fails when actions must occur where no such object exists. By contrast, SSR is very consistent wherever place actions should occur relative to existing objects.

Sim Square: SSR with cycle consistency completes 24% of trials, L2CD distance completes 8% of trials, and TransporterNet completes 20% of trials; with respective progress rates of 79%, 67%, and 32%; and recovery rates of 14%, 7% and 9%. SSR is again the strongest on these metrics, and the difference can in part be attributed to TransporterNets' propensity for matching anchored locations.

This concludes our extended simulation results. Our final note regards COVID-19 restrictions.

C.4 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%	$\mathbf{80}\pm\mathbf{5}\%$	22%
Unstack	**90%**	86%	$\mathbf{97}\pm\mathbf{3}\%$	–
Row	**30%**	28%	$\mathbf{77}\pm\mathbf{7}\%$	66%
Square	**10%**	27%	$\mathbf{75}\pm\mathbf{4}\%$	10%
Average	**40%**	42%	$\mathbf{82}\pm\mathbf{3}\%$	33%

Table 5: 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 ($\pm$) 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%.

D COVID-19 Restrictions

Some aspects of our experiments were limited by restricted access to the physical robot and the computer hardware due to the ongoing COVID-19 pandemic.