7 Appendix

7.1 Neural Network Architecture

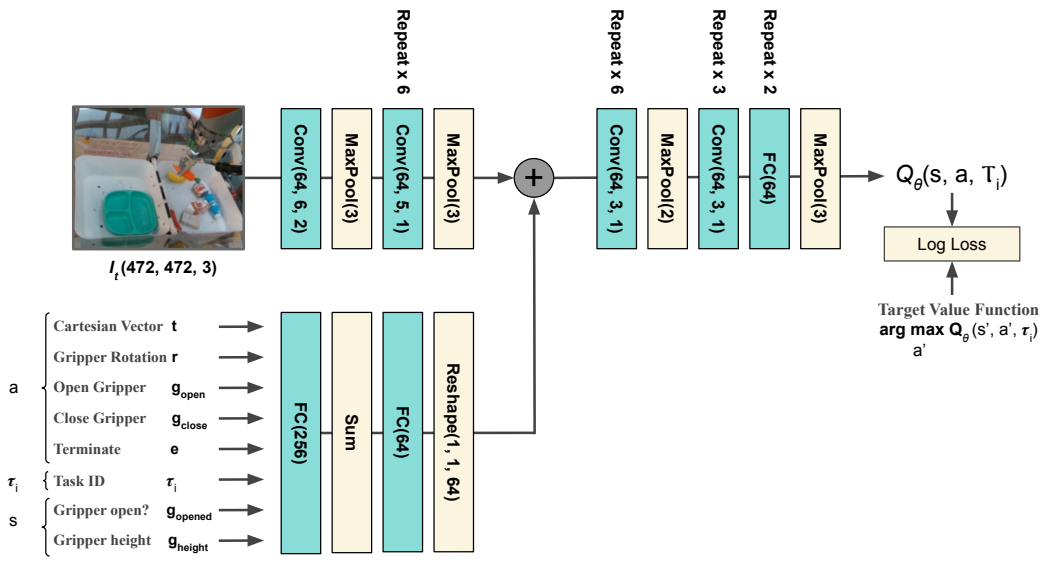

Figure 8: The architecture of MT-Opt Q-function. The input image is processed by a stack of convolutional layers. Action vector, state vector and one-hot vector $\mathcal{T}_i$ representing the task of interest are processed by several fully connected layers, tiled over the width and height dimension of the convolutional map, and added to it. The resulting convolutional map is further processed by a number of convolutional layers and fully connected layers. The output is gated through a sigmoid, such that Q-values are always in the range [0, 1].

We model the Q-function for multiple tasks as a large deep neural network whose architecture is shown in Fig. 8. This network resembles one from [1]. The network takes the monocular RGB image part of the state s as input, and processes it with 7 convolutional layers. The actions a and additional state features (g_{status}, g_{height}) and task ID $\mathcal{T}_i$ are transformed with fully-connected layers, then merged with visual features by broadcasted element-wise addition. After fusing state and action representations, the Q-value $Q_\theta(s, a)$ is modeled by 9 more convolutional layers followed by two fully-connected layers. In our system the robot can execute multiple tasks from in the given environment. Hence the input image is not sufficient to deduce which task the robot is commanded to execute. To address that, we feed one-hot vector representing task ID into the network to condition Q-Function to learn task-specific control.

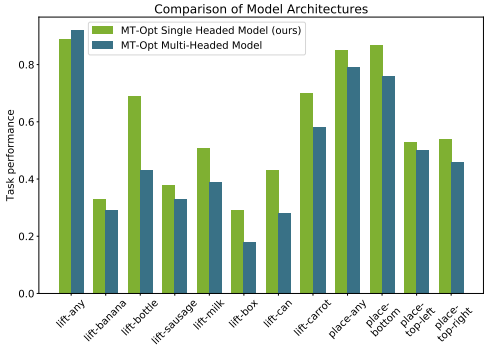

Figure 9: Comparison of single-headed and multi-headed neural networks approximating the Q-function. In both cased task ID was fed as the input to the network. Multi-headed architecture of the Q-function under-performs on a wide range of tasks, winning only on *lift-any* tasks which has most of the data.

In addition to feeding task ID we have experimented with multi-headed architecture, where n separate heads each having 3 fully connected layers representing n tasks were formed at the output of the network. Fig.9 shows that performance of the system with the multi-headed Q-function architecture is worse almost for all tasks. We hypothesize that dedicated per task heads "over-compartmentalizes" task policy, making it harder to leverage shared cross-task representations.

7.2 Description of Scripted Policies

As discussed in Section4 we use two crude scripted policies to bootstrap easy generic tasks.

Scripted Picking Policy: To create successful picking episodes, the arm would begin the episode in a random location above the right bin containing objects. Executing a crude, scripted policy, the arm is programmed to move down to the bottom of the bin, close the gripper, and lift. While the success rate of this policy is very low ($\approx 10\%$), especially with the additional random noise injected into actions, this is enough to bootstrap our learning process.

Scripted Placing Policy: The scripted policy programmed to perform placing would move the arm to a random location above the left bin that contains a fixture. The arm is then programmed to descend, open the gripper to release the object and retract. This crude policy yields a success rate of (47%) at the task of placing on a fixture (plate), as the initial fixture is rather large. Data collected by such a simplistic policy is sufficient to bootstrap learning.

7.3 $f_{I_{skill}}$ impersonation strategy details

Task impersonation is an important component of the MT-Opt method. Given an episode and a task definition, the SD classifies if that episode is an example of a successful task execution according to that particular goal definition. Importantly, both the success and the failure examples are efficiently utilized by our algorithm. The success example determines what the task is, while the failure example determines what the task is *not* (thus still implicitly providing the boundary of the task), even if it's an example of a success for some other task. Fig.11 shows offline success rates and Fig.12 shows by how much the per task data is expanded using the $f_{I_{\text{skill}}}$ impersonation function.

Algorithm 1 Task Impersonation

procedure $f_I(e^i$: original_episode)
  expanded_episodes = []
  $SD\{k_i\} \leftarrow$ set of SDs relevant to task T_i
  for SD_k in $SD\{k_i\}$ **do**
    // e^k: e^i but rewards for task T_k not T_i
    $e^k = SD_k(e^i)$
    expanded_episodes.append(e^k)
  return expanded_episodes

In Section 3.2 we discuss a problem arising when using a naive $f_{I_{all}}$ episodes impersonation function, and suggest a solution to impersonate data only within the boundaries of a skill. Namely, given an episode e_i generated by task $\mathcal{T}_i$, a skill S_j that task belongs to is detected. The e_i will be impersonated only for the tasks $\mathcal{T}_{\{S_j\}}$ belonging to that particular skill.

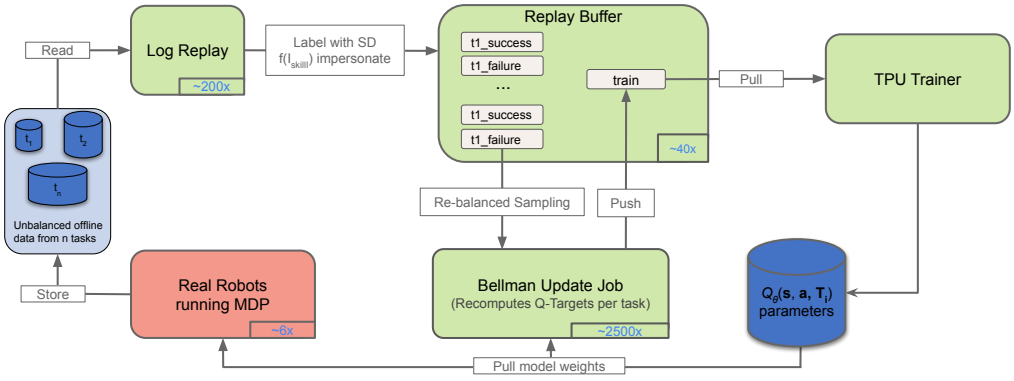

Figure 10: **System overview:** Task episodes from disk are continuously loaded by LogReplay job into task replay buffers. LogReplay process assigns binary reward signal to episodes using available Success Detectors and impersonates episodes using $f_{I_{skill}}$ (or other strategy). Impersonated episodes are compartmentalized into dedicated per task buffers, further split into successful and failure groups. Bellman Update process samples tasks using re-balancing strategy to ensure per task training data balancing and computes Q-targets for individual transitions, which are placed into train buffer. These transitions $(\mathbf{s}, \mathbf{a}, \mathcal{T}_i)$ are sampled by the train workers to update the model weights. The robot fleet and Bellman Update jobs are reloading the most up to date model weights frequently.

Note, that sometimes impersonation for all $\mathcal{T}_{\{S_j\}}$ tasks within a skill could result in too excessive data sharing. For example, the bulk of the data for our *object-acquisition* skill represents variants of tasks involving foods objects. If we want to learn a new task within the same skill using visually significantly different objects, e.g. transparent bottles, all offline episodes involving the plastic objects will be (correctly) impersonated as failures for the *lift-transparent-bottle* task. That is, a few intrinsic failures for that task will be diluted in large set of artificially created negatives.

To solve this issue we introduce a stochastic impersonation function. An impersonated episode candidate will be routed to training with the probability p_s if it's a success, or with probability p_f if it's a failure. We experiment with $p_s = 1.0$, and $p_f <= 1.0$. The reasoning is that it's always desirable to utilize surplus impersonated examples of a *successful* task execution, but it could be better to utilize only a fraction of the surplus *failures* to balance intrinsic v.s. artificial failures for that task.

This gives rise to the $f_{I_{skill}}(p_s, p_f)$ impersonation function which is suitable in some situations explained above.

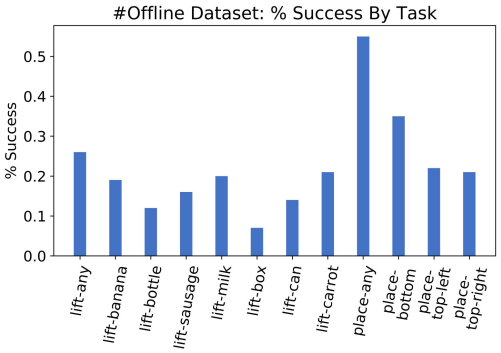

Figure 11: Effective success rate for each task in our offline dataset. This plot represents the distribution of successes within the entirety of our offline dataset collected over time from many policies, not the performance of any particular policy.

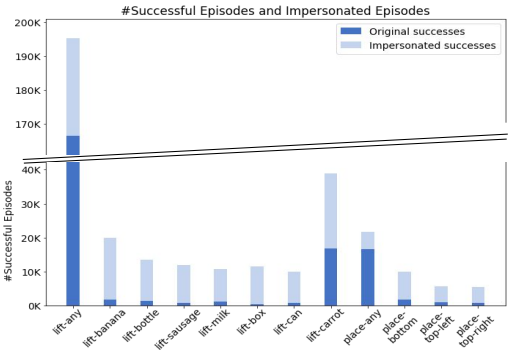

Figure 12: Practical effect of task impersonation for successful outcomes. Dark blue indicates data specifically collected for a task; light blue indicates episodes impersonated from some other tasks which happen to be a success for the target task.

7.4 Distributed Asynchronous System

Fig.10 provides an overview of our large scale distributed Multi-Task Reinforcement Learning system.

8 Reward Specification with Multi-Task Success Detector

Training a visual success detector is an iterative process, as a new task initially has no data to train from. We have two strategies to efficiently create an initial SD training dataset for a new task. 1) We collect 5Hz videos from 3 different camera angles where every frame of the video a human is demonstrating task success, and then a short video demonstrating failure (see examples in Fig. 14. Note that the user shows the desired and non-desired *outcome* of the task, not to be confused with demonstrations of how the task needs to be done.

The intention here is to de-correlate spurious parts of the scene from task-specifics. This process is repeated for approximately 30 minutes per task. 2) We relabel data from a policy that occasionally generated success for the new task (e.g., relabel *lift-any* data for *lift-carrot* task.).

The user would then change the lighting, switch out the objects and background, and then collect another pair of example videos (see Fig. 14 for example one video where there is always something on a plate being moved around paired with another video where there is never anything on a plate).

Once the initial SD is trained, we can train an RL policy, and begin on-policy collection. We continue to label on-policy data which keeps coming for the new task until SD is reliable. Table 3 shows false positive and false negative error rates on holdout data for the SD model used in our ablations. Our holdout data consisted of all images from a particular robot.

During the SD training process, the data is artificially augmented to improve generalization, which involves cropping, brightening, rotating, and superimposing random shadows onto the images.

Primary SD Name	Total Count	Success Count	Failure Count	Success Rate	F. Neg. Rate	F. Pos. Rate	Other F. Neg. Rate	Other F. Pos. Rate
lift-any	16064	7395	8672	46%	1%	2%	0%	0%
lift-banana	6255	510	5745	8%	2%	1%	0%	1%
lift-bottle	6472	430	6042	7%	5%	1%	0%	1%
lift-sausage	6472	461	6011	7%	3%	0%	0%	1%
lift-milk	6472	158	6314	2%	7%	0%	3%	9%
lift-box	6467	487	5980	8%	1%	1%	0%	2%
lift-can	6467	270	6197	4%	2%	0%	3%	3%
lift-carrot	6481	911	5570	14%	0%	1%	0%	0%
place-any	3087	1363	1724	44%	1%	2%	0%	0%
place-bottom	2893	693	2200	24%	2%	1%	1%	3%
place-top-left	2895	346	2549	12%	10%	0%	3%	8%
place-top-right	2897	312	2585	11%	4%	0%	0%	5%

Table 3: Success detection holdout data statistics. Table shows success detector error rate for held out labelled success detector data. We split out the evaluation dataset based on the robot, e.g. all data generated by Robot #1 is used for evaluations and not for training. This strategy results in a much better test of generalization power of the success detector, compared to the conventional way to split out 20% of the data randomly for evaluation. The Other Task False [Positive/Negative] Rates columns indicates how well the success detector for a task A classifies outcomes for all other tasks. For example we want to ensure that a successful *lift-carrot* episode does not trigger *lift-banana* success, i.e. not only a success detector should manifest its dedicated task success, but also reliably reason about other related tasks. This "contrastiveness" property of the success detectors is of great importance in our system. As success detectors determine tasks data routing and experience sharing, an error in this tasks data assignment would drive anti-correlated examples for each task, resulting in a poor performance of the system.

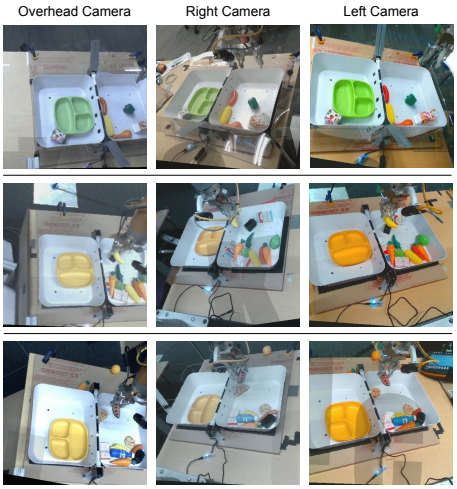

Figure 13: SD training images. Each row represents a set of images captured at the same time that are fed into the SD model. These images demonstrate our train-time SD data augmentation process as they have been distorted via cropping, brightening, rotating, and superimposing of shadows.

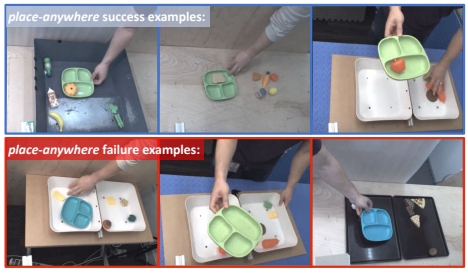

Figure 14: Video frames for the *place-anywhere* task. Success and failure videos are iteratively captured in pairs to mitigate correlations with spurious workspace features such as hands of the user, backgrounds, bins, and distractor objects.

Fig. 13 shows training images after these distortions have been applied. Our success detector model is trained using supervised learning, where we balance the data between success and failures as well as tasks. We use the architecture that is based on that from [67] with the exception of the action conditioning as it is not needed for this classification task. For each task the network outputs the probability representing whether a given state was a success or failure for the corresponding task. The model receives three images as an input that come from an over-the-shoulder camera (same image as RL policy), and two additional side cameras. These side camera images are only used by the SD model, not the RL model. The additional cameras ensured that the task goals would be unambiguous, with a single camera, it was often difficult for a human to discern from an image whether or not the task had succeeded.

A breakdown of the labelled SD training data is provided in Fig. 15. While training SD, we incorporated data sharing logic based on task feasibility. For example any success for *lift-carrot* would also be marked as failure for all other instance lifting tasks, and as a success for *lift-any*. In this manner, the original set of labelled data shown in Fig. 15 could act effectively as a much larger dataset for all tasks, where successes of one task often worked an interesting negatives for other tasks. Additionally we balanced the proportion of success and failure examples per task seen by the model during training.

9 Robot Setup

In order for our system to be able to learn a vision-based RL policy that can accomplish multiple tasks, we need to collect a large, diverse, real-robot dataset that represents data for various tasks.

To achieve this goal, we set up an automated, multi-robot data collection system where each robot picks a task $\mathcal{T}_i$ to collect the data for. Collected episode is stored on disk along with the $\mathcal{T}_i$ bit of information. Our learning system can then use this episode collected $\mathcal{T}_i$ for to train a set of other tasks utilizing MT-Opt data impersonation algorithm. Once the episode is finished, our data collection system decides whether to continue with another task or perform an automated reset of the workspace.

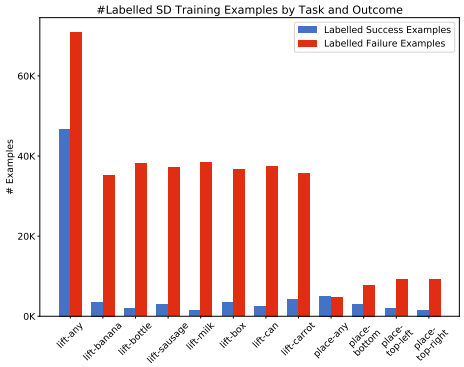

Figure 15: Counts of labelled SD training data by task and outcome. This data was generated either from human video demonstration, or by labelling terminal images from episodes produced by a robot. Note, that not all of the negatives were hand-labelled. As we may know dependencies between the tasks, e.g. that a success for *lift-carrot* is always a failure for *lift-banana*, we can automatically generate negative examples. Similarly, all successes for the semantic lifting tasks are also successes for the *lift-any* task.

In particular, we utilize 7 KUKA IIWA arms with two-finger grippers and 3 RGB cameras (left, right, and over the shoulder). In order to be able to automatically reset the environment, we create an actuated resettable bin, which further allows us to automate the data collection process. More precisely, the environment consists of two bins (with the right bin containing all the source objects and the left bin containing a plate fixture magnetically attached anywhere on the workbench) that are connected via a motorized hinge so that after an episode ends, the contents of the workbench can be automatically shuffled and then dumped back into the right bin to start the next episode. Fig. 16 depicts the physical setup for data collection and evaluation. This data collection process allows us to collect diverse data at scale: 24 hours per day, 7 days a week across multiple robots.

One episode has ≈ 10 steps on average, taking ≈ 25 seconds to be generated on a robot, including environment reset time. This accounts to ≈ 3300 episodes/day collected on a single robot, or $\approx 23K$ episodes/day collected across our fleet of 7 robots.

9.1 Details of Data Collection to bootstrap a Multi-Task System

Task Name	#Eps.	QT-Opt	$f_{I_{\text{orig}}}$, rand QT-Opt MultiTask	$f_{I_{\text{orig}}}$, rebal	$f_{I_{\text{all}}}$, rand DataShare MultiTask
lift-any	635K	0.88	0.94	0.85	0.62
lift-banana	9K	0.04	0.13	0.38	0.09
lift-bottle	11K	0.02	0.16	0.66	0.15
lift-sausage	5K	0.02	0.10	0.38	0.15
lift-milk	6K	0.01	0.13	0.42	0.13
lift-box	6K	0.00	0.12	0.16	0.08
lift-can	6K	0.01	0.16	0.46	0.07
lift-carrot	80K	0.71	0.41	0.72	0.37
place-any	30K	N/A	**0.86**	0.74	0.30
place-bottom	5K	N/A	0.43	0.57	0.30
place-top-right	4K	N/A	0.16	**0.55**	0.08
place-top-left	4K	N/A	0.23	**0.75**	0.19
Min		0.00	0.10	0.16	0.07
25-th percentile		0.00	0.13	0.41	0.09
Median		0.01	0.16	**0.56**	0.15
Mean		0.14	0.32	0.55	0.21
75-th percentile		0.03	0.42	0.73	0.30
Max		0.88	0.94	0.85	0.62
Mean (low data)		0.01	0.18	0.42	0.13

Task Name		$f_{I_{\text{all}}}$, rebal	$f_{I_{\text{skill}(1,\,0.15)}}$, rebal	$f_{I_{\text{skill}(1,\,1)}}$, rand	$f_{I_{\text{skill}(1,\,1)}}$, rebal (ours)
lift-any		**0.95**	0.80	0.88	0.89
lift-banana		0.30	0.58	**0.62**	0.33
lift-bottle		0.48	0.68	0.55	**0.69**
lift-sausage		0.39	**0.42**	0.28	0.38
lift-milk		0.27	0.27	**0.52**	0.51
lift-box		0.22	0.12	0.28	**0.29**
lift-can		0.28	**0.47**	0.43	0.43
lift-carrot		**0.75**	0.52	0.71	0.70
place-any		0.24	0.83	0.57	0.85
place-bottom		0.02	0.62	0.17	**0.87**
place-top-right		0.10	0.26	0.27	0.54
place-top-left		0.16	0.39	0.22	0.53
Min		0.02	0.12	0.17	**0.29**
25-th percentile		0.20	0.36	0.28	**0.42**
Median		0.28	0.50	0.48	0.54
Mean		0.35	0.49	0.46	**0.58**
75-th percentile		0.41	0.64	0.58	**0.74**
Max		**0.95**	0.83	0.88	0.89
Mean (low data)		0.21	0.36	0.32	**0.5**

Table 4: Quantitative evaluation of MT-Opt with different data impersonation and re-balancing strategies. This table reports performance of 7 different models on the 12 ablation tasks, trained on identical offline dataset, with identical computation budget, and evaluated executing 100 attempts for each task for each strategy on the real robots (totaling to 12*100*7=8400 evaluations). In all cases a shared policy for all 12 tasks is learned. The difference across the strategies is in the way the data is impersonated (expanded), and in the way the impersonated data is further re-balanced. The last column is our best strategy featuring skill-level data impersonation and further data re-balancing. This strategy outperforms other strategies on many different percentiles across all 12 tasks; however the effect of that strategy is even more pronounced for the tasks having scarce data, e.g. *lift-can*, *lift-box*, *place-top-right*, see Mean (low data) statistic. The column #2 indicates the number of episodes which were collected for each task.

This section contains more details on the data collection process introduced in Section 9.1. Real world robot data is noisy. For this project nearly 800,000 episodes were collected through the course of 16 months. The data was collected over different:

1. Locations: Three different physical lab locations.
2. Time of day: Robots ran as close to 24x7 as we could enable.
3. Robots: 6-7 KUKAs with variations in background, lighting, and slight variation in camera pose.
4. Success Detectors: We iteratively improved our success detectors.
5. RL training regimes: We developed better training loops hyper-parameters and architectures as time went on.
6. Policies: Varied distribution of scripted, epsilon greedy, and on-policy data collection over time.

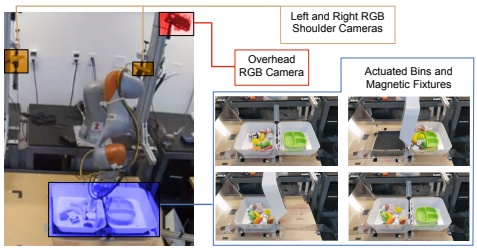

Figure 16: Robot workspace consisting of an overhead camera (red), two over the shoulder cameras (brown), and a pair of articulated resettable bins with a plate fixture that can be magnetically attached to the bin (blue).

Figure 17: Evaluation scene used for ablation experiments. Contains one of three different color plates. And nine graspable objects: One of each object from our seven object categories with two extra toy food objects sometimes from the seven object categories, sometimes not.

Our data collection started in an original physical lab location, was paused due to COVID-19, and the robots were later setup at a different physical lab location affecting lighting and backgrounds. Initially scripted policies were run collecting data for the *lift-anything* and *place-anywhere* tasks. Once performance of our learned policy for these tasks out-performed the scripted policy we shifted to a mix of epsilon greedy and pure on-policy data collection. The majority of our episodes were collected for the *lift-anything* and *place-anywhere* tasks with learned policies. It is worth mentioning that over the course of data collection many good and bad ideas where tried and evaluated via on-policy collection. All of these episodes are included in our dataset. Additional tasks being incorporated over time.

After we had a policy capable of the *lift-anything* and *place-anywhere* tasks we introduced more specific variations of pick and place tasks where either a specific object needed to be picked, or an object needed to be placed in a specific location on the plate. At this point, our data collection process consisted of executing a randomly selected pick task followed by a randomly selected place task.

As a result of the collection process described above, we were left with a 800,000+ episode offline dataset, very diverse along tasks, policies, success rate dimensions.

10 Details for real world experiments

The robot workspace setup for the 12 task ablations is shown in Fig 17. Table 4 summarizes studies of 7 different data impersonation and re-balancing strategies for 12 tasks. The last column features the model which on average outperforms other strategies. Note that this strategy is not the best across the board. For example, due to big imbalance of our offline dataset, the native data management strategy (column #3) yields best performance for the over represented tasks, but very bad performance for underrepresented tasks.