Extended Abstract

Motivation Imitation learning from human demonstrations is hindered by variable demonstration quality and operator strategy. Demo-SCORE (Chen et al. (2025)) addresses this using binary success/failure signals from robot rollouts to hard-filter demonstrations below a threshold. However, binarization discards useful signal: a demonstration scoring 0.4 is treated identically to one scoring 0.05, and the threshold is a sensitive hyperparameter. We instead use continuous score-weighted BC with thresholding, where a learned trajectory scorer assigns each demonstration a quality score using an inverse success-time label, which directly weights its loss contribution. We additionally experiment with exponentiated labels, which we hypothesized amplifies the difference between fast and slow demonstrations. We evaluate across threshold values and validate generalization on a second task.

Method We propose continuous score-weighted behavioral cloning as an alternative to binary demonstration filtering. A trajectory quality scorer (MLP) is trained on BC rollout data using inverse time-to-success labels: score $= 1/(t_{\text{success}} + 1)$ for successful trajectories and 0 for failures, encoding the intuition that faster success indicates higher quality. At BC training time, the frozen scorer makes a single pass over all demonstrations to precompute one quality score per demo, which then scales each sample's loss contribution, normalized to mean 1 per batch to preserve loss magnitude.

Implementation We evaluate four experimental conditions. First, continuous reweighting without filtering. Second, reweighting combined with thresholding, where demos scoring below τ are dropped entirely ($\tau \in 0.0025, 0.0035, 0.3$). Third, exponentiated weighting, where normalized weights are raised to a power $k > 1$ to sharpen discrimination between high- and low-quality demos, again combined with thresholding ($\tau \in 0.1, \ldots, 0.8$). Finally, we evaluate generalization by rerunning on a second task (Lift). As an offline baseline, we additionally train an operator skill classifier directly on the ground-truth MH tier labels, requiring no rollout data, to quantify the cost of distribution mismatch in our rollout-based scorers.

Results Continuous reweighting without thresholding ($\tau = 0.0$) yields only marginal gains over vanilla BC, converging to ≈0.184 versus the 0.155 baseline. Introducing thresholding produces meaningful improvements: at $\tau = 0.0025$, the policy peaks at 0.303 mean success rate and converges to 0.24–0.25, with the policy learning faster than without thresholding. Exponentiated weighting does not substantially improve over the best inverse-threshold configuration, though it consistently outperforms the binary classification baseline. We suspect the batch-level weight normalization step dampens the intended amplification effect, partially canceling the gains from exponentiation. On the Lift task, our approach generalizes well, achieving ≈80% mean success rate compared to ≈30% for binary filtration, demonstrating that continuous score-weighted BC transfers across tasks.

Discussion Our results show that continuous reweighting alone is insufficient without thresholding as it shows marginal differences from vanilla BC. The operator skill classifier, which scores demos from the same distribution it was trained on, outperforms the rollout-based inverse scorer without thresholding, potentially showing the scorer's training on rollout data could have underlying distribution mismatch. Exponentiated weighting does not substantially improve over the best inverse-threshold configuration, likely because batch-level normalization dampens the intended amplification effect.

Conclusion We introduced continuous score-weighted BC as a replacement for binary demonstration filtering, achieving a 15% absolute improvement over vanilla BC and ≈9% over binary filtration on the Can task. Gains are primarily driven by thresholding rather than soft reweighting alone, and our approach generalizes to the Lift task (≈80% vs. ≈30% for binary filtration). Future work should evaluate on more complex tasks, explore alternative quality labels beyond time-to-success, and train the scorer on higher-quality rollout data to reduce distribution mismatch.

Learning from Heterogeneous Data

Sydney Yan
Department of Computer Science
Stanford University
syyan@stanford.edu

Evy Shen
Department of Symbolic Systems
Stanford University
evyshen@stanford.edu

Tracy Wei
Department of Symbolic Systems
Stanford University
tracywei@stanford.edu

Abstract

Imitation learning from human demonstrations is hindered by variable demonstration quality and operator strategy. Demo-SCORE (Chen et al. (2025)) addresses this using binary success/failure signals from robot rollouts to hard-filter demonstrations below a threshold. However, binarization discards useful signal: a demonstration scoring 0.4 is treated identically to one scoring 0.05, and the threshold is a sensitive hyperparameter. We instead use continuous score-weighted BC with thresholding, where a learned trajectory scorer assigns each demonstration a quality score using an inverse success-time label, which directly weights its loss contribution. We additionally experiment with exponentiated labels, which we hypothesized amplifies the difference between fast and slow demonstrations. We evaluate across threshold values and validate generalization on a second task.

1 Introduction

Like many other machine learning methods, imitation learning policy success and robustness is heavily dependent on the input data. Unlike data for image recognition or text prediction, however, robot training data is extremely expensive to obtain. Better approaches for learning from heterogeneous datasets could allow robots to learn from crowdsourced datasets with multiple operators of varying abilities, greatly expanding the amount of data available for training. Robomimic (Mandlekar et al. (2021)) is one prior work that investigates training on heterogeneous datasets. They curate demonstration datasets that vary significantly in operator proficiency, behavioral strategy, and task coverage, and prior works show that filtration mechanisms can improve the effective quality of the distribution the policy imitates. Demo-SCORE (Chen et al. (2025)) trains a classifier on simulated trajectory rollouts to identify and filter out unsuccessful trajectories, achieving a 15-35% higher success rates over unfiltered baselines. Our work builds on this and asks whether binary filtering can be replaced with a continuous alternative. This approach seeks to use expected quality preferentially guide BC while still maintaining a larger, more diverse dataset.

2 Related Work

In robot imitation learning, data quality negatively impacts policy learning. Policies learned via behavioral cloning (BC) suffer from compounding errors at test time due to state distribution shift: small action errors accumulate, driving the robot into states not covered by training data, Belkhale et al. (2023). Instead of focusing on algorithmic-driven solutions, a growing body of work tackles it

from a data filtering perspective, or in other worfs, selecting or reweighting demonstrations before training to improve the effective quality of the distribution the policy imitates.

Real-world demonstration datasets are inherently heterogeneous, where trajectories vary simultaneously in operator proficiency, behavioral strategy, action noise, and task coverage. Mandlekar et al. (2021)'s robomimic work establishes an empirical framework for such practice, motivating the need for filtering beyond simple success/failure labeling.

Several works propose task-agnostic, offline metrics for scoring demonstrations without requiring environment interaction. Hejna et al. (2025) introduce DemInf, which estimates each trajectory's contribution to the mutual information between states and actions in the dataset. Complementarily, Sirigiri et al. (2026)'s FAKTUAL curates demonstrations by maximizing trajectory entropy over a kernel-based similarity measure, selecting diverse subsets without access to the policy or rollouts. While effective, these purely offline metrics measure proxies for quality rather than directly optimizing downstream task success.

Chen et al. (2025)'s Demo-SCORE work explores the integration of online experience in this traditionally offline approach by training a binary data quality classifier to differentiate success/failure outcomes from robot rollouts and then applying this as a hard-filter. This approach achieves 15–35% higher absolute success rates over unfiltered baselines. However, the binarization of classifier scores discards potentially useful gradient signal: a demonstration scoring 0.4 is treated identically to one scoring 0.05. Furthermore, the filtering threshold is a sensitive hyperparameter. Our work directly addresses this limitation by replacing hard filtering with a continuous score-weighted training objective, preserving mid-quality demonstrations' contribution to learning while reducing the influence of unreliable ones.

3 Method

3.1 Data

In our approach, we utilize both an open-source, heterogeneous robotics dataset and rollout data that we generate using the former. Our primary dataset is robomimic v0.1's Multi-Human (MH) split on the Can task, which is heterogeneous in nature due to varied proficiencies of 6 operators, as there are "2 "worse" operators, 2 "okay" operators, and 2 "better" operators, resulting in diverse, mixed quality datasets" (Mandlekar et al. (2021)). Upon manually examining the dataset to validate the heterogeneity, we find there is variation in factors like the operator's gripper angle, gripper tip position, joint positions, and such.

We then train two policies on this dataset using two imitation learning algorithms: BC (Behavioral Cloning) and Diffusion Policy. Each algorithm's trained policy then generated 250 rollout trajectories, resulting in a total of 500. We then evaluate both policies and find that BC achieves a success rate of 15.5% while Diffusion Policy achieves 85.2%. Since the BC policy produced a significantly lower success rate, we decide to focus on the rollout data to train our proposed scorers.

3.1.1 Dataset Heterogeneity

To characterize the heterogeneity of the MH Can and Lift datasets, we analyze trajectory length (Figure 1). *Worse* operators tend to produce shorter trajectories concentrated below 200 timesteps, many corresponding to early failures before the can is grasped, whereas *better* operators show a broader, right-skewed distribution reflecting more deliberate executions; failed or abandoned demonstrations are thus systematically shorter. The Can dataset exhibits substantially longer and more variable trajectories (98–1050 steps) than Lift (41–303 steps), reflecting its greater complexity. In both, tier separation is consistent: *better* operators finish fastest and *worse* operators slowest. However, the gap is more pronounced in Can, with *worse* operators showing a heavy tail past 600 steps; Lift's distributions are tighter, suggesting even low-quality operators complete the simpler task without getting severely stuck.

3.1.2 Baselines

For our baselines, we started with standard vanilla behavioral cloning with no filtration. We also utilized with discrete classifiers as a baseline. A Demo-SCORE binary classifier was implemented as a

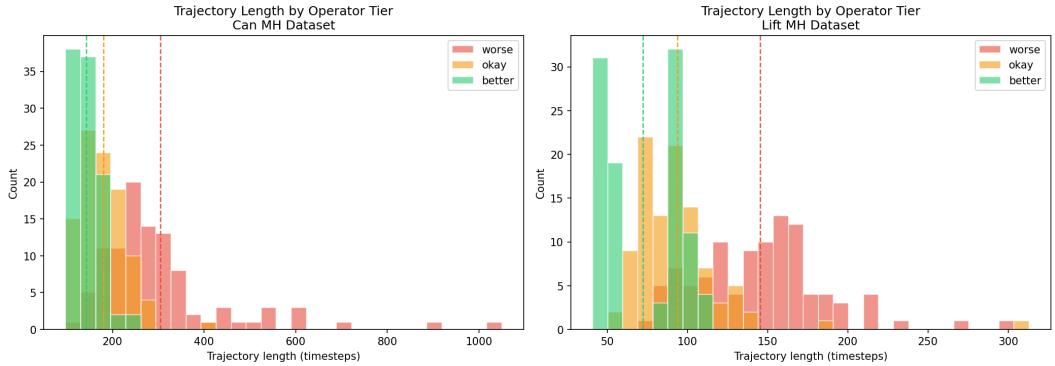

Figure 1: Dataset analysis of the MH Can dataset. Trajectory length distributions by operator tier. *worse* operators produce more short, failed trajectories. at a given score threshold. **Left**: Can dataset. **Right**: Lift dataset.

baseline, where trajectories that were classified 0 were filtered out. Like Demo-SCORE, this classifier was trained using both rollout data and the input expert dataset, and was trained on predicting rollout success labels. Trajectories that looked more similar to failed rollouts were filtered out. We observed that filtration using the binary classifier outperformed standard vanilla BC.

3.2 Trajectory Quality-Weighted BC

Here, we wanted a method to upweight high-quality demos and discard low-quality ones during BC training, using a learned trajectory scorer to determine demo quality. In Demo-SCORE, the binary 0 1 classification was trained using whether the trajectory successfully completed a task or not (a can was dropped into a certain bin, or lifted to a certain height). This was approach was effective and simple, as these tasks lended themselves to binarization. Since we wanted to generate a continuous score, we could not train against binary success. We noticed that within the 300 trajectories, the mean length differed significantly between each category of operator (Table 1): mean length is 142.6 timesteps for *better* operators, 180.5 for *okay*, and 304.4 for *worse*, with length negatively correlated with operator skill. But while the correlation coefficient is low, we note that there is a large variation in trajectory length despite only have three distinct labels. This motivated us to investigate trajectory length, or the number of timesteps before success, as a continuous signal to use as the basis for our scorer.

Thus, our scoring function aimed to a quality score dependent on the number of timesteps needed for this trajectory to be successfully. Notably, it was not practical to use a similar

Table 1: Trajectory length by operator-quality tier in the multi-human (MH) Can dataset. Worse demonstrations are substantially longer and more variable than better ones. Length is negatively correlated with quality label (Pearson $r = -0.58$; Spearman $\rho = -0.72$, $p < 10^{-48}$).

Tier	Label	n	Mean	Median	Std	Min	Max
Worse	0.0	100	304.4	270.5	147.9	123	1050
Okay	0.5	100	180.5	176.0	47.0	110	398
Better	1.0	100	142.6	136.0	29.5	98	236

3.2.1 Inverse Scoring

Each trajectory is assigned a continuous, inverse label: $\texttt{score} = 1/(\texttt{t\_success}+1)$, $\texttt{score} = 0$ for failure, where `t_success` is the number of time steps until reward. We use this inverse to design shorter trajectories to have higher scores and longer trajectories to have lower scores, with the assumption that faster is better, while accounting for how failed trajectories should be represented in such a continuous time-to-success-based label.

The inverse form

$$\text{score} = \frac{1}{t_{\text{success}} + 1}, \qquad \text{score} = 0 \text{ for failure} \tag{1}$$

was chosen for several reasons. First, it is monotonically decreasing in t_{success}, encoding the intuition that faster success is strictly better. Second, it is bounded in $(0, 1]$, making it directly usable as a normalized weight without additional rescaling. Third, the $+1$ offset prevents singularities when $t_{\text{success}} = 0$ and compresses the score range in a way that avoids extreme weight disparities between only marginally different trajectories. Failed trajectories, which never receive reward, are assigned a score of 0, cleanly representing their lack of task completion while remaining expressible in the same continuous space as successful ones.

Thresholding We realize that although a continuous signal would help us learn from partially good trajectories, it may still be better to filter out some trajectories completely. Many subpar trajectories will receive a nonzero score from our classifier, and while our goal is to allow the model to learn from suboptimal but useful trajectories, there are suboptimal trajectories that may not be useful to us. Therefore, we incorporate thresholding in our approach, filtering out demo data scored less than a certain threshold τ. The inverse label score was still used to weight trajectories with values above τ. We vary τ across different values for the following rationales: 0.0025 (to drop scores below mean), 0.0035 (to keep scores above mean & consider the top 40% of demos), and 0.3 (for a more aggressive threshold).

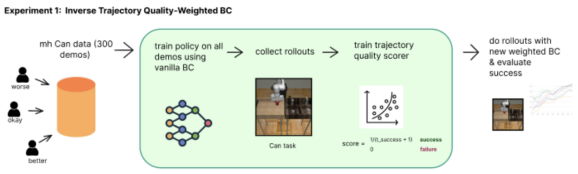

Figure 2: Pipeline for Inverse Trajectory Quality-Weighted BC (no-threshold)

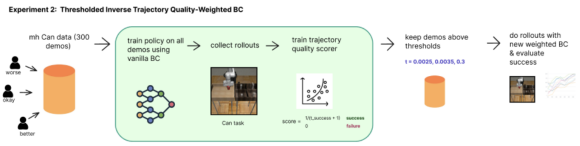

Figure 3: Pipeline for Inverse Trajectory Quality-Weighted BC (thresholded)

3.2.2 Exponentiated Weights

Inverse scoring provides a natural quality signal, but the resulting weight distribution may be insufficiently discriminative. Trajectories of moderate quality receive weights that are not much lower than those of the best demonstrations, limiting the degree to which the training objective is concentrated on high-quality behavior. To sharpen this discrimination, we consider exponentiating the normalized weights

$$\tilde{w}_i = \frac{w_i^k}{\sum_j w_j^k} \tag{2}$$

which amplifies differences between high- and low-scoring trajectories and pushes the effective weight distribution toward a more peaked form centered on the top demonstrations.

Exponentiation has a well-known interpretation in the context of softmax and energy-based weighting. Raising weights to a power $k > 1$ increases the relative influence of high-scoring examples while further suppressing low-scoring ones. This provides a principled way to interpolate between uniform weighting ($k = 1$, flat distribution) and hard filtering ($k \to \infty$, winner-takes-all), with k acting as a continuous knob to control selectivity.

Thresholding Adopting the same rationale as above, we experiment with different levels of thresholding after weights are normalized. We vary τ across the values: 0.1, 0.2, 0.3, .. 0.8.

3.2.3 Operator Skill-Weighted BC (Baseline)

As a reference point for our rollout-based scoring approaches, we also evaluate a quality-weighted BC policy whose weights come directly from the ground-truth operator skill labels provided in the MH dataset, rather than from a learned scorer trained on rollouts. We train a 3-class trajectory-level classifier on the MH demonstrations to predict operator tier (*worse*, *okay*, *better*) using the ground-truth `mask/` group annotations, and convert its output to a scalar weight $w = 0 \cdot P(\text{worse}) + 0.5 \cdot P(\text{okay}) + 1.0 \cdot P(\text{better})$.

This approach requires no rollout collection and introduces no distribution mismatch, since the classifier is trained and applied on the same MH dataset, which aims to represent a strong offline baseline. Performance above this baseline would suggest our rollout-based scorers are capturing useful signal beyond what operator labels alone provide.

4 Experimental Setup

4.1 Training Scorers

We train our scorers on the rollout data from the BC policy, splitting each trajectory into 10 temporal chunks and sampling one timestep per chunk with the following training configurations: epochs=2,000, batch size=100, 2-layer MLP [1024, 1024]. We evaluated via 50 rollouts every 50 epochs, with a horizon of 400 steps.

4.2 Training the Operator Skill Classifier

We train the 3-class trajectory-level classifier on the can multi-human (MH) dataset to predict operator skill tier: *worse* (0), *okay* (1), or *better* (2). Labels are taken directly from the ground-truth `mask/` groups in the HDF5 file.

Each demonstration is represented as a fixed-length feature vector by splitting the trajectory into $N{=}5$ equal temporal chunks and computing the per-chunk mean and standard deviation across all 10 observation dimensions (end-effector position and orientation, gripper joint positions and velocities, joint positions, and object pose), yielding a 570-dimensional input ($5 \times 2 \times 57$).

We train a 3-layer MLP [256, 128, 64] with BatchNorm, ReLU, and Dropout ($p{=}0.3$) at each hidden layer using CrossEntropyLoss, AdamW (lr$=10^{-3}$, weight decay 10^{-4}), and `ReduceLROnPlateau` (patience=5, factor=0.5). We train for 80 epochs with batch size 32.

At BC training time, the frozen classifier scores each demo via a single forward pass, converting the 3-class softmax output to a scalar quality weight: $w = 0 \cdot P(\text{worse}) + 0.5 \cdot P(\text{okay}) + 1.0 \cdot P(\text{better})$.

4.3 Training Trajectory-Quality BC

We train a deterministic BC policy using a 2-layer MLP [1024, 1024] on low-dimensional state observations (end-effector position and orientation, gripper joint positions, and object pose).

To incorporate demonstration quality, we reweight the per-sample loss by the frozen trajectory scorer. At the start of the BC training, the scorer runs a single deterministic pass over all 500 demos to precompute one quality score per demo. Each training demo inherits parent demo's score. Per-sample losses are then scaled by these quality weights, normalized to mean 1 within each batch to keep the loss scale comparable to vanilla BC. We train for 2,000 epochs with a batch size of 100, evaluating via 50 rollouts every 50 epochs with a horizon of 400 steps.

4.3.1 Scorer Discriminability

Before training, we examine how well each scorer separates operator tiers across thresholds. Figure 4 shows the fraction of demos retained per tier as τ increases for both scorers.

The inverse scorer (left) exhibits clear tier ordering: *worse* demos are filtered out earliest, *okay* next, and *better* last. At the mean score ($\tau \approx 0.705$), roughly 50% of *worse* demos are already dropped while most *better* demos remain, confirming the scorer assigns meaningfully higher weights to higher-quality demonstrations.

The exponentiated scorer (right) shows almost no tier separation. All three curves fall together across the full threshold range. This could mean the scorer does not reliably discriminate between operator tiers, which helps explain why its BC performance is less sensitive to the choice of τ.

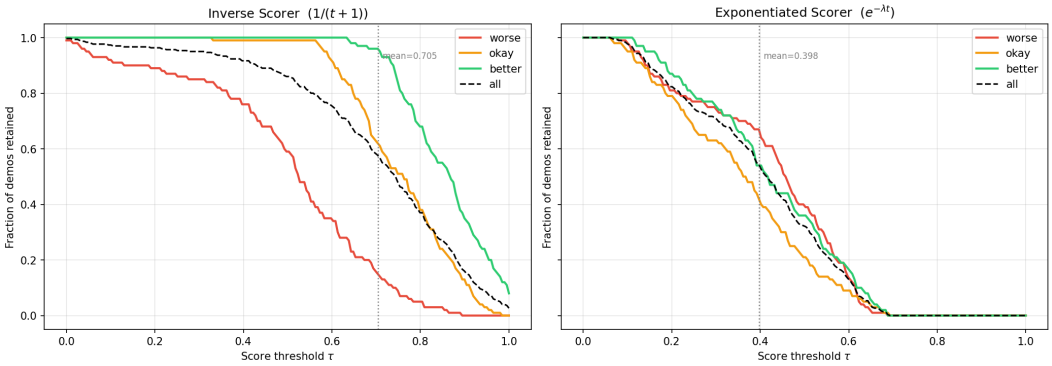

Figure 4: Retention vs. threshold τ for the inverse scorer (left) and exponentiated scorer (right), colored by operator tier. The dotted vertical line marks the mean score. The inverse scorer clearly stratifies tiers; the exponentiated scorer does not.

5 Results

Across all experiments, we report mean rollout success rate, computed as the fraction of evaluation episodes in which the policy completes the task within the horizon.

5.1 Baselines: Vanilla BC and Binary Classification-based Filtered BC

We establish two baselines on the Can task (Figure 5). Vanilla BC, trained on the full unfiltered MH dataset, converges to approximately 0.155 by step 650 and serves as our primary baseline. Binary classification-based filtering, which hard-filters demonstrations that resemble failed rollouts as our Demo-SCORE reproduction, converges slightly higher to 0.177. This plateau of 17.7% represents another baseline for comparison against the variants of our score-weighted approaches.

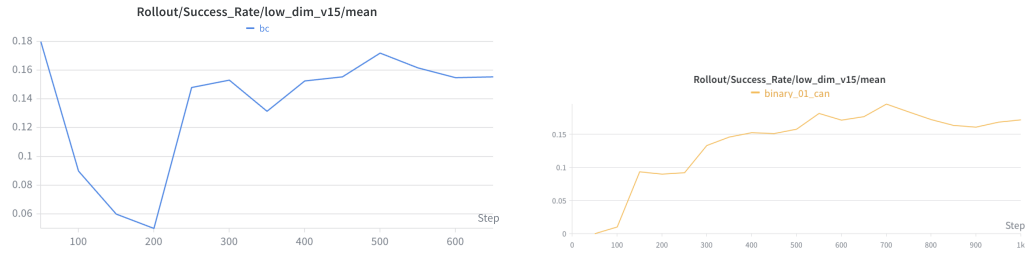

Figure 5: Mean success rate of the two baselines on the Can task. **Left**: vanilla BC, converging to ≈0.155. **Right**: binary classification-based filtered BC, converging to ≈0.177.

5.2 Inverse Trajectory Quality-Weighted BC

Without Thresholding. We find that using this approach without any thresholding ($\tau = 0.0$) does not beat the vanilla BC baseline significantly, as it converges to a mean success rate of approximately 0.184 by step 2000, shown in Table 7.

With Thresholding. Introducing thresholding yields meaningful gains over both the no-threshold approach and the vanilla BC baseline. As shown in Figure 6 and Table 7, when $\tau = 0.0025$, meaning when the scores below the mean are dropped, this approach achieves the highest peak mean

success rate of 0.303 and converges to around 0.24-0.25. $\tau = 0.0035$ sees similar success rates. However, when the threshold is too aggressive, like at $\tau = 0.3$, it significantly underperforms in a way comparable to without thresholding.

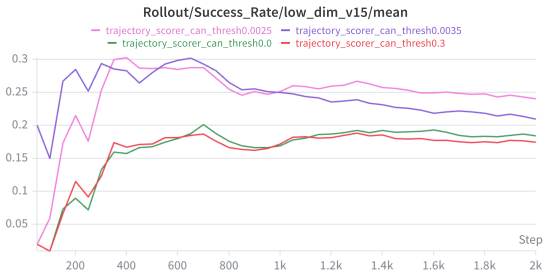

Figure 6: Mean success rate of Inverse Trajectory Quality-Weighted BC across thresholds, on the Can task.

Step	τ=0.0	τ=0.0025	τ=0.0035	τ=0.3
50	0.0200	0.0200	0.2000	0.0200
200	0.0900	0.2150	0.2850	0.1150
400	0.1575	0.3025	0.2825	0.1675
600	0.1800	0.2850	0.2983	0.1817
800	0.1762	0.2550	0.2650	0.1663
1000	0.1690	0.2510	0.2500	0.1720
1200	0.1867	0.2592	0.2358	0.1817
1400	0.1921	0.2571	0.2314	0.1857
1600	0.1931	0.2494	0.2181	0.1775
1800	0.1839	0.2478	0.2183	0.1756
2000	0.1840	0.2405	0.2095	0.1745

Figure 7: Mean success rate at representative steps for each threshold on the Can task. Vanilla BC converges to ≈0.155 for reference.

5.3 Exponentiated Trajectory Quality-Weighted BC

From Figure 8, we find that all exponentiated threshold conditions outperform the binary baseline, which stays below 0.20 across training. As with the inverse approach, performance is non-monotonic in τ, but here higher thresholds dominate: $\tau = 0.7$ attains the overall peak of 0.308 at step 600 and converges to 0.266. This suggests that admitting too many low-quality demonstrations dampens the learning signal even when their weights are sharply reduced.

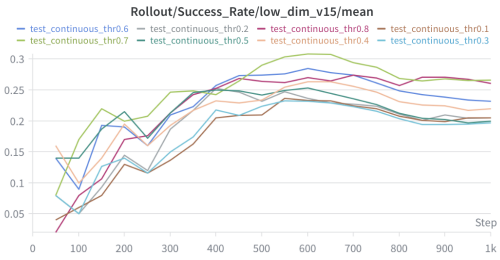

Figure 8: Mean success rate of Exponentiated Trajectory Quality-Weighted BC across $\tau \in \{0.1, \ldots, 0.8\}$ and binary baseline, on the Can task.

Step	τ=.1	τ=.2	τ=.3	τ=.4	τ=.5	τ=.6	τ=.7	τ=.8	Bin.
50	0.040	0.080	0.080	0.160	0.140	0.140	0.080	0.020	0.000
200	0.130	0.145	0.140	0.195	0.215	0.190	0.200	0.170	0.090
400	0.205	0.253	0.218	0.233	0.250	0.258	0.243	0.253	0.153
600	0.233	0.237	0.232	0.263	0.253	0.285	0.308	0.270	0.172
800	0.208	0.211	0.204	0.231	0.213	0.249	0.269	0.258	0.173
1000	0.205	0.205	0.197	0.220	0.200	0.232	0.266	0.261	0.172

Figure 9: Mean success rate at representative steps, exponentiated weighting on the Can task.

5.4 MH Classifier-Weighted BC

We train a deterministic BC policy reweighted by the frozen operator skill classifier (Section 4.3.1) with no thresholding ($\tau = 0.0$), assigning each demo a continuous quality weight $w = 0 \cdot P(\textit{worse}) + 0.5 \cdot P(\textit{okay}) + 1.0 \cdot P(\textit{better})$.

As shown in Figure 10, the policy peaks at a mean success rate of 0.244 at step 450, before stabilizing around 0.21–0.225. This represents a meaningful improvement over the vanilla BC baseline (≈0.155), suggesting that upweighting demonstrations from higher-quality operators provides a useful learning signal even without an explicit threshold to discard low-quality data.

5.5 Lift Task

Inverse Trajectory Quality-Weighted BC. We evaluate generalization of the inverse scorer on the Lift task, using the same three threshold values ($\tau \in \{0.0, 0.0025, 0.0035\}$) from the Can experiments. The aggressive threshold $\tau = 0.3$ is included for consistency but filters out nearly all

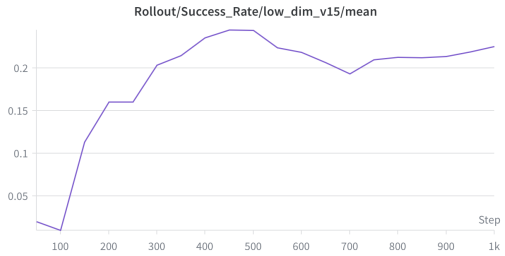

Figure 10: Mean success rate of MH Classifier-Weighted BC (τ=0.0) on the Can task. Peaks at 0.244 (step 450); vanilla BC converges to ≈0.155 for reference.

Lift demonstrations, resulting in zero success throughout training before the run terminates early at step 1400, which confirms that this setting is too aggressive to be useful.

As seen in Figure 11, the convergence of all three conditions toward similar final rates of around 0.20-0.21 suggests that on a simpler task like Lift, the choice of moderate threshold matters less than on Can.

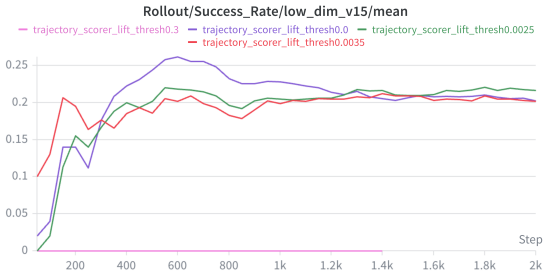

Figure 11: Mean success rate of Inverse Trajectory Quality-Weighted BC across thresholds, on the Lift task. $\tau = 0.3$ reaches zero success and terminates early.

Step	τ=0.0	τ=0.0025	τ=0.0035	τ=0.3
50	0.0200	0.0000	0.1000	0.0000
200	0.1400	0.1550	0.1950	0.0000
400	0.2225	0.2000	0.1850	0.0000
600	0.2617	0.2183	0.2017	0.0000
800	0.2325	0.1962	0.1825	0.0000
1000	0.2280	0.2050	0.1990	0.0000
1200	0.2142	0.2058	0.2050	0.0000
1400	0.2057	0.2164	0.2121	0.0000
1600	0.2081	0.2106	0.2031	—
1800	0.2106	0.2206	0.2089	—
2000	0.2025	0.2165	0.2020	—

Figure 12: Mean success rate at representative steps for each threshold on the Lift task. "—" indicates the run had terminated.

Exponentiated Trajectory Quality-Weighted BC. Compared to our inverse-weighted approach, this exponential scoring approach achieves substantially higher absolute success rates (>0.80), which is higher than any other configuration in this work. As shown in Figure 13 and Table 14, both $\tau = 0.3$ and $\tau = 0.4$ rise steeply through the first 500 steps and converge to roughly 0.80–0.83, while the binary filtration baseline on the same task plateaus near 0.30. However, it should be noted that the training length is shorter than the rest of our experiments, ending at 1000 epochs.

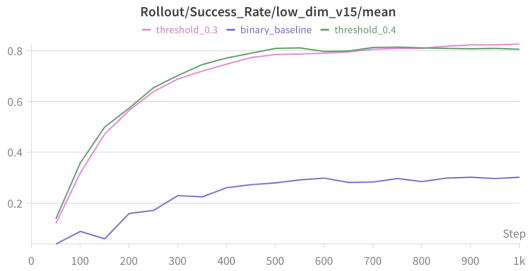

Figure 13: Mean success rate of Exponentiated Trajectory Quality-Weighted BC ($\tau \in \{0.3, 0.4\}$) vs. binary filtration baseline on the Lift task.

Step	τ=0.3	τ=0.4	Binary
50	0.1200	0.1400	0.0400
100	0.3200	0.3600	0.0900
200	0.5650	0.5750	0.1600
300	0.6900	0.7033	0.2300
400	0.7475	0.7725	0.2625
500	0.7860	0.8100	0.2800
600	0.7917	0.7983	0.3000
700	0.8057	0.8129	0.2843
800	0.8100	0.8113	0.2863
900	0.8233	0.8089	0.3022
1000	0.8270	0.8060	0.3030

Figure 14: Mean success rate at representative steps. Binary baseline and τ=0.3 runs end at step 1000; τ=0.4 continues to step 1700.

6 Discussion

From our experiments with thresholding and without, we find that continuous weighting alone is insufficient. The inverse score-weighted objective without any filtering ($\tau = 0.0$) converges to roughly 0.184 on Can, only marginally above the 0.155 vanilla BC plateau. This indicates that simply down-weighting low-quality demonstrations does not meaningfully change the learned policy when those demonstrations still contribute gradient signal. The substantial gains appear only once thresholding removes the lowest-scoring trajectories entirely. At $\tau = 0.0025$, the policy peaks at 0.303 and converges to roughly 0.24-0.25. We interpret this as evidence that soft re-weighting cannot fully overcome a noise floor of unhelpful demonstrations and that thresholding can aid in this.

We further find that threshold selection remains task-dependent. Performance is non-monotonic in the threshold under both scoring approaches, and the best setting differs by task: intermediate thresholds dominate on Can while overly aggressive filtering collapses to baseline, and on Lift the usable thresholds cluster tightly while the aggressive setting fails outright by discarding nearly all demonstrations. This suggests that continuous weighting widens the band of effective thresholds but does not eliminate threshold sensitivity.

Limitations Since the 2 tasks we experiment with (Can and Lift) are simple and in a sense, almost binary in nature, with how rewards are given rather unambiguously (i.e. putting a can in a target bin, lifting a can to a target height = rewarded), this limits the way we can represent the demo quality. We use timesteps until reward, but this may not be the best proxy for demo quality. However, for tasks with such sharply defined success conditions, this may be a coarse proxy that captures speed rather than the broader notion of demonstration usefulness we ultimately care about.

The trajectory scorer is trained exclusively on rollouts generated by the BC policy. As a result, roughly 84.5% of the scorer's training data consists of failures, and the quality signal it learns is defined over a narrow, low-quality slice of trajectory space. When this scorer is then applied to the original MH dataset to assign quality weights, it is extrapolating substantially beyond its training distribution as the MH demonstrations from proficient operators occupy regions of trajectory space the scorer has barely encountered. This raises the possibility that the scorer's rankings reflect proximity to the BC policy's failure modes rather than genuine demonstration quality.

Distribution Mismatch and Quality Signal. The MH Classifier-Weighted BC result offers a direct test of the distribution mismatch concern raised above. Unlike the inverse scorer which is trained on BC rollout failures and applied to MH demonstrations it was never exposed to, the operator skill classifier is trained directly on the same MH dataset it scores, eliminating the extrapolation problem. The higher no-threshold baseline (0.244 peak vs. 0.184 for inverse scoring without thresholding) suggests the distribution mismatch might've impeded the inverse scoring performance. However, it does not reach the peak of the inverse scorer with optimal thresholding (0.303), indicating that thresholding compensates for a noisy quality signal more than a better quality source alone can.

7 Conclusion

In this work, we introduced a continuous, score-weighted behavioral cloning objective that replaces the hard success/failure filtering of Demo-SCORE with a learned inverse-time quality score and exponential quality sore, while experimenting with thresholding that removes the lowest-scoring demonstrations. We demonstrate that our approach achieved better mean success rates than our vanilla BC and binary filtration baselines. We saw a 15% increase in success as compared to vanilla BC, and 9% as compared to binarization for the Can task.

Future Work Can and Lift are both simple tasks, so further experiments can also be run on more complex and long-horizon tasks, like Tool Hang and Transport, to test the performance of our approach in generalizing across complexities of tasks. Given the limitations of time-to-success as a proxy, we also plan to explore alternative quality labels—for example, action smoothness, state-coverage contribution, or learned value estimates—and to revisit weight-normalization so that label transformations like exponentiation translate into the intended gradient magnitudes represented.

Earlier in our paper, we also found that the Diffusion Policy achieved a success rate of 85.2%, which is significantly higher than BC's 15.5%. To see more improvement with our method, future work

could train the scorer on rollouts from the Diffusion Policy instead of vanilla BC, which would expose it to a far richer success distribution, and examine whether the resulting quality rankings over the MH demos change meaningfully.

AI Tools Disclosure

We used Claude (Anthropic) and ChatGPT (OpenAI) as AI assistants during this project to write the boilerplate code for our scorer and classifier functions, data loading pipelines, and HDF5 file parsing utilities. AI was also used to help identify environment and dependency issues.

We developed independently the core algorithmic contributions of exponentiated and inverse weight scoring and thresholding approaches and the BC training objective.

8 Team Contributions

- **Sydney:** Implemented pipeline to process and save rollout data, implemented binary classification baseline as well as continuous exponentially scored trajectory MLP, implemented weighted BC algorithm and integrated with existing pipeline. Formulated and implemented time-based scoring and thresholding. Trained and produced rollout data on Lift dataset.
- **Evy:** Trained the operator skill classifier on the MH Can dataset, integrated the classifier into the BC training pipeline as a rollout-free quality scorer, and conducted dataset heterogeneity analysis including trajectory length visualizations for both Can and Lift tasks.
- **Tracy:** Trained and produced rollout data from 2 policies on Can dataset with imitation learning algorithms, implemented inverse trajectory quality-weighted BC approach with thresholding for both Can and Lift datasets, set up script for training with Modal GPUs

Changes from Proposal In this final form of our project, rather than collecting our own simulated dataset, we used the open-source robomimic Multi-Human split and validated generalization on a second task (Lift). More substantially, we changed the scoring function: the proposal planned to use the classifier's sigmoid probability directly as a continuous $[0, 1]$ quality score, but we found trajectory length to be strongly correlated with operator skill and instead adopted an inverse time-to-success label. Our proposal also intended to avoid hard thresholds entirely in favor of pure soft reweighting, but in practice, we found continuous weighting alone yielded only marginal gains over vanilla BC, so we reintroduced thresholding and analyzed its sensitivity. We additionally added two components not in the proposal: an exponentiated-weight variant and an operator skill classifier baseline.

AI Disclosure Statement We used Claude Code during the coding part of our project. These tools assisted with standard coding tasks such as setting up PyTorch Dataset and DataLoader boilerplate, HDF5 file I/O, checkpoint save/load patterns, and debugging issues, like observation array shape mismatches and sample index alignment. We also used AI to help wire the bc_custom algorithm into robomimic's config/factory system and configure Modal for remote training. All essential project components were developed independently. This includes the core data curation framing (using rollout reward signals as proxy quality labels), the scorer and classifier architectures (trajectory chunking with mean-pooled embeddings, per-timestep credit assignment via discounted returns, binary success classification), the BC_Custom loss reweighting scheme (per-sample weighted losses normalized to mean 1 per batch with score thresholding), and all experimental design decisions.

References

Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. 2023. Data Quality in Imitation Learning. arXiv:2306.02437 [cs.RO] https://arxiv.org/abs/2306.02437

Annie S. Chen, Alec M. Lessing, Yuejiang Liu, and Chelsea Finn. 2025. Curating Demonstrations using Online Experience. arXiv:2503.03707 [cs.RO] https://arxiv.org/abs/2503.03707

Joey Hejna, Suvir Mirchandani, Ashwin Balakrishna, Annie Xie, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, Dhruv Shah, Coline Devin, and Dorsa Sadigh. 2025. Robot Data Curation with Mutual Information Estimators. arXiv:2502.08623 [cs.RO] https://arxiv.org/abs/2502.08623

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. 2021. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *arXiv preprint arXiv:2108.03298*.

Sreevardhan Sirigiri, Nathan Samuel de Lara, Christopher Agia, Florian Shkurti, and Fabio Ramos. 2026. Diversity You Can Actually Measure: A Fast, Model-Free Diversity Metric for Robotics Datasets. arXiv:2603.11634 [cs.RO] `https://arxiv.org/abs/2603.11634`