**Robotics: Science and Systems 2025**
**Los Angeles, CA, USA, June 21–25, 2025**

# Curating Demonstrations using Online Experience

Annie S. Chen*, Alec M. Lessing*, Yuejiang Liu, Chelsea Finn
Stanford University

***Abstract*—Many robot demonstration datasets contain heterogeneous demonstrations of varying quality. This heterogeneity may benefit policy pre-training, but can hinder robot performance when used with a final imitation learning objective. In particular, some strategies in the data may be less reliable than others or may be underrepresented in the data, leading to poor performance when such strategies are sampled at test time. Moreover, such unreliable or underrepresented strategies can be difficult even for people to discern, and sifting through demonstration datasets is time-consuming and costly. On the other hand, policy performance when trained on such demonstrations can reflect the reliability of different strategies. We thus propose for robots to self-curate based on online robot experience (Demo-SCORE). More specifically, we train and cross-validate a classifier to discern successful policy roll-outs from unsuccessful ones and use the classifier to filter heterogeneous demonstration datasets. Our experiments in simulation and the real world show that Demo-SCORE can effectively identify suboptimal demonstrations without manual curation. Notably, Demo-SCORE achieves over 15-35% higher absolute success rate in the resulting policy compared to the base policy trained with all original demonstrations.**

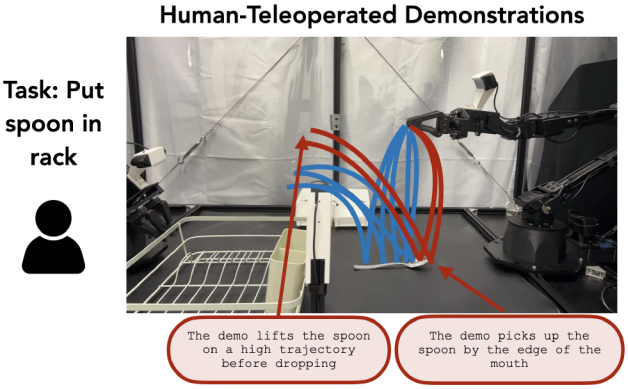


Fig. 1: **Human Demonstrations may be Unreliable.** Human demonstrations may include a wide range of strategies, not all of which are beneficial for the robot to learn. For example, some humans may try to complete the task by picking up the spoon by its edge or by lifting it above the rack and dropping it, but these strategies may be unreliable for imitation by a robot policy.

## I. Introduction

Learning from demonstrations is a powerful technique for acquiring a range of different robotic manipulation skills. Although large, diverse demonstration datasets are beneficial for pre-training, a more selective use of demonstrations may be required to achieve optimal performance. As robot learning datasets scale to include demonstrations from multiple operators across increasingly complex tasks, they naturally encompass a wide range of strategies—some of which may be challenging for robots to replicate reliably or lack sufficient examples for effective learning. For example, when teaching a robot to pick up a spoon, a human operator may collect demonstrations using a strategy that may be unreliable for the robot to imitate, like gripping the edge of the spoon head, which may be prone to slipping out of the robot's grasp, as shown in Figure 1. As such, in this paper, we study how to effectively curate demonstration datasets to enable optimal policy performance.

Manual labeling and filtering of such data is both time-consuming and error-prone, especially as the scale of robot learning datasets continues to grow. Moreover, determining which demonstrations are unreliable can be difficult, even for human experts, as successes for most real-world tasks exist on a spectrum of approaches with subtle variations in execution and context dependence, potentially making it difficult to describe or group different strategies together. For example, when grasping objects, demonstrations might vary continuously in terms of approach angles, contact points, and force application. Without careful curation, robots may learn strategies that succeed in specific demonstrations but are brittle under slightly different conditions, limiting their generalization and robustness. This motivates the need for an automated approach to selectively curate demonstrations *within a single task/dataset*, allowing robots to learn only the most reliable and effective strategies for that task.

Our key insight is that when a policy is trained on heterogeneous demonstrations, its performance during rollouts can reveal which demonstrated strategies are most reliable. Building on this observation, we present Demo-SCORE, which leverages online policy rollout experiences to automatically curate reliable demonstrations. The approach begins by training an initial policy on the full set of demonstrations. Rollouts are generated from this policy and used to train a classifier that approximately distinguishes successful trajectories from failed ones, thereby learning to separate the underlying strategies by their reliability. One challenge is that classifier may overfit to rollouts from one policy checkpoint–to address this, we cross-validate its performance on rollouts from another checkpoint in the initial policy training run. By applying this classifier to the original dataset, Demo-SCORE identifies and discards unreliable examples. This method works because it leverages the robot's own experience through rollouts, allowing it to assess the true reliability of different strategies after policy learning. Thus, by focusing on the end results of policy rollouts, Demo-SCORE filters demonstrations based on practical outcomes rather than potentially misleading surface-

*Equal contribution. Correspondence to asc8@stanford.edu. Videos of real-world trials at https://anniesch.github.io/demo-score/.

level cues.

We evaluate Demo-SCORE on a set of simulated and real-world robotic manipulation tasks with heterogeneous demonstration mixtures that include a variety of strategies. More specifically, our evaluation suite spans multiple tasks in Robosuite [27] along with simulated and real-world ALOHA [48] with different imbalanced mixtures of higher and lower quality demonstrations, and we evaluate with different policy classes, including both Diffusion Policy [12]. ACT [48]. Our experiments show that policies trained on our filtered demonstrations significantly outperform the base policies trained on all demonstrations, leading to 15-35% higher absolute success rate. Demo-SCORE also significantly outperforms prior approaches that also leverage rollout data, like autonomous imitation learning, achieving almost 2x lower failure rate. We additionally provide a detailed empirical analysis of the design choices of our method and show that it is robust to these and to hyperparameter choices. Our results highlight the benefits of curating demonstration data and show that our simple approach to automated filtering can significantly enhance policy performance in robot learning from demonstrations.

## II. Related Work

**Behavior Cloning.** Behavior cloning has traditionally been formulated as supervised learning from homogeneous expert demonstrations [2, 33, 45]. In practice, however, human demonstrations often exhibit wide variations in strategy [42], proficiency [3], and preference [19], especially in large-scale datasets [7, 20, 30]. To handle such variation, recent methods have sought to model the distribution of actions or action chunks [5, 13, 16, 24, 26, 39, 46, 48] or decompose demonstrations into shorter sequences corresponding to distinct high-level strategies [6, 28, 49]. Nonetheless, not all strategies are equally optimal: some can be inefficient, overly complex, or brittle to perturbations, thereby hindering robust deployment [9, 18, 21, 32, 41]. Our method addresses this issue by explicitly curating demonstration datasets.

**Data Curation.** Early efforts in data curation often rely on hand-crafted heuristics, such as near-duplicate removal [23], metadata balancing [38], and diversity expansion [15]. While manual curation has shown promise in the development of robot foundation models [7], it is inherently limited in its ability to capture nuanced aspects of data quality. For example, the commonly used heuristic of maximizing state diversity has been shown not to always be beneficial [4]. Recent works have instead shifted towards curating training data based on properties derived from the learned policy. Notably, Du et al. [14], Nasiriany et al. [31] leverage the embeddings of a small set of high-quality demonstrations to retrieve relevant examples from a larger candidate dataset. Hejna et al. [17] propose to estimate the importance weights of subsets through the lens of domain robustness [43]. However, estimating the quality of individual demonstrations within a single task remains an open challenge. Our work addresses this challenge by identifying and pruning suboptimal demonstrations.

**Suboptimal Demonstrations.** Previous works have attempted to tackle suboptimal demonstrations in two primary ways. One line of research assumes access to annotations, such as reward [37] or ranking [9], for all collected demonstrations, aiming to learn an optimal policy via weighted behavior cloning [40, 44] or offline reinforcement learning [25]. However, gathering dense annotations at scale beyond sparse binary success detection can be tedious and expensive. As such, another line of work seeks to learn a reward function from a limited number of annotations (e.g., quality [44], confidence [34], preference [11]) and subsequently estimate the rewards of a broader unlabeled dataset. While these methods reduce annotation expenses, they still rely on *human definitions of data quality*, which does not always align with robot capabilities or task conditions. In contrast, we leverage policy rollouts to identify suboptimal trajectories autonomously and only use sparse binary success/failure annotations of the data. Prior works like autonomous imitation learning [1, 8, 29] and reward-conditioned policies [22, 35] also leverage policy rollouts filtered by success as additional data to improve the policy. Compared to the these, we find in Section V that Demo-SCORE is more effective at improving policy performance. By grounding curation directly in the downstream task and embodiment, our method provides a scalable, effective pathway for filtering demonstrations.

## III. Preliminaries

We address a problem setting where we are given a dataset of $N$ successful demonstrations, $\mathcal{D}_{\text{demo}} = \{\tau_1, \ldots, \tau_N\}$. Each demonstration $\tau_i = \{s_0, a_0, s_1, a_1, \ldots, s_T, a_T\}$ consists of a trajectory of states and actions. These demonstrations are collected in a Markov Decision Process (MDP) characterized by $(\mathcal{S}, \mathcal{A}, P, r)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P$ is the transition dynamics, $r$ is the reward function. The goal is to learn a policy $\pi_\theta(a|s)$ that maximizes expected cumulative reward, $\mathbb{E}[R] = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$, during deployment. We additionally have the opportunity to collect a small amount of experience in the environment for further offline training, where experience is labeled with a binary indicator for success or failure.

We assume that all of the demonstrations in $\mathcal{D}_{\text{demo}}$ are successful (i.e. achieve reward $r = 1$), or equivalently that any unsuccessful demonstrations are thrown out. Beyond being successful, motivated by the discussion above, we focus on settings where the demonstrations are heterogenous and differ in reliability and effectiveness. This diversity arises due to variations in operator styles, environmental conditions, or task execution strategies. As a result, some demonstrations may lead to suboptimal behaviors if modeled indiscriminately, posing a challenge for imitation learning methods that aim to model the distribution of demonstrations. Our goal is to learn a maximally successful policy, $\pi_{\text{final}}$, despite varying demonstration quality.

Our problem setting is a case of the autonomous imitation learning (Auto-IL) problem. In autonomous imitation learning [1, 8, 29], an initial policy $\pi_0$ is trained, which is used to generate rollouts. These are processed through a filtering

mechanism, often binary task success/failure, to generate a supplementary dataset. A new policy $\pi_i$ is then trained or fine-tuned using a combination of demos and newly acquired data. The online experience is limited to far fewer trajectories than the typical budget of reinforcement learning methods, making it particularly practical for real robots. In the next section, we will describe how we use this experience to identify how to filter the demonstrations to improve performance.

## IV. Demo-SCORE Data Curation

In this section, we present Demo-SCORE, where we seek to filter out unreliable demonstrations. The key empirical observation behind Demo-SCORE is that successes and failures in policy rollouts can reflect unreliable strategies in the training data that may not be easily discernible from the demonstrations themselves, which are all labeled as successful. By focusing on the outcomes of policy rollouts, Demo-SCORE can identify and filter out unreliable demonstrations that the robot policy cannot replicate successfully and therefore lead to inconsistent policy performance.

Our approach consists of the following main steps: first, we train an initial policy on the full set of demonstrations. We then use the policy to generate rollouts that are used to train a data quality classifier to distinguish between successful and failed outcomes. This classifier is subsequently applied to the original dataset to perform filtering (which can be done at the episode level or the chunk level), retaining only the reliable episodes.

Our method does not require any additional human labeling beyond the detection of success or failure during the rollout phase. This lack of manual dense supervision makes Demo-SCORE scalable to large, diverse demonstration datasets where traditional human annotation would be challenging and costly. Furthermore, grounding demo curation in policy rollouts allows us to identify suboptimal demos that are difficult for human annotators to recognize.

### A. Initial policy and data quality classifier training

We first train an initial policy, $\pi_{0,K}$, on the full set of given demonstrations, $\mathcal{D}_{\text{demo}}$, for a total of $K$ steps. In this step, we prioritize learning a policy capable of modeling the full distribution of demonstrated behaviors while avoiding mode collapse when presented with heterogeneous demonstrations. Recently proposed policy classes such as diffusion policy [12] and ACT [47] have demonstrated robust performance in this context, making them suitable choices for our base policy. This policy is trained until convergence, so that the model learns to replicate the different behaviors shown in the demonstrations.

After the initial training phase, we generate rollouts produced by executing the policy in the environment, which may be rollouts obtained in the process of evaluating the initial policy at various checkpoints. We want to use these rollouts to train a data quality classifier that distinguishes between successful and failed outcomes and therefore hopefully between reliable and unreliable strategies. More formally, let $+$ denote reliable demonstrations and $-$ denote unreliable ones. The learned classifier is expected to provide plausible estimates of demo quality if $p_{\text{rollout}}(\tau|\text{success}) \approx p_{\text{demo}}(\tau|+)$ and $p_{\text{rollout}}(\tau|\text{failure}) \approx p_{\text{demo}}(\tau|-)$, but in practice, we do not have such rollout distributions nor quality labels $+/-$. Thus, one key challenge is that the classifier may overfit to the rollouts it is trained on, which may not perfectly match the original demo distribution. In particular, at convergence time, even though the policy is trained to imitate the full demo distribution and good strategies should be much more reliable than bad ones, the policy rollouts may still contain failures near good strategies and successes near bad strategies. Overfitting to a rollout distribution from one policy checkpoint could lead to poor generalization on the demo distribution.

To address this challenge of distribution shift and potential classifier overfitting, we leverage multiple rollout distributions taken from different checkpoints across the initial training run and use cross-validation. We obtain rollouts at $C$ evenly spaced checkpoints of the initial training run, giving $\{\mathcal{D}_{\pi_{0,i}}\}_{i=1}^{C}$, each taken at checkpoint $i * K/C$. We then train $C-1$ classifiers, $\{q_{\phi_i}\}_{i=1}^{C-1}$, on rollouts from each of the first $C-1$ checkpoints, and perform cross-validation with rollouts from the last checkpoint $\mathcal{D}_{\pi_{0,C}}$ as the validation set. Intuitively, the rollout distribution changes throughout the course of training, typically being broader at the beginning and becoming narrower as the policy fits the original demonstrations. We choose to use $C$ spaced-out checkpoints for several reasons. First, they are spread apart enough to ensure that the rollout distributions are distinct from each other, so a classifier that overfits to rollouts from a checkpoint would be less likely to be chosen through cross-validation. Second, when evaluating the initial training run, it is common to evaluate performance at several evenly spaced checkpoints, so these online rollouts occur little additional online cost beyond evaluating the original training run. Other choices of checkpoints are also effective, as we find in our testing of variations of Demo-SCORE in Section V-D. Furthermore, using as few as 10 rollouts per checkpoint can be sufficient, for a total of less than 100 rollouts.

Each data quality classifier $q_{\phi_i}$ is trained to predict whether a state from the corresponding set of rollouts $\mathcal{D}_{\pi_{0,i}}$ is from a successful or a failed trajectory. The classifiers are trained using a binary cross-entropy loss, where the input consists of the proprioceptive state $s_t$, and the output is a binary label $y_i \in \{0, 1\}$ indicating the outcome of the corresponding rollout. For a single trajectory $\tau = \{(s_t, a_t)\}_{t=1}^{T} \in \mathcal{D}_{\pi_{0,i}}$ with label $y$, the loss at the trajectory level is defined as:

$$L_\phi(\tau, y) = \frac{1}{T}\sum_{t=1}^{T}\left[-y \log q_{\phi_i}(y \mid s_t) - (1-y)\log(1 - q_{\phi_i}(y \mid s_t))\right],$$

where $T$ is the number of timesteps in the trajectory $\tau$.

The overall loss is then the average over all trajectories in the rollout dataset $\mathcal{D}_{\pi_{0,i}}$:

$$\mathcal{L}_\phi = \frac{1}{M}\sum_{(\tau_j, y_j)\in\mathcal{D}_{\pi_{0,i}}} L_\phi(\tau_j, y_j),$$

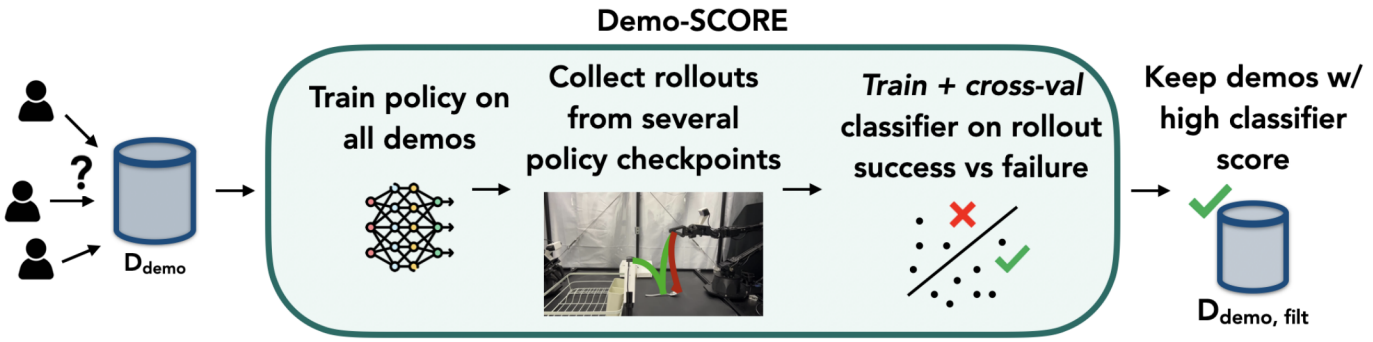


Fig. 2: **Illustration of Demo-SCORE.** Our method self-curates demonstrations through online robot experience in four steps: (i) train a policy on the full demonstration set; (ii) evaluate the policy by generating rollouts with different checkpoints; (iii) use these rollouts to train and cross-validate a classifier that distinguishes between success and failure trajectories; (iv) filter out unreliable demonstrations based on the learned classifier, retaining only ones that do not reflect policy failures for subsequent policy training.

where $M$ is the number of rollouts in the training dataset, $\mathcal{D}_{\pi_{0,i}}$.

The classifiers are small MLPs (with two hidden layers with eight hidden units). We train the classifiers with large weight decay and dropout to prevent overfitting to the rollout distribution, although we find in Section V-D that the classifier is robust even without heavy regularization due to cross-validation. The classifier checkpoint with the lowest validation loss is chosen as the final data quality classifier, $q_{\phi^*}$ and is then used in the next phase to filter the demonstrations.

### *B. Demonstration Filtering*

In the demonstration filtering phase, we apply the learned data quality classifier to the original dataset of demonstrations, in the following manner: Each state $s_t$ from the demonstration is passed as input to the classifier, which returns the probability of success. Averaged across all states in the episode, if the probability of success exceeds a threshold, $\gamma$, the demonstration is kept; otherwise it is discarded. For an episode $\tau = \{(s_t, a_t)\}_{t=1}^{T}$, the average success probability is

$$\bar{q}_{\phi^*}(\tau) = \frac{1}{T}\sum_{t=1}^{T_i} q_{\phi^*}(y = 1 \mid s_t).$$

We use the average probability of success over all states in the rollout training dataset $\mathcal{D}_{\pi_{0,i^*}}$, corresponding to the training set of the best classifier $q_{\phi^*}$, as the threshold, e.g.

$$\gamma = \frac{1}{|\mathcal{D}_{\pi_{0,i^*}}|}\sum_{(s_t)\in\mathcal{D}_{\pi_{0,i^*}}} q_{\phi^*}(y = 1 \mid s_t),$$

and the filtered dataset is then:

$$\mathcal{D}_{\text{demo, filt}} = \{\tau_j \mid \bar{q}_{\phi^*}(\tau_j) > \gamma, \tau_j \in \mathcal{D}_{\text{demo}}\}.$$

$\gamma$ can be tuned further but we find this formulation to work well empirically for all of our experiments in Section V. The probability learned by the classifier of the failures should be much lower than those of successes, which is why $\gamma$ is an effective threshold, even if the quality composition of the dataset is imbalanced, and we find this to be the case in our experiments.

We can then use the filtered dataset $\mathcal{D}_{\text{demo, filt}}$ to train a new policy, $\pi_{\text{final}}$, either from scratch or by fine-tuning the initial policy which was trained on the full demonstration dataset. We then apply our same trained classifier on the successful rollouts themselves, obtaining $\mathcal{D}_{\text{rollouts, filt}}$, and we can then train on the union of $\mathcal{D}_{\text{demo, filt}}$ and $\mathcal{D}_{\text{rollouts, filt}}$ in this last stage. By filtering out unreliable demonstrations, Demo-SCORE helps ensure that the policy learns the most effective strategies, ultimately improving the performance and robustness of the learned robot behavior. Our full method is summarized in Algorithm 1.

**Algorithm 1** Demo-SCORE Summary

**Input:** $\mathcal{D}_{\text{demo}}$, $C$ checkpoints, $M$ rollouts, $K$ steps
$\pi_0 \leftarrow \text{Train}(\mathcal{D}_{\text{demo}})$
$\{\mathcal{D}_{\pi_{0,i}}\}_{i=1}^{C}, \leftarrow \text{GenerateRollouts}(\pi_{0,i*K/C}, M)$
$q_{\phi^*} \leftarrow \text{TrainClassifier, CrossValidate}(\{\mathcal{D}_{\pi_{0,i}}\}_{i=1}^{C})$
$\gamma \leftarrow \frac{1}{|\mathcal{D}_{\pi_{0,i^*}}|}\sum_{(s_t)\in\mathcal{D}_{\pi_{0,i^*}}} q_{\phi^*}(y = 1 \mid s_t)$
$\mathcal{D}_{\text{demo, filt}}, \mathcal{D}_{\text{rollouts, filt}} \leftarrow \{\tau_j \mid \bar{q}_{\phi^*}(\tau_j) > \gamma, \tau_j \in \mathcal{D}_{\text{demo}} \cup \mathcal{D}_{\text{rollouts}}\}$
$\pi_{\text{final}} \leftarrow$ Use $\mathcal{D}_{\text{demo, filt}} \cup \mathcal{D}_{\text{rollouts, filt}}$ to re-train & fine-tune $\pi$
Return $\pi_{\text{final}}$

## V. Experimental Results

In this section, we aim to answer the following empirical questions:

1) Across a range of tasks and heterogenous demonstration mixtures, does Demo-SCORE filter out unreliable demonstrations in a manner that leads to higher policy performance than training on all original demonstrations?
2) How does Demo-SCORE compare to prior methods that leverage online rollouts to improve policy performance?
3) What can the composition of demonstrations filtered out by Demo-SCORE inform us about the quality of the original dataset?
4) How sensitive is Demo-SCORE to method design choices and hyperparameters?

To answer the first two questions, we describe experiments for two simulated settings and then present results from several real-world robotic manipulation tasks with ALOHA. We then analyze our method through ablation studies to answer the last two questions.

### A. Experimental Setup

*a) General Experimental Setup:* We use base learning algorithms Diffusion Policy [12] for all experiments in Robosuite and ACT [48] for simulated and real-world ALOHA experiments. Both of these policy classes have been shown to fit the distribution of heterogeneous demonstrations. We implement all methods on top of state-of-the-art implementations of these policies using the original diffusion policy codebase, original ACT codebase for real-world experiments, and the LeRobot codebase [10] for simulated ALOHA experiments.

*b) Comparisons:* We evaluate Demo-SCORE along with the following prior methods, the last two of which also leverage online rollouts to improve policy performance: (1) Policy trained on all original demonstrations, which we refer to as **Base Policy**; (2) **Training loss** [36], which weights demonstrations inversely proportional to the loss of the policy on the demonstration data. (3) Autonomous imitation learning (**Auto-IL**) [1, 8, 29], where we train the policy on all original demonstrations and successful online rollout episodes; (4) Return-conditioned policies (**RCP**) [22, 35], which trains a policy with an additional input associated with return (here 1 if from a success trajectory, 0 for failure) of the state-action pair on all demos and rollout data; For our main simulated experiments, we use 100 rollouts from $C = 4$ checkpoints for Demo-SCORE, and all 400 rollouts for Auto-IL and RPC baselines. For real-world experiments, we use 20 rollouts per checkpoint for Chocolate and Strawberry tasks and 25 for the Jellybean, Sharpie, and Spoon tasks for $C = 5$ checkpoints to train the classifier. For evaluation, we measure and report success rate and 90% confidence intervals across 256 rollouts in simulated experiments and 30 rollouts in real.

### B. Simulated Experiments in Robosuite and ALOHA

We test Demo-SCORE on a bimanual peg insertion task in the ALOHA [48] environment and a square peg task from Robosuite [27], shown in Figure 3. In the peg insertion task, an ALOHA simulated in MuJoCo picks up a peg and a socket in each hand and inserts the peg into the socket. The observation space for this task consists of joint positions and an overhead RGB camera. In the square peg task, a simulated Panda arm is tasked with picking up a large square-shaped nut with a handle attached and inserting or dropping it on square-shaped peg fixed to the table. Observations for this task consist of proprioception for the robot arm and pose for the square nut.

*a) Demonstration Collection:* For the peg insertion task, we collect demonstrations by scripting three policies (PegA, PegB, PegC) and saving rollouts in which the task was successfully completed. In demos from PegA, the grippers pick up the peg and socket near the center of gravity of both objects with both arms at natural positions. Without changing the orientation of the peg and socket, the arms align the peg and socket and then come together to insert them. In demos from PegB and PegC, when picking up the peg and socket, the positioning of the arms may sometimes obscure the overhead camera's view of the peg and socket during insertion, which may lead to worse policy performance with this strategy.

For the square peg task, we leverage the proficient human-collected dataset given in the benchmark, denoted Human, collected by one human skilled at the task. Additionally, we add demonstrations from two scripted policies: SquareA, in which the gripper grasps the midpoint on the side of the square, moves the nut over the peg, and then drops the nut, SquareB, in which the gripper grasps the nut across the diagonal at the corner of the square and then drops the nut on the peg.

Although they use different strategies, all of the demonstrations successfully complete the task and we do not add any artificial noise. We evaluate our method on a variety of heterogeneous data mixtures taken from a combination of the demonstration sources. In particular, these mixtures include both even and strongly lopsided ratios of different demonstration strategies, in order to represent the varied potential heterogeneity in demonstration datasets.

*b) Results:* In Figure 4, we show the results of Demo-SCORE on the simulated peg insertion and square peg tasks for a variety of data mixtures. We find that on average over all data mixtures in both settings, Demo-SCORE improves upon training on all original demonstrations by 15-35% absolute success rate. With filtering by Demo-SCORE, the policy achieves an average success rate of over 94% on the square peg and over 90% on the peg insertion task across all mixtures. Policies trained with Demo-SCORE have average failure rates that are on average over $2\times$ lower than those of the best prior method (Auto-IL) for the square task and 1.6 times lower for the peg insertion task. Auto-IL is the second-best method, achieving 88.2% average success rate on the square task and 79.9% on the peg insertion task. It improves upon the base policy, as it is trained with more data from successful rollouts, but unreliable demonstrations are still present in the dataset, which still leads to inconsistent policy performance. The Training Loss comparison improves upon training on all original demonstrations for the peg insertion task but not for square peg, suggesting that unreliable strategies may not necessarily correspond to ones that are easiest or fastest to learn. Return-conditioned policies outperforms the base policy on both tasks but performs significantly worse than Demo-SCORE, suggesting that training with the return signal is not as effective in pushing the policy away from unreliable strategies as better curating demonstrations.

Qualitatively, Demo-SCORE filters out between 16% and 67% of demonstrations for the square peg task, depending on the data mixture. We note Demo-SCORE is effective even with strongly lopsided heterogeneous mixtures, such as 200 Human, 20 SquareA or 200 Human, 400 SquareA, showing that Demo-SCORE adjusts accordingly based on the dataset composition.

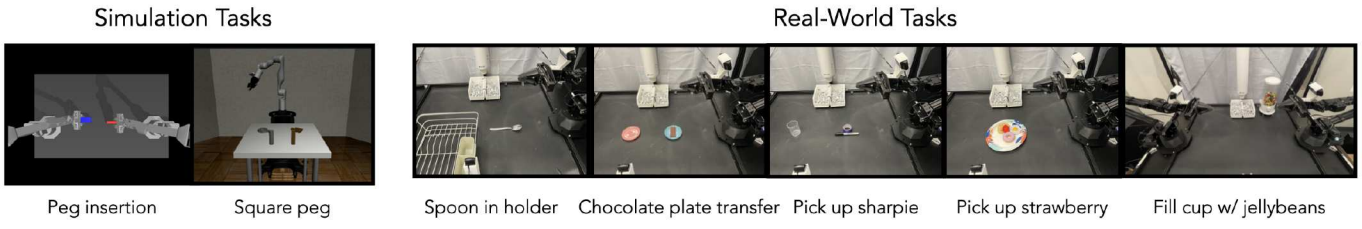


Fig. 3: **Tasks for Evaluation.** In simulation, we evaluate Demo-SCORE on a bimanual peg insertion task in an ALOHA environment and a square peg task from Robosuite. In the real-world, we evaluate on four tasks with ALOHA: spoon in rack cupholder, chocolate plate transfer, sharpie pick up, and strawberry pick up from a cluttered scene.

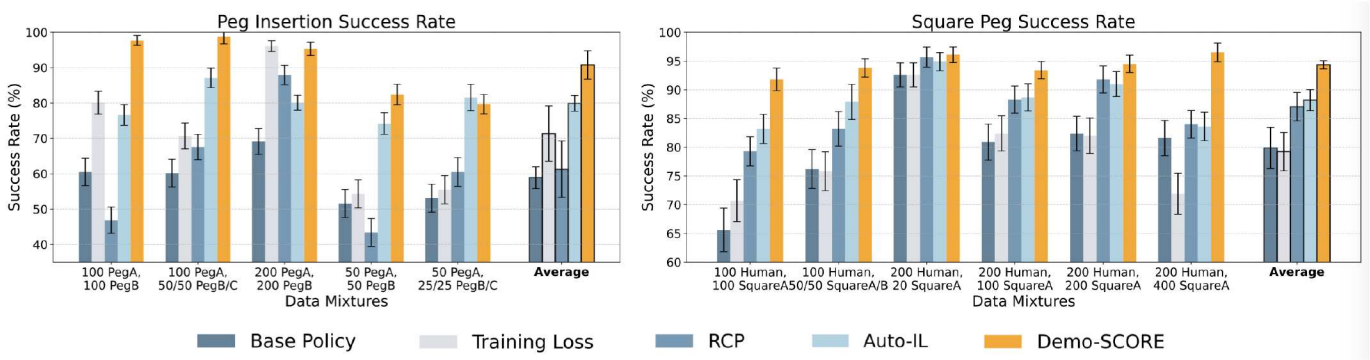


Fig. 4: **Simulated Experiment Results.** We report the success rates and 90% confidence intervals of the policy trained on the filtered demonstrations by Demo-SCORE along with prior methods leveraging online rollouts, across 256 trials for a variety of heterogeneous demonstration data mixtures. For example, in the square peg domain, "100 Human, 50/50 SquareA/B" indicates a demonstration dataset with 100 trajectories from the proficient human dataset, 50 from the SquareA script, and 50 from SquareB. Across both tasks and all data mixtures, Demo-SCORE matches or outperforms all other methods.

### C. Real-world Experiments

We test Demo-SCORE on several tasks requiring precise manipulation with a real-world ALOHA [48] environment, shown in Figure 3: (1) Spoon, where the robot needs to pick up a spoon and put it in the cupholder on a rack; (2) Chocolate, where the robot needs to transfer a chocolate bar from one plate to another; (3) Sharpie, where the robot needs to pick up the sharpie and avoid the adjacent tape roll; (4) Strawberry, where the robot needs to pick up a strawberry from a cluttered breakfast platter; (5) Jellybean, where the robot must take a cup from a dispenser, place it under the jellybean holder, add jellybeans to the cup, and move the filled cup to the front of the table. The base policy trained on all demonstrations for the Jellynean task cannot complete the full task. Therefore, for this task alone, we break the task into five subtasks and use an aggregated subtask score — the fraction of the five subtasks completed in a given episode — for evaluation and classifier training supervision.

*a) Demonstration Collection:* We collect a total of 40 successful demonstrations for the Spoon task, 50 for the Chocolate and Sharpie tasks, and 120 for the Strawberry task. We aim to reflect the diversity of demonstrations found in crowdsourced demonstrations, including a mix of different strategies. In particular, for each of these tasks, we collect demonstrations picking up the object from different grasp positions and angles, encompassing a variety of initial conditions for the objects. For the Jellybean task, we use 124 crowdsourced demonstrations and 100 expert demonstrations from Mirchandani et al. [30].

| | Base Policy | Demo-SCORE |
|---|---|---|
| Retrieved Cup | 70 | 96 |
| Placed Cup | 50 | 64 |
| Moved Cup Below Dispenser | 10 | 44 |
| Filled Cup | 0 | 20 |
| Delivered Filled Cup | 0 | 20 |
| Aggregated Score | 27 | **49** |
| Full Success | 0 | **20** |

TABLE I: We report the percent success rate for each subtask in the Jellybean task, the aggregated score (the percentage of all subtasks completed), and the percent success rate on the full Jellybean task. On this challenging task, Demo-SCORE is able to curate the data so that the robot is able to even achieve success on the full task.

*b) Results:* In Figure 6, we show the results of Demo-SCORE on four real-world ALOHA tasks. Across all four tasks, we find that Demo-SCORE significantly improves upon the base policy, yielding 15-40% absolute improvement in success rate, and almost 30% on average. For the jellybean task, in addition to the aggregated subtask score increasing from 27 to 49 percent, the success rate on the long-horizon full task increases from 0 to 20 percent (details found in Table I). These results underscore the benefits of carefully considering the quality of human-collected demonstrations– fewer demonstrations may learn a more robust policy than

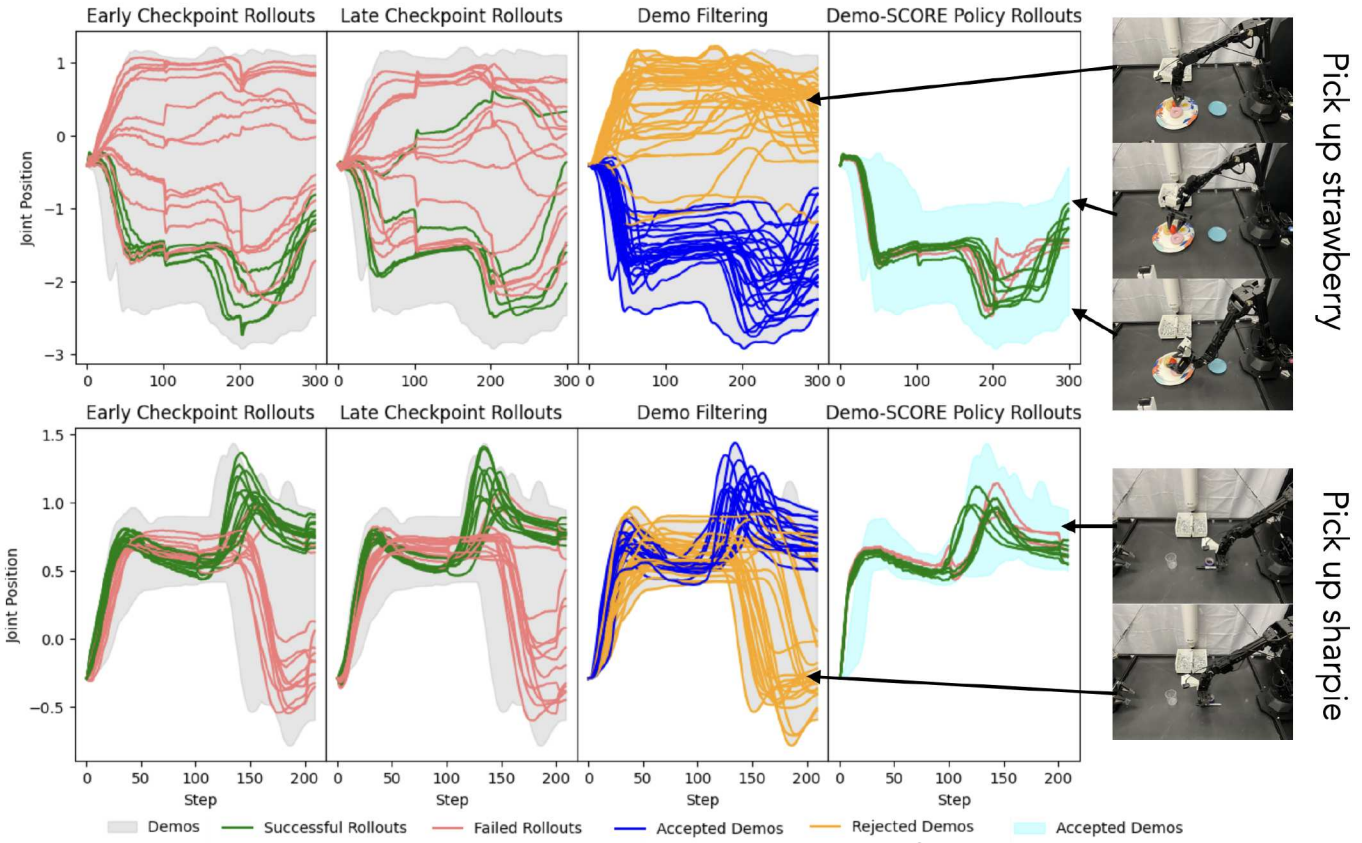


Fig. 5: **Rollout and Demo distributions on real-world tasks.** For the strawberry and sharpie tasks, we show the rollout distributions at early and late checkpoints of the initial policy run, the composition of demonstrations kept by Demo-SCORE compared to the original distribution of demonstrations, and the resulting rollout distribution after filtering. The x axis represents step number in the rollout, and the y axis represents the first PCA component of the joint positions.

| | 100 Human, 100 SquareA | 100 Human, 50/50 SquareA/B | 200 Human, 20 SquareA | 200 Human, 100 SquareA | 200 Human, 200 SquareA | 200 Human, 400 SquareA | Average |
|---|---|---|---|---|---|---|---|
| **Demo-SCORE (original)** | 91.8 | 93.8 | 96.1 | 93.4 | 94.5 | 96.5 | 94.4 |
| Demo-SCORE (chunk) | 90.2 | 91.4 | 94.1 | 96.1 | 95.7 | 93.0 | 93.4 |
| Demo-SCORE (trajectory) | 93.0 | 95.7 | 97.3 | 97.3 | 96.1 | 95.3 | 95.8 |
| Demo-SCORE (plateau checkpoints) | 92.6 | 91.0 | 96.9 | 95.3 | 94.9 | 96.1 | 94.5 |
| Demo-SCORE (no reg) | 94.1 | 91.4 | 92.2 | 92.2 | 96.1 | 95.7 | 93.6 |
| Demo-SCORE (no cross-val) | 93.8 | 89.1 | 87.1 | 84.4 | 81.3 | 95.3 | 88.5 |

TABLE II: **Variations of Demo-SCORE.** We report the success rate of Demo-SCORE with different variations in the Robomimic square peg setting. Demo-SCORE achieves strong performance when filtering at the chunk level instead of episode level, showing that it can be applied at different levels of granularity. Demo-SCORE also performs well using a trajectory classifier instead of a single-step classifier. Using checkpoints near the performance plateau of the initial training run instead of evenly spaced checkpoints is also a good option for training an effective data quality classifier. Finally, we find that cross-validation and regularization are crucial for Demo-SCORE to prevent overfitting to the rollout distribution.

training on all available demonstrations if curated effectively.

In Figure 5, we show visualizations of the rollout distributions and composition of demonstrations that are kept and filtered by Demo-SCORE for the Strawberry and Sharpie tasks, along with the rollout distributions for the policy after filtering with Demo-SCORE. We see that the rollout distributions at different checkpoints differ slightly, which we can leverage for cross-validation. As shown by the red in the visualization of rollouts, some strategies lead to very low success rate when imitated by the learned policy. For the strawberry task, we find that Demo-SCORE tends to filter out demonstrations where the robot approaches the strawberry from above instead of from the side, and for the sharpie task, Demo-SCORE filters out demonstrations where the robot grasps the sharpie by the edge of the cap. These strategies are unreliable for the robot to replicate, as evidenced by the low policy success with that strategy, but may be difficult for human demonstrators or annotators to recognize.

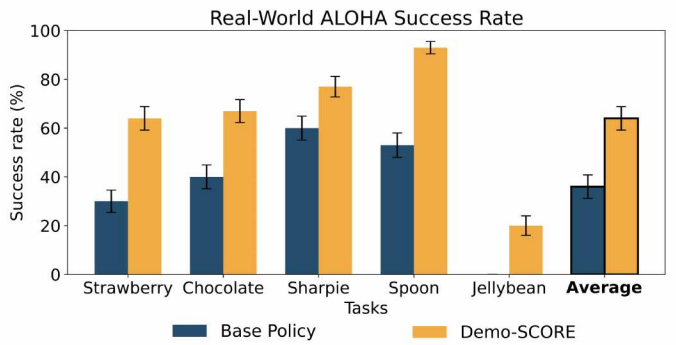


Fig. 6: **Real-World Results with ALOHA.** We report the success rate and 90% confidence interval of the policy trained on the filtered demonstrations by Demo-SCORE compared to training on all original demonstrations across 30 trials. Demo-SCORE improves policy performance almost 30% absolute success rate on average across four real-world tasks.

| | Base Policy | Training Loss | RCP | Auto-IL | Demo-SCORE |
|---|---|---|---|---|---|
| Square peg (100 rollouts) | 79.9 | 79.2 | 87.1 | 88.2 | **94.4** |
| Square peg (50 rollouts) | 79.9 | 79.2 | 85.2 | 83.0 | **92.6** |
| Square peg (25 rollouts) | 79.9 | 79.2 | 83.8 | 83.3 | **93.2** |
| Square peg (10 rollouts) | 79.9 | 79.2 | 81.2 | 81.6 | **92.0** |
| Peg insertion (100 rollouts) | 58.9 | 71.3 | 61.3 | 79.9 | **90.8** |
| Peg insertion (50 rollouts) | 58.9 | 71.3 | 59.6 | 74.4 | **88.0** |
| Peg insertion (25 rollouts) | 58.9 | 71.3 | 56.3 | 70.2 | **89.1** |
| Peg insertion (10 rollouts) | 58.9 | 71.3 | 58.9 | 67.8 | **87.7** |

TABLE III: **Ablation on number of rollouts needed.** On two simulated tasks, Demo-SCORE achieves strong performance using as few as 10 rollouts per checkpoint for classifier training.

### D. *Variations of Demo-SCORE*

In the following, we study particular design choices of Demo-SCORE to understand their impact on policy performance in the simulated Robomimic square peg domain.

*a) Importance of cross-validation using multiple checkpoints:* One key technical challenge is that the rollout distribution may not perfectly match the original demo distribution, and the classifier may overfit to classifying success vs failures on the rollouts it is trained on and fail to generalize to identifying unreliable strategies in the demonstrations. Rollouts from different checkpoints will have slightly different distributions, so we use cross-validation to ensure that the chosen classifier will generalize well to different sets of rollout distributions and therefore better generalize to the demo distribution. Many strategies for choosing these checkpoints may work–instead of taking evenly-spaced checkpoints, another strategy for choosing checkpoints is to use checkpoints near the initial plateau in policy performance where the policy first learns to fit the demos, and cross-validating on rollouts from the best checkpoint (which will occur later in the training run). We find that this strategy is also effective for obtaining a classifier that generalizes well to the original demo distribution, as shown in Table II. Both of these strategies work because the rollout distributions used for training and validating the classifier will be representative of the demonstration data but will vary, so the chosen classifier will not overfit to rollouts produced by a single policy checkpoint. We find in Table II that without cross-validation on sets of rollouts from different checkpoints, Demo-SCORE performs poorly, as the classifier struggles to generalize to the demo distribution.

*b) Number of rollouts needed to train an effective classifier:* In this ablation, we test whether using fewer rollouts, such as 50 or even 10, from the first $C-1$ checkpoints to train the classifier, and only 50 rollouts from the last checkpoint for the validation set, leads to the similar performance for Demo-SCORE. This would be especially appealing for real-world settings where collecting rollouts may be expensive or time-consuming. In Table III, we find that using fewer rollouts leads to only a small decrease in performance, suggesting that only a small number of rollouts are needed to train an effective classifier. We note that other methods that leverage online rollouts have a much larger decrease in performance with fewer rollouts. Notably, with 10 rollouts per checkpoint and fewer than 100 rollouts total, Demo-SCORE achieves 92% average success rate across data mixtures, over 10% higher than the next best method, Auto-IL, on the square peg task. Similarly, with only 10 rollouts per training checkpoint and 80 total rollouts, Demo-SCORE achieves an 87.7 % success rate, compared to a 58.9% success rate for the base policy.

*c) Using a trajectory classifier instead of a single-step classifier:* There are several ways to design the classifier. In our main experiments, we use a single-step classifier that takes in the state as input and outputs a single binary label, and we average the output across all states in the episode to determine whether to keep the demonstration. Here we try using a trajectory classifier (a small transformer) that takes in the entire trajectory as input and outputs a single binary label for the entire trajectory. Because the length of trajectory is more directly correlated with the success of the trajectory than the strategy, we only use the first 100 steps of the trajectory as input to the classifier. In Table II, we find that the trajectory classifier is as effective as the single-step classifier on the simulated square peg task, even giving slightly higher average performance, showing that multiple types of classifiers may be appropriate for Demo-SCORE.

*d) Filtering at the chunk level instead of episode level:* In our main experiments, we filter at the episode level, keeping or discarding the entire demonstration based on the classifier output. However, Demo-SCORE can also be applied at the chunk level, where we keep or discard individual chunks of the demonstrations based on the classifier output. In Table II, we find that filtering at the chunk level is also very effective in our simulated experiments, matching the performance of episode filtering, showing that Demo-SCORE can be applied reliably at different levels of granularity. This is especially appealing for long-horizon tasks, to not discard valuable information in segments of demonstrations.

*e) Importance of classifier regularization:* Finally, we ablate the effect of classifier regularization on the performance of Demo-SCORE. We test the effect of setting weight decay to 1e-4 and removing dropout on the classifier, and find that the performance of Demo-SCORE decreases slightly but is still

comparable when these are removed, as seen in Table II. Thus, the classifier is robust even without heavy regularization, due to cross-validation.

## VI. Limitations

In this paper, we propose Demo-SCORE, which curates demonstration datasets by pruning suboptimal examples estimated from online robot experience. Our experiments on a variety of simulated and real-world tasks show that Demo-SCORE significantly improves the robustness of learned robot policies compared to training on all available demonstrations and to prior methods that leverage online rollouts. While we are excited about the potential of Demo-SCORE, there are several limitations and directions for future work that we discuss as follows. First, Demo-SCORE relies on the assumption that the policy rollouts are representative of the original demonstration distribution, which may not always be the case if the policy is incapable of modeling the full distribution of demonstrated behaviors while avoiding mode collapse. In particular, the policy may not be able to replicate all strategies shown in the demonstrations, and so the policy rollouts may not perfectly match the original demo distribution. Also, Demo-SCORE may not be effective in settings where the demonstrations are all so low quality or sparse that the initial policy is unable to successfully complete the task at all, although we find that we can sometimes address this using subtask scores, as done in the Jellybeans task. Lastly, Demo-SCORE may reduce the coverage of states for which the policy is in distribution. As our goal is instead to maximize final performance on a given task, we no longer aim to maximize state coverage in demonstration curation and instead aim to maximize the reliability of demonstration strategy. Thus, Demo-SCORE is helpful in maximizing final performance, so we recommend using it for post-training and not for pre-training stages where training on wider distributions may be helpful. Our results highlight the importance of high-quality demonstrations and suggest that careful curation can significantly enhance the final performance of learned robot policies.

## Acknowledgments

We thank Suvir Mirchandani, Jonathan Yang, Zach Witzel, Kaylee Burns, and others in the Stanford IRIS lab for support with real-robot experiments. This work was supported in part by an NSF CAREER award and ONR grants N00014-21-1-2685 and N00014-22-1-2621. A.S. Chen acknowledges support by the NSF GRFP and an OpenAI superalignment grant. Y. Liu acknowledges support by an SNSF Postdoc fellowship.

## References

[1] Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, et al. Autort: Embodied foundation models for large scale orchestration of robotic agents. *arXiv preprint arXiv:2401.12963*, 2024.

[2] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5): 469–483, May 2009.

[3] Mark Beliaev, Andy Shih, Stefano Ermon, Dorsa Sadigh, and Ramtin Pedarsani. Imitation Learning by Estimating Expertise of Demonstrators. In *Proceedings of the 39th International Conference on Machine Learning*, pages 1732–1748. PMLR, June 2022.

[4] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. Data Quality in Imitation Learning. *Advances in Neural Information Processing Systems*, 36:80375–80395, December 2023.

[5] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. RoboAgent: Generalization and Efficiency in Robot Manipulation via Semantic Augmentations and Action Chunking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4788–4795, May 2024.

[6] A. Billard, Y. Epars, Gordon Cheng, and S. Schaal. Discovering imitation strategies through categorization of multi-dimensional data. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2398–2403 vol.3, October 2003.

[7] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. $\pi_0$: A Vision-Language-Action Flow Model for General Robot Control, November 2024.

[8] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

[9] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations. In *Proceedings of the 36th International Conference on Machine Learning*, pages 783–792. PMLR, May 2019.

[10] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. https://github.com/huggingface/lerobot, 2024.

[11] Xingchen Cao, Fan-Ming Luo, Junyin Ye, Tian Xu, Zhilong Zhang, and Yang Yu. Limited Preference Aided Imitation Learning from Imperfect Demonstrations. In *Proceedings of the 41st International Conference on Machine Learning*, pages 5584–5607. PMLR, July 2024.

[12] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Dif-

fusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[13] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023. ISBN 978-0-9923747-9-2.

[14] Maximilian Du, Suraj Nair, Dorsa Sadigh, and Chelsea Finn. Behavior Retrieval: Few-Shot Imitation Learning by Querying Unlabeled Datasets. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023. ISBN 978-0-9923747-9-2.

[15] Jensen Gao, Annie Xie, Ted Xiao, Chelsea Finn, and Dorsa Sadigh. Efficient Data Collection for Robotic Manipulation via Compositional Generalization, March 2024.

[16] Siddhant Haldar, Zhuoran Peng, and Lerrel Pinto. BAKU: An Efficient Transformer for Multi-Task Policy Learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, November 2024.

[17] Joey Hejna, Chethan Anand Bhateja, Yichen Jiang, Karl Pertsch, and Dorsa Sadigh. ReMix: Optimizing Data Mixtures for Large Scale Imitation Learning. In *8th Annual Conference on Robot Learning*, September 2024.

[18] Sravan Jayanthi, Letian Chen, Nadya Balabanska, Van Duong, Erik Scarlatescu, Ezra Ameperosa, Zulfiqar Haider Zaidi, Daniel Martin, Taylor Keith Del Matto, Masahiro Ono, and Matthew Gombolay. DROID: Learning from Offline Heterogeneous Demonstrations via Reward-Policy Distillation. In *7th Annual Conference on Robot Learning*, August 2023.

[19] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-Aware Reinforcement Learning for Collision Avoidance. *arXiv:1702.01182 [cs]*, February 2017.

[20] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R. Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J. Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset, March 2024.

[21] Sachit Kuhar, Shuo Cheng, Shivang Chopra, Matthew Bronars, and Danfei Xu. Learning to Discern: Imitating Heterogeneous Human Demonstrations with Preference and Representation Learning. In *Proceedings of The 7th Conference on Robot Learning*, pages 1437–1449. PMLR, December 2023.

[22] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[23] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, 2022.

[24] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior Generation with Latent Actions, March 2024.

[25] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems, November 2020.

[26] Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional Decoding: Improving Action Chunking via Closed-Loop Resampling, December 2024.

[27] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *Proceedings of the 5th Conference on Robot Learning*, pages 1678–1690. PMLR, January 2022.

[28] Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. QueST: Self-Supervised Skill Abstractions for Learning Continuous Control, September 2024.

[29] Suvir Mirchandani, Suneel Belkhale, Joey Hejna, Evelyn Choi, Md Sazzad Islam, and Dorsa Sadigh. So you think you can scale up autonomous robot data collection? *arXiv preprint arXiv:2411.01813*, 2024.

[30] Suvir Mirchandani, David D. Yuan, Kaylee Burns, Md Sazzad Islam, Tony Z. Zhao, Chelsea Finn, and Dorsa Sadigh. RoboCrowd: Scaling Robot Data Collection through Crowdsourcing, November 2024.

[31] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. Learning and Retrieval from Prior Data for Skill-

based Imitation Learning. In *Proceedings of The 6th Conference on Robot Learning*, pages 2181–2204. PMLR, March 2023.

[32] Rohan Paleja and Matthew Gombolay. Heterogeneous learning from demonstration. In *Proceedings of the 14th ACM/IEEE International Conference on Human-Robot Interaction*, HRI '19, pages 730–732, Daegu, Republic of Korea, January 2020. IEEE Press. ISBN 978-1-5386-8555-6.

[33] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. Recent Advances in Robot Learning from Demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(Volume 3, 2020): 297–330, May 2020.

[34] Fumihiro Sasaki and Ryota Yamashina. Behavioral Cloning from Noisy Demonstrations. In *International Conference on Learning Representations*, October 2020.

[35] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[36] Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *International conference on machine learning*, pages 5739–5748. PMLR, 2019.

[37] Voot Tangkaratt, Bo Han, Mohammad Emtiyaz Khan, and Masashi Sugiyama. Variational Imitation Learning with Diverse-quality Demonstrations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9407–9417. PMLR, November 2020.

[38] Daan van Esch, Tamar Lucassen, Sebastian Ruder, Isaac Caswell, and Clara Rivera. Writing system and speaker metadata for 2,800+ language varieties. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 5035–5046, Marseille, France, June 2022. European Language Resources Association. URL https://aclanthology.org/2022.lrec-1.538.

[39] Dian Wang, Stephen Hart, David Surovik, Tarik Kelestemur, Haojie Huang, Haibo Zhao, Mark Yeatman, Jiuguang Wang, Robin Walters, and Robert Platt. Equivariant Diffusion Policy. In *8th Annual Conference on Robot Learning*, September 2024.

[40] Yunke Wang, Chang Xu, Bo Du, and Honglak Lee. Learning to Weight Imperfect Demonstrations. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10961–10970. PMLR, July 2021.

[41] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation Learning from Imperfect Demonstration. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6818–6827. PMLR, May 2019.

[42] Annie Xie, Dylan Losey, Ryan Tolsma, Chelsea Finn, and Dorsa Sadigh. Learning Latent Representations to Influence Multi-Agent Interaction. In *Proceedings of the 2020 Conference on Robot Learning*, pages 575–588. PMLR, October 2021.

[43] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing Data Mixtures Speeds Up Language Model Pretraining. In *Thirty-Seventh Conference on Neural Information Processing Systems*, November 2023.

[44] Haoran Xu, Xianyuan Zhan, Honglei Yin, and Huiling Qin. Discriminator-Weighted Offline Imitation Learning from Suboptimal Demonstrations. In *Proceedings of the 39th International Conference on Machine Learning*, pages 24725–24742. PMLR, June 2022.

[45] Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges. *IEEE Transactions on Cybernetics*, 54(12):7173–7186, December 2024.

[46] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3D Diffusion Policy, March 2024.

[47] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[48] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, April 2023.

[49] Ruijie Zheng, Ching-An Cheng, Hal Daumé Iii, Furong Huang, and Andrey Kolobov. PRISE: LLM-Style Sequence Compression for Learning Temporal Action Abstractions in Control. In *Proceedings of the 41st International Conference on Machine Learning*, pages 61267–61286. PMLR, July 2024.

## APPENDIX

### A. Training Details

*a) Simulated ALOHA domain with ACT policy:* We train every sim ACT policy for 300,000 steps and measure policy success rate with 256 rollouts every 10,000 steps. The reported success rate is the maximum success rate across each of these evaluations. Other than extending training to 300,000 steps, we use the default policy hyperparameters for the LeRobot codebase [10] (which are based on the hyperparameter choices in the original ACT paper). We use precisely the same policy training procedure for initial policies, our method, and comparisons.

*b) Robosuite Domain with Diffusion Policy:* We train all policies for 1,000 epochs. Each evaluation is done with 256 rollouts. The success rate reported for each policy is the maximum success rate in any evaluation. All policy hyperparameters are the same as those in the original Diffusion Policy paper.

### B. Additional Method Details

*a) Demo-SCORE Checkpoint Selection:* For our method, we collect rollouts for use by Demo-SCORE or the baselines using rollouts from epochs 250, 500, 750, and the last checkpoint. For the ALOHA Sim experiments, checkpoints at steps 70,000, 150,000, 220,000, and 300,000 were used to collect rollouts. For the ablation with Demo-SCORE classifier training checkpoints chosen to be at the initial plateau in policy performance, those checkpoints are chosen from the same set of checkpoints used for policy evaluation.

For real-world experiments, upon training the initial policy past the optimal range, one or two rollouts were taken for a wide variety of checkpoints to determine the approximate range of checkpoints where the policy performed best. Within that range, 5 evenly spaced epochs were used for evaluation and rollout collection for each policy trained on that task. Checkpoints with poor performance (success rates less than half that of the best checkpoint's) were discarded.

*b) Demo-SCORE Classifier Training Details:* There are $C-1$ training datasets $D_{\pi_{0,i}}$ for $i = 1, 2, \ldots, C-1$, and we use $C = 4$ for the simulated experiments and $C = 5$ for real-world experiments. For each of these, a classifier was trained. For each classifier training run, the best checkpoint was chosen using our cross-validation metric (the unweighted binary-cross entropy loss of our classifier on the cross-validation set, $D_{\pi_{0,C}}$). Then, of the classifier runs, the best was chosen with the same metric. When training step classifiers, the loss was computed for each step in the dataset, and then averaged. For the trajectory classifier variation, the loss was computed across trajectories in the dataset and then averaged. For the ablations without cross-validation, a total of 200 rollouts from the same policy checkpoint (the best performing one) were used. For each training run, these were randomly divided into a set of 100 train and 100 val rollouts. All classifiers use dropout of 0.3 and AdamW weight decay of 0.1.

### C. Additional details on comparisons

*a) Training Loss method description:* To compute the training loss used in the training loss weighting baseline, we compute the loss using the policy from the initial training run using all demos (the best checkpoint) for every example drawn from a given episode in the set of original demos. We then take the mean across all examples for a given episode to get a mean training loss for that episode. For diffusion policy, we do this eight times for each episode and take the mean due to stochasticity in the diffusion policy forward pass. We then associate a weight with each episode in the demo set by taking the inverse of the mean loss for that episode. We normalize this set of weights by subtracting the minimum value and dividing by the standard deviation. Having calculated these episode-wise weights, we train the policy again, multiplying the loss for each example by the weight associated with the episode that example came from.

### D. Additional Simulated Result Tables

We provide the number of demonstrations filtered out by Demo-SCORE in Table IV for the square peg task, showing that Demo-SCORE adjusts the amount of demos filtered depending on the quality mixture of the original demonstrations. In Table, V, we show that the performance of DemoSCORE on the square task is consistent as we vary the number of hidden units and layers in the step classifier MLP.

### E. Additional Details on Jellybean Task

The subtasks of the jellybean task are:

- retrieving the cup from the cup dispenser
- placing the cup on the table next to the jellybeans dispenser
- pushing the cup underneath the spout of the jellybean dispenser
- filling the cup with jellybeans
- moving the filled cup to the front of the table

We provide success rates by subtask for the Jellybean task in Table I. Additionally, we define an aggreated subtask score as the fraction of subtasks completed in an episode. For example, if the robot completed the first 3 of 5 subtasks in a given episode, that episode would get a score of 60%. As the base policy trained on all demonstrations did not ever succeed on the full task, this aggregated subtask score was used as the target label to supervise classifier training for this task.

### F. Impact of Demo-SCORE Filtering on OOD Performance

One potential concern is that Demo-SCORE narrows the state distribution in the original dataset, which may affect generalization capabilities. To understand this, we evaluate two generalization settings in the square peg domain: (1) OOD initial conditions, and (2) different train-test splits across in-task variations of different initial conditions.

For (1), in Table VI, we show the success rate of the square task policies trained on each data split within a modified environment where the xy dimensions of the region where the square nut is initialized are expanded by 50% (easy) and 100% (hard) beyond the standard environment. As the demos used

| | 100 Human, 100 SquareA | 100 Human, 50/50 SquareA/B | 200 Human, 20 SquareA | 200 Human, 100 SquareA | 200 Human, 200 SquareA | 200 Human, 400 SquareA |
|---|---|---|---|---|---|---|
| Num Demos Filtered Out | 104 | 93 | 35 | 142 | 200 | 400 |

TABLE IV: **Number of demonstrations filtered by Demo-SCORE.** Demo-SCORE is effective even with strongly lopsided heterogeneous mixtures and adjusts accordingly depending on how much should be filtered from a given dataset.

| Classifier MLP Size | Base Policy | [8, 8] | [8, 8, 8] | [16, 16] | [16, 16, 16] | [32, 32] | [32, 32, 32] |
|---|---|---|---|---|---|---|---|
| Success Rate | 79.9 | 94.4 | 95.8 | 95.3 | 94.4 | 94.0 | 95.1 |

TABLE V: **Ablation on Classifier Size**. For the square task, we report the average percent success rate for the across data splits when using Demo-SCORE with step MLP classifiers of different sizes. For each MLP architecture, we report the number of hidden units in each hidden MLP layer and the average success rate of the final Demo-SCORE policies. The main experiments used an "[8, 8]" architecture with two hidden layers with 8 units each.

to train these policies were all from the original environment, the nut's initial position is OOD in these expanded regions relative to the datasets used to train these policies. We find that Demo-SCORE (with either full episode filtering or chunk-level filtering) does not harm OOD generalization in this setting compared to the base policy or Auto-IL and even improves performance whe the environment is initialized in these OOD states.

For (2), our goal is to test the test the policy's generalization ability to initial conditions that are both in-distribution and OOD but have mostly or only suboptimal data in some regions. Specifically, we train new policies on new datasets in which there is a correlation between the initial state of the nut and the strategy used in each demo. We divide the initial state space of the nut by the orientation of the nut, defining "even" and "odd" regions corresponding to the quadrant in which the angle of rotation around the z-axis of the nut falls. (In other words, if the angle by which the nut is rotated around the z-axis relative to its reference position falls between 0 and 90 degrees or 180 and 270 degrees, it is considered within the "odd" quadrant region).

In these new datasets, we control how many Human and Square A demos there are in which the nut's initial orientation is Odd or Even. Each dataset has 200 total demonstrations, 100 for the odd and even region. In the Even region, we vary the percentage of demos that are on the Square A mode from 100% to 50%, in increments of 10%. We either chose the opposite ratio of modes in the Odd region or keep 50 demos from each modes in the odd region. We present the success rates of the policies trained on these datasets in Table VII.

Demo-SCORE improves performance when different ratios of high-quality and suboptimal demonstrations exist for different initial conditions and does not degrade performance for states dominated by suboptimal demonstrations, suggesting that Demo-SCORE does not filter out suboptimal data that contributes to generalization success. However, in some of the most extreme cases (such as one data split where 100% of demos in the even region of initial state space have the Square A strategy), Demo-SCORE can slightly under-perform other methods. Nonetheless, these results suggest that Demo-SCORE generally does not excessively narrow the state distribution in a way that harms policy generalization. Intuitively, Demo-SCORE aims to keep any data that contributes to policy success and therefore may retain suboptimal strategies if they appear in successful rollouts. Additionally, if there are settings where filtering may be overly restrictive, this can be mitigated by common alternatives, e.g. reweighting. Our key contribution is an approach for estimating the quality of a demo or chunk, and this score can be applied in various ways, including but not limited to filtering.

| Data Split | 50% Expanded XY | | | | 100% Expanded XY | | | |
|---|---|---|---|---|---|---|---|---|
| | Base Policy | Auto-IL | Demo-SCORE | Demo-SCORE (chunk) | Base Policy | Auto-IL | Demo-SCORE | Demo-SCORE (chunk) |
| 100 Human 100 SquareA | 65.2 | 71.9 | 85.9 | 85.9 | 51.6 | 63.3 | 73 | 72.3 |
| 100 Human, 50/50 Square A/B | 66.4 | 77.7 | 88.7 | 85.2 | 54.3 | 64.8 | 74.2 | 72.3 |
| 200 Human, 20 Square A | 89.8 | 90.6 | 92.6 | 92.2 | 80.1 | 80.9 | 83.2 | 82 |
| 200 Human, 100 Square A | 73.4 | 84.8 | 89.1 | 90.6 | 64.5 | 75.4 | 81.3 | 82 |
| 200 Human, 200 Square A | 75.4 | 87.1 | 93 | 94.5 | 61.7 | 82.4 | 83.4 | 80.5 |
| 200 Human, 400 Square A | 68 | 76.2 | 94.1 | 94.9 | 54.7 | 65.6 | 83.6 | 84.4 |
| Average | 73.0 | 81.4 | 90.6 | 90.6 | 61.2 | 72.2 | 79.8 | 78.9 |

TABLE VI: **OOD Evaluation of Square Task Policies**. The success rates of the base policies, policies trained with Auto-IL, and two variants of Demo-SCORE, on OOD environments. The xy dimensions of the region in which the square nut's initial position is sampled (uniformly) are expanded by 50% and 100% relative to the original environment. The demonstrations used to train the evaluated policies come from the original environment, and these policies are thus OOD on the modified environments. Demo-SCORE outperforms the base policy and the Auto-IL policy when the environment is initialized to OOD states.

| Data Mixture | Standard Environment | | | 50% Expanded XY | | | 100% Expanded XY | | |
|---|---|---|---|---|---|---|---|---|---|
| | Base Policy | Auto-IL | Demo-SCORE | Base Policy | Auto-IL | Demo-SCORE | Base Policy | Auto-IL | Demo-SCORE |
| 100% SqA Even, 100% Human Odd | 54.3 | 62.9 | 69.9 | 57.0 | 56.2 | 63.7 | 46.9 | 49.6 | 51.2 |
| 90% SqA Even, 90% Human Odd | 59.8 | 63.3 | 78.1 | 57.4 | 60.5 | 71.5 | 50.4 | 52.3 | 62.9 |
| 80% SqA Even, 80% Human Odd | 66.4 | 73.4 | 86.7 | 62.9 | 68.4 | 80.9 | 53.9 | 56.6 | 68.4 |
| 70% SqA Even, 70% Human Odd | 69.5 | 83.6 | 91.4 | 68.8 | 75.4 | 89.8 | 62.5 | 60.5 | 78.1 |
| 60% SqA Even, 60% Human Odd | 67.2 | 78.1 | 88.7 | 64.8 | 77.0 | 86.3 | 57.0 | 67.6 | 70.7 |
| 50% SqA Even, 50% Human Odd | 67.6 | 76.2 | 91.4 | 65.2 | 71.9 | 85.5 | 57.0 | 62.1 | 75.4 |
| 100% SqA Even, 50% Human Odd | 45.7 | 56.2 | 52.3 | 43.8 | 49.6 | 51.2 | 43.8 | 40.6 | 39.8 |
| 90% SqA Even, 50% Human Odd | 48.8 | 60.5 | 69.9 | 44.5 | 61.3 | 65.6 | 41.8 | 48.4 | 52.0 |
| 80% SqA Even, 50% Human Odd | 61.7 | 68.0 | 85.2 | 60.9 | 64.8 | 79.7 | 53.1 | 53.9 | 66.4 |
| 70% SqA Even, 50% Human Odd | 63.3 | 69.5 | 78.1 | 59.8 | 64.5 | 75.8 | 53.1 | 58.6 | 64.1 |
| 60% SqA Even, 50% Human Odd | 67.6 | 76.2 | 89.8 | 65.2 | 77.0 | 86.3 | 59.8 | 69.1 | 71.1 |

TABLE VII: **Success Rates on Correlated Datasets**. The success rates on the square task of base policies, Auto-IL, and Demo-SCORE policies when the original data mixture has a correlation between the initial orientation of the square nut and the strategy exhibited in each demo as described in section F. Each row corresponds to an initial data mixture and is denoted by the percentage of demos within the even region that are employ the Square A strategy and percentage of demos within the odd region that are Human demos (which employ the same strategy). We present evaluations for the standard environment, as well as OOD evaluations on a modified environments where the square nut's initial xy position is initialized within a region expanded 50 or 100% relative the standard environment. Demo-SCORE generally outperforms the base policy and Auto-IL even when there is a correlation between initial state and demonstrated strategy *and* the evaluation environment is OOD.