STEAM: SELF-SUPERVISED TEMPORAL ENSEMBLE ADVANTAGE MODELING FOR REAL-WORLD ROBOT LEARNING

Anonymous authors
Paper under double-blind review

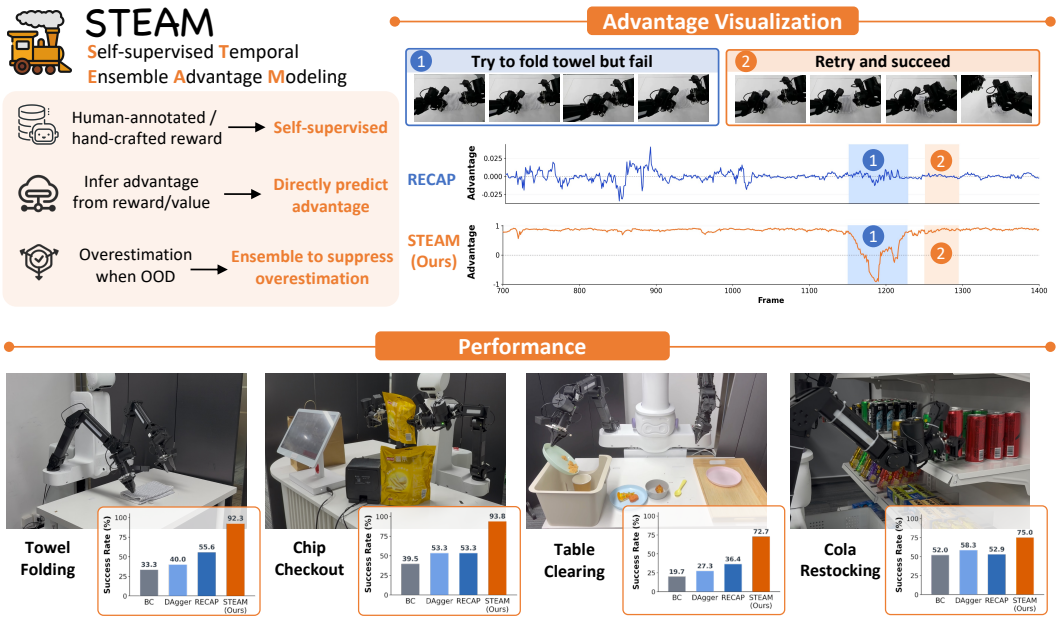


Figure 1: **Overview.** STEAM is a self-supervised advantage modeling framework for real-world robot learning. It learns advantage prediction offline from expert demonstrations without manual annotations or hand-crafted rewards, and can be applied to expert data, human corrections, and policy rollouts to provide robust advantage estimates. When combined with CFGRL (Frans et al., 2025), STEAM substantially improves policy performance on various real-world tasks.

ABSTRACT

Real-world robot learning increasingly relies on heterogeneous data, but demonstrations and rollouts often mix useful progress with stalls, corrections, and suboptimal behavior. Effective policy learning therefore requires frame-level advantages that distinguish reliable local progress from failures and regressions. We propose **S**elf-supervised **T**emporal **E**nsemble **A**dvantage **M**odeling (**STEAM**), a label-free method that learns such advantages from expert demonstrations. STEAM trains an ensemble of temporal-offset predictors on frame pairs within expert trajectories, using the normalized temporal offset between two frames as a self-supervised signal. Each predictor maps a frame pair to a distribution over temporal offsets, which is converted into a scalar advantage. STEAM then takes the minimum advantage across the ensemble to score mixed-quality rollout data conservatively. Across real-world bimanual towel folding, chip checkout, table clearing, and cola restocking tasks, STEAM identifies stalls, failures, and recoveries. When combined with CFGRL, STEAM achieves the highest policy success rate on all four tasks, averaging 83.5% against 58.8% for the best baseline.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

The rapid progress of robot foundation models relies heavily on large collections of expert demonstrations. However, collecting expert trajectories is expensive, and real robot datasets often contain mixed-quality behavior, especially in complex long-horizon tasks (Belkhale et al., 2023; Chen et al., 2026; Mao et al., 2026). To broaden state-action coverage and improve robustness, recent methods (Amin et al., 2025; Yang et al., 2026) incorporate non-expert trajectories, including policy rollouts, human interventions, and unsuccessful trials. These trajectories can be useful, but their quality often changes within the same episode. A rollout may make progress before failing, and a human-intervention trajectory may contain poor autonomous behavior before recovery. Trajectory-level filtering can therefore discard useful transitions or keep harmful ones. This raises a central question for policy improvement: *how can we assign fine-grained credit to distinguish task-advancing actions from stalled or regressive ones?*

We formulate this question as frame-level advantage estimation. Estimating such advantages without dense rewards remains difficult. Recent robot-learning methods often follow a two-stage pipeline that first learns an intermediate reward or value signal (Chen et al., 2026; Amin et al., 2025; Yang et al., 2026; Lee et al., 2026; Liu et al., 2026; Zhai et al., 2025), then converts it into an advantage or sample weight for weighted behavior cloning (Chen et al., 2026; Mao et al., 2026; Xu et al., 2022) or offline reinforcement learning (Frans et al., 2025; Kostrikov et al., 2022). However, deriving reliable signals remains difficult for three reasons. First, hand-crafted rewards, human annotations, and cross-trajectory calibration require substantial external supervision and limit scalability. Second, VLM-based reward or value models require broad pretraining and can still produce noisy frame-level signals when visual-language priors are not physically grounded. Third, many progress estimators impose an absolute or monotonic notion of progress, which is poorly matched to real rollouts that may temporarily regress, recover after mistakes, or leave the expert distribution. When a learned reward or value model is applied to these out-of-distribution rollout states, it may assign high advantages to unreliable transitions, creating false-positive training signals that drive policy optimization toward misleading behavior.

Instead of assigning an absolute progress value to each frame, we use the relative order of frames within the same expert trajectory. This relative signal is local to each demonstration and avoids cross-trajectory calibration or a hand-specified global progress curve. For any two frames, the normalized temporal offset can be computed directly from the trajectory. This yields dense pairwise progress supervision without hand-crafted rewards, human labels, or external value estimates.

Building on such motivations, we introduce **STEAM** (**S**elf-supervised **T**emporal **E**nsemble **A**dvantage **M**odeling), a label-free framework for frame-level advantage estimation. STEAM trains an ensemble of temporal-offset predictors on expert frame pairs. Each predictor outputs a distribution over temporal bins, which is converted into a scalar advantage that reflects local temporal efficiency. STEAM uses reversed frame pairs from successful trajectories to expose regressive motion without failed demonstrations. To score mixed-quality trajectories robustly, STEAM takes the minimum advantage across the ensemble, suppressing overestimated advantages on out-of-distribution samples. The resulting STEAM advantage can then be used to select high-advantage frames for policy improvement with offline RL methods such as classifier-free guidance reinforcement learning (CFGRL) (Frans et al., 2025). Our contributions are threefold.

- We formulate label-free frame-level advantage estimation from expert temporal structure, using normalized temporal offsets between frame pairs as dense self-supervised targets.
- We propose STEAM, which converts distributional temporal-offset predictions into scalar advantages and uses conservative ensemble aggregation to suppress overestimated advantages on mixed-quality data.
- We validate STEAM on real-world bimanual towel folding, chip checkout, table clearing, and cola restocking tasks. STEAM localizes stalls, failures, and recoveries in demonstrations and rollouts, and when combined with CFGRL achieves the highest policy success rate on all four tasks, averaging 83.5% against 58.8% for the best baseline.

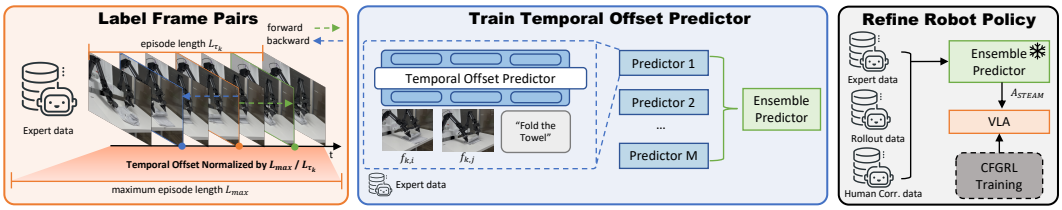


Figure 2: **STEAM framework.** (a) Expert demonstrations provide frame pairs for normalized temporal offset calculation. Both forward and reversed pairs are used as self-supervised targets. (b) An ensemble of M predictors is trained on expert data to map frame pairs and language instructions to categorical distributions over temporal bins, converting these into scalar advantage scores. (c) The trained ensemble scores mixed-quality data, and these A_{STEAM} advantages then guide a VLA policy through CFGRL for robot policy refinement.

2 Related Work

Reward Models for Robot Manipulation. Refining policies from mixed-quality data requires fine-grained signals, such as rewards, from which value and advantage estimates can be derived to distinguish useful behaviors from suboptimal ones (Peng et al., 2019; Kostrikov et al., 2022). Since explicit rewards are rarely available in real-world robot manipulation, recent works instead learn reward or value estimates from auxiliary supervision, including human annotations (Chen et al., 2026; Mao et al., 2026), hand-designed reward or progress targets (Amin et al., 2025; Yang et al., 2026), and foundation-model priors (Lee et al., 2026; Tan et al., 2026; Ma et al., 2025; Liang et al., 2026). In contrast, STEAM predicts advantage directly rather than recovering it from a learned reward or value (Amin et al., 2025; Yang et al., 2026). This is similar to ARM (Mao et al., 2026), which estimates relative progress between frame pairs. The key difference lies in supervision: ARM relies on a classifier trained on manually annotated data, whereas STEAM exploits the temporal structure of expert demonstrations as self-supervision, requiring no manual annotation.

Temporal Structure as Supervision Signals. Temporal order is a free supervisory signal, since a frame's position along a trajectory encodes task progress. This idea has long been used in self-supervised video representation learning (Dwibedi et al., 2019; Zakka et al., 2022; Nair et al., 2022; Ma et al., 2023b;a). In robot learning, temporal order is typically converted into a dense reward signal to guide policy learning. TimeRewarder (Liu et al., 2026) regresses frame-wise temporal offsets into dense rewards. ReWiND (Zhang et al., 2025) rewinds successful videos into failure-like reward sequences. VLAC (Zhai et al., 2025) uses a large vision-language critic for signed progress deltas. Among existing methods, TimeRewarder is the closest prior work to STEAM, as both exploit signed temporal offsets between frame pairs, but STEAM uses them as per-frame advantages for offline data-quality assessment rather than dense rewards for online RL.

3 Methodology

STEAM trains an ensemble of temporal-offset predictors using expert demonstrations. These predictors then identify high-quality frames within heterogeneous, mixed-quality data to enhance policy performance (Figure 2). Section 3.1 details our self-supervised temporal offset learning target. Section 3.2 turns those offsets into robust scalar advantages through a signed-bin predictor, and then into the binary optimality labels that policy training consumes. Section 3.3 shows that the resulting score is an H-step advantage under an expert reference policy, so STEAM models the advantage directly instead of routing through a reward or a value model.

3.1 Normalized Temporal Offset as Learning Target

Self-Supervised Temporal Targets. Our training data comes exclusively from expert demonstrations. Despite occasional brief stalls, the inherent temporal progression within these trajectories reliably indicates task advancement, eliminating the need for hand-crafted rewards. For an expert episode $\tau_k = (f_{k,1}, f_{k,2}, \ldots, f_{k,L_{\tau_k}})$, we define the temporal offset between frames $f_{k,i}$ and $f_{k,j}$

as:

$$\Delta_{\tau_k}(i,j) = \Delta(f_{k,i},\, f_{k,j}) = j - i. \tag{1}$$

Positive offsets, pairing a frame with a future observation, supervise forward progress. Conversely, reversing expert trajectories provides negative offsets, creating pseudo-failure sequences that allow the model to learn regressive behaviors from successful demonstrations alone.

Trajectory-Length Normalization. Raw temporal offsets are not directly comparable across trajectories due to varying execution times. A given offset $\Delta_{\tau_k}(i,j)$ might signify minor progress in a long episode but substantial progress in a short one. To address this, we normalize the offset by trajectory length:

$$\tilde{\Delta}_{\tau_k}(i,j) = \Delta(f_{k,i},\, f_{k,j}) \cdot \frac{L_{\max}}{L_{\tau_k}} = (j-i) \cdot \frac{L_{\max}}{L_{\tau_k}}, \tag{2}$$

where $L_{\max}$ is the maximum length among expert episodes, or a high quantile of the episode lengths to reduce sensitivity to outliers. This normalization is crucial as it unifies temporal-offset targets from diverse trajectories onto a consistent mathematical scale, ensuring stable supervision during training.

3.2 From Offset Prediction to Optimality Labels

Categorical Prediction and Bin Assignment. We model the temporal-offset predictor as a distributional model. Given the long, high-frequency expert episodes, directly predicting continuous offsets would lead to an unmanageably large output space. To address this, we bound the continuous temporal offset $\tilde{\Delta}_{\tau_k}(i,j)$ within $[-\tilde{\Delta}_{\max}, \tilde{\Delta}_{\max}]$ and discretize it into N evenly spaced bins. We write $\tilde{\Delta}^{\mathcal{B}}_{\tau_k}(i,j)$ for the index of the bin containing the offset, and use its one-hot encoding of length N as the training target. Our predictor, $p_\theta(\Delta \mid f_{k,i}, f_{k,j}, \ell)$, parameterized by θ, maps an expert frame pair $(f_{k,i}, f_{k,j})$ and its natural language instruction ℓ to a categorical distribution over these N temporal bins. We train this predictor by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{\tau_k,\, i,\, j}\Big[\mathcal{H}\big(\,\tilde{\Delta}^{\mathcal{B}}_{\tau_k}(i,j),\; p_\theta(\Delta \mid f_{k,i}, f_{k,j}, \ell)\big)\Big], \tag{3}$$

where $\mathcal{H}$ denotes the cross-entropy between the one-hot target and the predicted categorical distribution.

Advantage Modeling. To quantify a frame's contribution to task progress, we derive a scalar advantage from the learned distributional predictor. Unlike standard RL advantages, our metric focuses solely on temporal efficiency. For a frame $f_{k,i}$, we infer the model's prediction using a future frame $f_{k,i+H}$ (fixed lookahead H). We then collapse the predicted categorical distribution into an expected bin index. The advantage score is computed as the normalized expected bin index minus the ground-truth quantized temporal offset:

$$A(f_{k,i};\, \theta) \;=\; \frac{2}{N}\left[\mathbb{E}_{b\sim p_\theta(\cdot \mid f_{k,i}, f_{k,i+H}, \ell)}[b] - \tilde{\Delta}^{\mathcal{B}}_{\tau_{\max}}(i, i+H)\right]. \tag{4}$$

Since H is constant and $\tau_{\max}$ is the longest expert episode, $\tilde{\Delta}^{\mathcal{B}}_{\tau_{\max}}(i, i+H)$ is a fixed bin index. This formulation inherently prioritizes execution efficiency due to the trajectory-length normalization (Eq. 2). Shorter, more efficient expert demonstrations, scaled by a factor greater than one, yield higher advantage scores, while slower, suboptimal executions are penalized.

Ensemble for Overestimation Control. A single predictor trained on expert data can become overly confident on out-of-distribution rollout samples, leading to severely overestimated advantage scores (Lakshminarayanan et al., 2017). Such overestimations generate false-positive signals that degrade policy refinement. To mitigate this, we employ a conservative ensemble strategy. We train M independent temporal offset predictors $\Theta = \{\theta_1, \ldots, \theta_M\}$ with distinct random initializations using Equation 3. Our final metric, the STEAM advantage, is the minimum predicted advantage across the ensemble:

$$A_{\text{STEAM}}(f_{k,i};\, \Theta) \;=\; \min_{m=1,\ldots,M} A(f_{k,i};\, \theta_m). \tag{5}$$

This minimum-aggregation exploits the tendency of ensemble members to agree within the training distribution but diverge in unfamiliar state spaces (Lakshminarayanan et al., 2017). By penalizing

high variance, this acts as a robust regularizer against reward overoptimization, a common issue in RLHF (Coste et al., 2024). While this conservative bias might slightly reduce recall, it crucially suppresses false positives, which our empirical validation (Section 4.3) shows is vital for stable and effective policy optimization.

From Advantages to Optimality Labels. Policy training consumes A_{STEAM} through a binary optimality label. Since expert demonstrations and non-expert data (human corrections and autonomous rollouts) occupy distinct advantage ranges, we threshold each of the two pools separately rather than at a shared absolute value:

$$o_{k,i} = \mathbf{1}\left[A_{\mathrm{STEAM}}(f_{k,i};\Theta) \geq \delta_q\right], \tag{6}$$

where δ_q labels the top-q fraction of each pool as optimal. We supply $o_{k,i}$ as the optimality condition of CFGRL (Frans et al., 2025), which steers action generation toward frames with high estimated local progress. Appendix A.5 gives the full training and inference procedure.

3.3 Theoretical Justification: Direct Advantage Modeling

Although STEAM fits neither a reward model nor a value function, its score is an H-step advantage when elapsed time is the only cost, as we show next.

Reference value. Let every step before task completion receive the reward $r(s) = -1$, and let π_{ref} be a reference expert that completes the task in $L_{\max}$ steps. Without discounting ($\gamma = 1$), the value $V_{\mathrm{ref}}(s)$ is the negative expected number of steps π_{ref} still needs from state s.

H-step advantage. For the H-step segment that starts at frame $f_{k,i}$, whose underlying state we denote by $s_{k,i}$, the undiscounted H-step advantage (Schulman et al., 2016; Amin et al., 2025) relative to π_{ref} is

$$A^{(H)}(s_{k,i}) \;=\; \underbrace{\sum_{h=0}^{H-1} r(s_{k,i+h})}_{=-H} \;+\; \underbrace{V_{\mathrm{ref}}(s_{k,i+H}) - V_{\mathrm{ref}}(s_{k,i})}_{=\Delta V_{\mathrm{ref}}} \;=\; \Delta V_{\mathrm{ref}} - H. \tag{7}$$

Since the reward term is the constant $-H$, estimating the advantage reduces to estimating the value difference ΔV_{ref}, which is the expected number of steps that π_{ref} needs to move from $s_{k,i}$ to $s_{k,i+H}$.

Temporal offsets as value-difference estimates. STEAM estimates this value difference directly instead of fitting V_{ref}: an expert that moves from $s_{k,i}$ to $s_{k,j}$ takes $j - i$ of its own steps, and the rescaling by $L_{\max}/L_{\tau_k}$ in Equation 2 converts them into steps of π_{ref}, so that the normalized offset is a sampled estimate of the value difference:

$$\tilde{\Delta}_{\tau_k}(i,j) \;=\; (j-i)\,\frac{L_{\max}}{L_{\tau_k}} \;\approx\; V_{\mathrm{ref}}(s_{k,j}) - V_{\mathrm{ref}}(s_{k,i}). \tag{8}$$

Deviations of individual episodes from the average pace average out over demonstrations, so the predictor trained with Equation 3 estimates the value difference itself, much as sampled returns serve as targets for a value function in RECAP (Amin et al., 2025), but without fitting and then differencing a value function.

Correspondence to the STEAM advantage. In Equation 4, $\mathbb{E}[b]$ estimates the bin of ΔV_{ref} and the subtracted term is the bin of H, since the longest episode satisfies $\tilde{\Delta}_{\tau_{\max}}(i, i+H) = H$; as the bins have equal width $2\tilde{\Delta}_{\max}/N$,

$$A(f_{k,i};\theta) \;=\; \frac{2}{N}\Big[\mathbb{E}[b] - \tilde{\Delta}^{\mathcal{B}}_{\tau_{\max}}(i, i+H)\Big] \;\approx\; \frac{2}{N}\cdot\frac{N}{2\tilde{\Delta}_{\max}}\big(\Delta V_{\mathrm{ref}} - H\big) \;=\; \frac{A^{(H)}(s_{k,i})}{\tilde{\Delta}_{\max}},$$

that is, the STEAM advantage equals the H-step advantage scaled by the positive constant $1/\tilde{\Delta}_{\max}$. In practice, we omit the constant term $\tilde{\Delta}^{\mathcal{B}}_{\tau_{\max}}(i, i+H)$, which does not change the labels in Equation 6.

4 EXPERIMENTS

Tasks. We evaluate on four long-horizon bimanual tasks on an ARX dual-arm robot: towel folding, chip checkout, table clearing, and cola restocking. They involve deformable objects, arm-to-arm handovers, and placement into tight spaces. Appendices A.1 and A.2 give the robot setup, stage-by-stage procedure, and dataset composition of every task.

Baselines. We compare STEAM against four reward model baselines. **ReWiND** (Zhang et al., 2025) rewinds successful demonstrations to synthesize failures, so its reward drops when the robot regresses. **TimeRewarder** (Liu et al., 2026) learns a dense reward by regressing the temporal offset between video frames. **RoboMeter** (Liang et al., 2026) uses a VLM as a general-purpose robotic reward model; we use the released RoboMeter-4B checkpoint. **RECAP** (Amin et al., 2025) uses task success and elapsed time as its reward. The reward models of ReWiND and TimeRewarder are trained on expert demonstrations only. Each reward model then labels all expert and rollout frames, and these labels are used to train a value model and compute the advantage as in RECAP. STEAM is trained on expert demonstrations only and predicts the advantage directly, without fitting a value model. For policy learning, all methods label the same data and train the policy with CFGRL under identical settings.

Implementation. Each STEAM advantage predictor uses SigLIP-SO400M as the visual encoder and Gemma-3-270M as the language backbone, followed by task-specific prediction heads. Hyperparameters are given in Appendix A.5.

We answer three questions:

- **Q1.** Does STEAM produce more robust advantage estimates than existing reward and value models on long-horizon, real-world manipulation data?
- **Q2.** Does STEAM improve real-world policy performance?
- **Q3.** How do STEAM's design choices and data composition affect final performance?

4.1 Q1: ADVANTAGE ESTIMATION QUALITY

Protocol and metrics. Evaluation consists of two parts. First, **Clustering accuracy** tests whether the advantage statistically distinguishes between different successful and failed episodes. Intuitively, successful episodes should accumulate higher advantage than failed ones. We select 30 successful and 30 failed episodes and fit a two-component Gaussian mixture to divide them into two clusters by the sum of advantages. Episodes in the higher-mean cluster are labeled $\hat{y}(\tau) = 1$, otherwise $\hat{y}(\tau) = 0$. The clustering accuracy is the fraction of episodes classified correctly,

$$\text{Clustering accuracy} = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \mathbf{1}\big[\hat{y}(\tau) = y(\tau)\big], \tag{9}$$

where $y(\tau)$ is the ground-truth success label and $\mathcal{T}$ is the set of 60 episodes. Second, **Intervention AUROC (I-AUROC)** tests whether a model detects an impending failure. We use 20 human-correction episodes, in which an operator takes over when the policy is about to fail. Around each takeover frame t_{int}, we compare pre-takeover frames $\mathcal{F}^-$ with post-takeover frames $\mathcal{F}^+$,

$$\mathcal{F}^- = [\, t_{\text{int}} - H - W,\ t_{\text{int}} - H \,), \qquad \mathcal{F}^+ = [\, t_{\text{int}} + 5W,\ t_{\text{int}} + 6W \,), \tag{10}$$

where $W = 40$ (2 s at 20 Hz). I-AUROC is the probability that a post-takeover frame receives a higher advantage than a pre-takeover frame,

$$\text{I-AUROC} = \Pr\big[A(f_i) > A(f_j)\big], \quad f_i \in \mathcal{F}^+,\ f_j \in \mathcal{F}^-. \tag{11}$$

Results. Table 1 compares the performance of the five models. STEAM achieves the best results on both metrics, with 93.3% clustering accuracy and 78.8% I-AUROC, exceeding the strongest baseline by 5.0 and 13.7 points. Its advantage drops as soon as the policy begins to fail and recovers once the task advances again, whereas the baselines respond weakly to failure; ReWiND, for example, stays near chance at 53.3% I-AUROC, almost unable to distinguish pre- and post-takeover frames.

We further visualize the advantages of all models on a successful and a failed towel-folding episode in Figure 3. STEAM stays at a stable positive level during normal execution. In the successful

Table 1: **Advantage estimation quality.** Clust. Acc.: clustering accuracy (%) of an unsupervised two-component Gaussian mixture on episode advantage sums. I-AUROC (%): probability that a post-takeover frame receives a higher advantage than a pre-takeover frame.

Method	Clust. Acc. ↑	I-AUROC ↑
ReWiND (Zhang et al., 2025)	63.3	53.3
TimeRewarder (Liu et al., 2026)	88.3	65.1
RoboMeter (Liang et al., 2026)	75.0	45.5
RECAP (Amin et al., 2025)	60.0	48.3
STEAM (Ours)	**93.3**	**78.8**

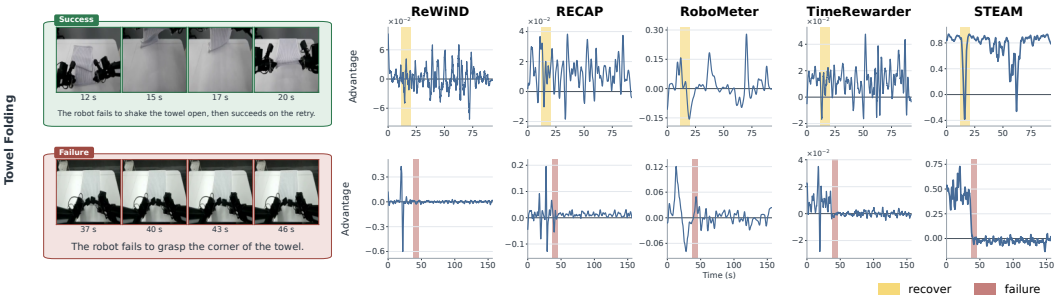


Figure 3: **Frame-level advantages on a successful and a failed towel-folding episode.** Each column shows the advantage assigned by one model to the same two episodes. In the successful episode (top), the robot fails to shake the towel open and then succeeds on the retry; the shaded band marks the recovery. In the failed episode (bottom), the robot never grasps the corner of the towel; the shaded band marks the onset of failure. Note the differing vertical scales.

episode, it dips at the retry and recovers; in the failed episode, it falls to zero once the robot stops making progress. In contrast, the baselines fluctuate even during normal execution. We attribute this difference to STEAM modeling the advantage directly, which reduces estimation error. Appendix A.3 and A.4 show more episodes and the corresponding advantage distributions.

4.2 Q2: STEAM Enhances Policy Performance

We next ask whether better advantage estimates translate into better policies. All methods label the same mixture of expert demonstrations, human corrections, and autonomous rollouts and train a π_0 (Black et al., 2025) policy through CFGRL for one iteration.

Table 2: **Policy performance comparison.** Succ. represents the average success rate (%), Score denotes the average number of completed sub-stages, Thr. is the throughput measured in successful episodes per hour, and Average is the mean success rate over the four tasks.

Method	Towel Folding			Chip Checkout			Table Clearing			Cola Restocking			Average
	Succ. (%)	Score	Thr.	Succ. (%)	Score	Thr.	Succ. (%)	Score	Thr.	Succ. (%)	Score	Thr.	Succ. (%)
TimeRewarder	53.3	3.9	27	70.6	5.7	30	46.7	14.5	20	64.7	2.6	64	58.8
RoboMeter	60.0	4.1	30	58.8	4.8	24	33.3	12.3	13	60.0	2.7	57	53.0
RECAP	55.6	2.9	39	53.3	5.3	24	36.4	12.5	15	52.9	2.1	46	49.6
STEAM	**92.3**	**4.9**	**58**	**93.8**	**7.5**	**48**	**72.7**	**17.7**	**37.5**	**75.0**	**3.0**	**90**	**83.5**

As shown in Table 2, STEAM achieves the highest success rate on every task and raises the average success rate to 83.5%, while the best baseline reaches only 58.8%, and its margin is largest on the more complex tasks: STEAM leads the best baseline by 32.3 points on towel folding and 26.0 points on table clearing, which involves a deformable object and multi-step processes respectively. In contrast, on the simpler cola restocking task the margin narrows to 10.3 points. We attribute this gap to the more stable advantage estimation of STEAM: the baselines must train a value model to obtain their advantages, whereas STEAM predicts the H-step advantage directly from two frames. In complex tasks with frequent retries and stalls, such a value model is hard to fit, and its errors (Figure 3) lead the policy to imitate unproductive segments. Conversely, on tasks composed of simple steps,

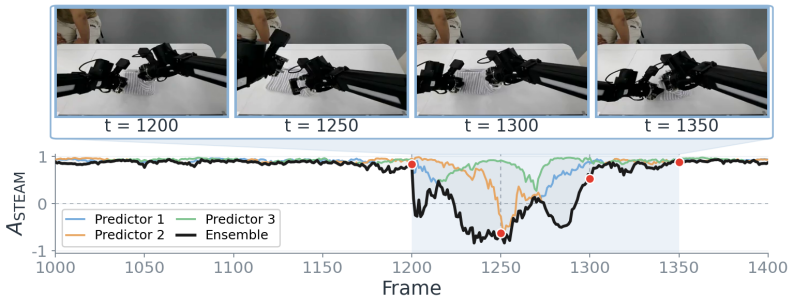


Figure 4: **Effect of conservative ensemble-min aggregation.** Advantages of the three ensemble members and the aggregated A_{STEAM} (black) on a towel-folding rollout. During the retry of the final fold (shaded, frames 1200–1350), Predictor 3 (green) stays near $+1$, while the minimum drops below -0.5.

the advantage of STEAM becomes less pronounced. Appendix A.7 provides a further experiment on a short-horizon, two-stage task, where near-optimal expert data leave little improvement space until rollouts are added.

Moreover, STEAM does not trade speed for success: it also achieves the highest throughput, since it assigns lower advantages to segments executed more slowly than the expert (Equation 7). We provide more visualizations in Appendix A.3.

4.3 Q3: Ablations on Design Choices and Data Composition

We first ablate the bin count N and ensemble size M on the towel-folding task with all expert demonstrations, human corrections, and rollout episodes, and then vary the composition of the training data. Table 3 reports the effect of bin count, and Table 4 reports the effect of ensemble size.

Table 3: Effect of bin count N.

Bins	Succ.	Score	Thr.
$N = 2$	27.3	2.8	41
$N = 8$	54.6	3.8	51
$N = 32$ (default)	**92.3**	**4.9**	**58**

Table 4: Effect of ensemble size M.

Ens.	Succ.	Score	Thr.
$M = 1$	72.7	3.9	53
$M = 3$ (default)	**92.3**	**4.9**	**58**
$M = 5$	90.9	4.6	55

Effect of bin count. The bin count N sets the resolution of the temporal-offset target. With $N = 2$, the target reduces to the sign of the offset, so every forward-moving segment receives nearly the same advantage regardless of its pace. This coarse signal even falls below behavior cloning on towel folding (27.3% against 33.3% in Figure 5): the labels discard frames without ranking them by quality. Finer bins let STEAM separate fast progress from slow progress, raising the success rate to 54.6% at $N = 8$ and 92.3% at $N = 32$, and throughput from 41 to 58 successful episodes per hour, consistent with the finer target distinguishing execution pace.

Effect of ensemble size. The ensemble size M controls how conservative the worst-of-M aggregation is. A single predictor ($M = 1$) cannot flag its own errors on out-of-distribution rollout frames, so overestimated advantages admit poor segments into the optimal set and the success rate drops to 72.7%. With $M = 3$, disagreement among the members pulls these frames down, and the success rate rises to 92.3%. A fifth member brings no further gain (90.9%), so we use $M = 3$ and avoid the extra training cost. With a single predictor, such false positives appear as in Figure 4: Predictor 3 keeps a high advantage during a retry that Predictors 1 and 2 flag as regression, and the ensemble minimum suppresses it.

Effect of training data composition. To verify the effectiveness of STEAM on data from different sources, we further ablate the composition of the training data, from expert demonstrations only (STEAM (Exp)) to expert demonstrations with human corrections (STEAM (Exp+Dagg)) and

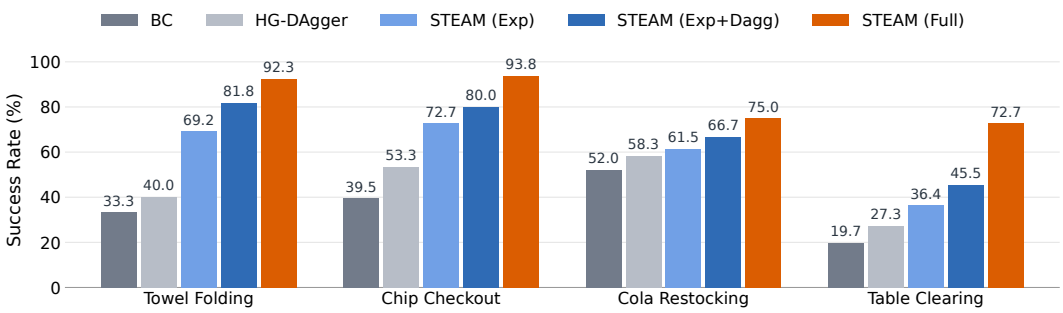


Figure 5: **STEAM performance across different training data combinations.** We compare the success rate of Behavior Cloning and HG-DAgger against STEAM trained with varying sources of data: expert demonstrations only (STEAM (Exp)), expert data supplemented with human correction episodes (STEAM (Exp+Dagg)), and the full dataset containing expert, correction, and autonomous rollout data (STEAM (Full)).

the full mixture with autonomous rollouts (STEAM (Full)), and compare each setting against behavior cloning (BC) and HG-DAgger (Kelly et al., 2019) in Figure 5. STEAM effectively selects high-quality frames from both human corrections and autonomous rollouts, which cover states that expert demonstrations rarely visit, so adding these sources raises the success rate over STEAM (Exp) on all four tasks. Even in the expert-only setting, STEAM (Exp) improves over both BC and HG-DAgger on all four tasks, although HG-DAgger additionally trains on human corrections, showing that STEAM isolates and emphasizes the task-advancing segments within expert demonstrations. The only exception is a short-horizon task with clean but limited expert data, analyzed in Appendix A.7, where filtering expert data alone slightly underperforms BC until rollouts and corrections are added.

5 Limitations

STEAM learns advantages through temporal offsets, where frame pairs with the same offset receive the same target value. This ignores that different task phases may contribute unequally to progress. Structure-aware methods such as SPARS (He et al., 2026) assign different reward signals to different task stages, suggesting a possible way to incorporate phase-aware progress for fine-grained evaluation. Also, STEAM mainly relies on visual observations, which may fail to capture subtle but critical state differences. Adding robot state as a learning target could further expand STEAM's ability to retrieve low-quality frames beyond visually observable errors.

6 Conclusion

We presented STEAM, a self-supervised framework that learns frame-level advantages from expert demonstrations alone, without hand-crafted rewards, annotations, or a fitted value model. By training an ensemble on length-normalized temporal offsets, STEAM directly estimates an H-step advantage, which separates high- and low-quality segments across expert, corrected, and rollout data. Its advantage separates successful from failed episodes with 93.3% clustering accuracy and detects impending failures with 78.8% I-AUROC, exceeding the strongest baseline by 5.0 and 13.7 points, respectively. Combined with CFGRL on four real-world bimanual tasks, it achieves the highest success rate and throughput on every task, with the largest gains on the most complex ones. These results suggest that estimating the advantage directly from temporal progress is a more reliable route to learning from mixed-quality robot data than fitting a reward model and then a value function. Ablations show that fine-grained bins and the ensemble are both essential, and that every added data source, even of lower quality, further improves the policy. Future work includes phase-aware progress and robot state to capture errors beyond visual observation.

AI USE STATEMENT

In this work, we used generative AI tools to clean and reformat datasets, to assist with translation, to create and edit software code, and to draft and edit parts of this paper for readability. We have not used generative AI tools to generate synthetic data sets, to design or provide feedback on our research methodology or experiments, to implement the proposed method, or to analyze or interpret our experimental results. The conceptual framework of Section 3.3, its mathematical claims, and the critical ingredients of their derivation were developed by the authors.

We have reviewed all AI-assisted work. All LLM-generated code was read, tested, and verified for correctness by the authors, and every experimental number reported in this paper comes from real-robot runs of our own pipeline rather than from any AI-generated content. All AI-drafted or AI-edited text was checked sentence by sentence by the authors to ensure that the technical claims are accurate and reflect what was actually done. All references were manually verified to exist and to be cited appropriately. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, et al. $\pi^*_{0.6}$: a VLA that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. Data quality in imitation learning. *Advances in neural information processing systems*, 36:80375–80395, 2023.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, 2025.

Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Yide Shentu, and Philipp Wu. SARM: Stage-aware reward modeling for long horizon robot manipulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=aemqAxScl9.

Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. Reward model ensembles help mitigate overoptimization. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.02743.

Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Temporal cycle-consistency learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1801–1810, 2019. URL https://openaccess.thecvf.com/content_CVPR_2019/html/Dwibedi_Temporal_Cycle-Consistency_Learning_CVPR_2019_paper.html.

Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025. URL https://arxiv.org/abs/2505.23458.

Ruhan He, Yifei Wei, Lianqing Yu, and Xianyi Zeng. Spars: Structure-informed progress-aware reward shaping for fabric manipulation learning from demonstration. *Robotics and Autonomous Systems*, 203:105499, 2026.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083. IEEE, 2019.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. RoboReward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026. URL https://arxiv.org/abs/2601.00675.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S. Huang, Luke Zettlemoyer, Dieter Fox, Yu Xiang, Anqi Li, Andreea Bobu, Abhishek Gupta, Stephen Tu, Erdem Biyik, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. In *Robotics: Science and Systems*, 2026. URL https://arxiv.org/abs/2603.02115.

Yuyang Liu, Chuan Wen, Yihang Hu, Dinesh Jayaraman, and Yang Gao. Timerewarder: Learning dense reward from passive videos via frame-wise temporal distance. In *International Conference on Machine Learning*, 2026.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-image representations and rewards for robotic control. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23301–23320, 2023a.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. VIP: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=YJ7o2wetJ2.

Yecheng Jason Ma, Joey Hejna, Ayzaan Wahid, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Jonathan Tompson, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision language models are in-context value learners. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=friHAl5ofG.

Yiming Mao, Zixi Yu, Weixin Mao, Yinhao Li, Qirui Hu, Zihan Lan, Minzhao Zhu, and Hua Chen. ARM: Advantage reward modeling for long-horizon manipulation. *arXiv preprint arXiv:2604.03037*, 2026.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3M: A universal visual representation for robot manipulation. In *6th Conference on Robot Learning*, 2022. URL https://arxiv.org/abs/2203.12601.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019. URL https://arxiv.org/abs/1910.00177.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1506.02438.

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Cheng Chi, Yuheng Ji, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, Shaoxuan Xie, Guocai Yao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. General process reward modeling for robotic reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22412–22422, 2026. URL https://arxiv.org/abs/2512.23703.

Haoran Xu, Xianyuan Zhan, Honglei Yin, and Huiling Qin. Discriminator-weighted offline imitation learning from suboptimal demonstrations. In *International Conference on Machine Learning*, pp. 24725–24742. PMLR, 2022.

Jiazhi Yang, Kunyang Lin, Jinwei Li, Wencong Zhang, Tianwei Lin, Longyan Wu, Zhizhong Su, Hao Zhao, Ya-Qin Zhang, Li Chen, Ping Luo, Xiangyu Yue, and Hongyang Li. RISE: Self-improving robot policy with compositional world model. In *Robotics: Science and Systems*, 2026. URL https://arxiv.org/abs/2602.11075.

Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. XIRL: Cross-embodiment inverse reinforcement learning. In *5th Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=RO4DM85Z4P7.

Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Fuxian Huang, Haoran Zhang, Ming Zhou, Shengzhe Zhang, Litao Liu, Sixu Lin, and Jiangmiao Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint arXiv:2509.15937*, 2025. URL https://arxiv.org/abs/2509.15937.

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh A. Sontakke, Joseph J. Lim, Jesse Thomason, Erdem Bıyık, and Jesse Zhang. ReWiND: Language-guided rewards teach robot policies without new demonstrations. In *9th Conference on Robot Learning*, 2025.

A Appendix

A.1 Hardware Setup

All real-robot experiments are carried out on the bimanual tabletop platform shown in Figure 6. The system consists of:

- Two ARX robot arms, built by the manufacturer ARX Robotics, mounted side by side in a bimanual configuration.
- Three Orbbec Astra Pro RGB cameras: one mounted on each wrist and a third mounted on the head of the platform for an overview of the workspace.



Figure 6: **The bimanual platform used for data collection and policy evaluation.** Two ARX arms are mounted on a fixed torso facing the table, each carrying a wrist camera, with a third camera on the head looking down on the workspace. The scene shown is the towel-folding task.

Both expert demonstrations and human-correction episodes are collected through a leader–follower teleoperation system: the operator drives a pair of kinematically matched leader arms, and the follower arms on the platform reproduce the motion. The same interface is used during autonomous execution, which allows the operator to take over at any moment and hand control back once the robot has recovered; the frames recorded while the operator is in control are exactly the human-correction data used throughout this paper.

Episodes are recorded at 20 fps and contain the three synchronized camera streams (head, left wrist, right wrist) together with the robot proprioception. We use the *absolute* end-effector pose as both the state and the action representation, i.e. the Cartesian position and orientation of each gripper expressed in the robot base frame, rather than joint angles or relative displacements, so that the advantage model and the policy operate in the same task-space coordinates across all tasks.

The pick-and-place task analyzed in Appendix A.7 is the one exception to this setup: it runs on a single Franka arm and is included as a short-horizon control condition.

A.2 Task Descriptions and Training Data Composition

To evaluate STEAM across different manipulation horizons, contact patterns, and coordination requirements, we design a set of real-world tasks, shown in Figure 7. Towel folding (5 stages), chip checkout (8 stages), table clearing (21 stages), and cola restocking (4 stages) run on an ARX dual-arm robot. For each task we collect a dataset that mixes expert demonstrations, autonomous rollouts, and human correction episodes in varying proportions. Table 5 summarizes the resulting dataset composition, and the step-by-step procedure of each task is given below.

Towel Folding. This long-horizon task utilizes an ARX dual-arm robot to fold a towel. A successful execution requires completing all five consecutive stages without severe fabric misfolding. The

Table 5: **Training data composition.** Number of episodes of each data source per task. Expert demonstrations are collected by teleoperation, autonomous rollouts are generated by a behavior-cloning policy trained on the expert data, and human-correction episodes are autonomous rollouts in which the operator took over and recovered the task.

Task	Stages	Expert demos	Autonomous rollouts	Human corrections	Total
Towel Folding	5	240	125	20	385
Chip Checkout	8	200	63	20	283
Table Clearing	21	203	94	42	339
Cola Restocking	4	89	49	27	165

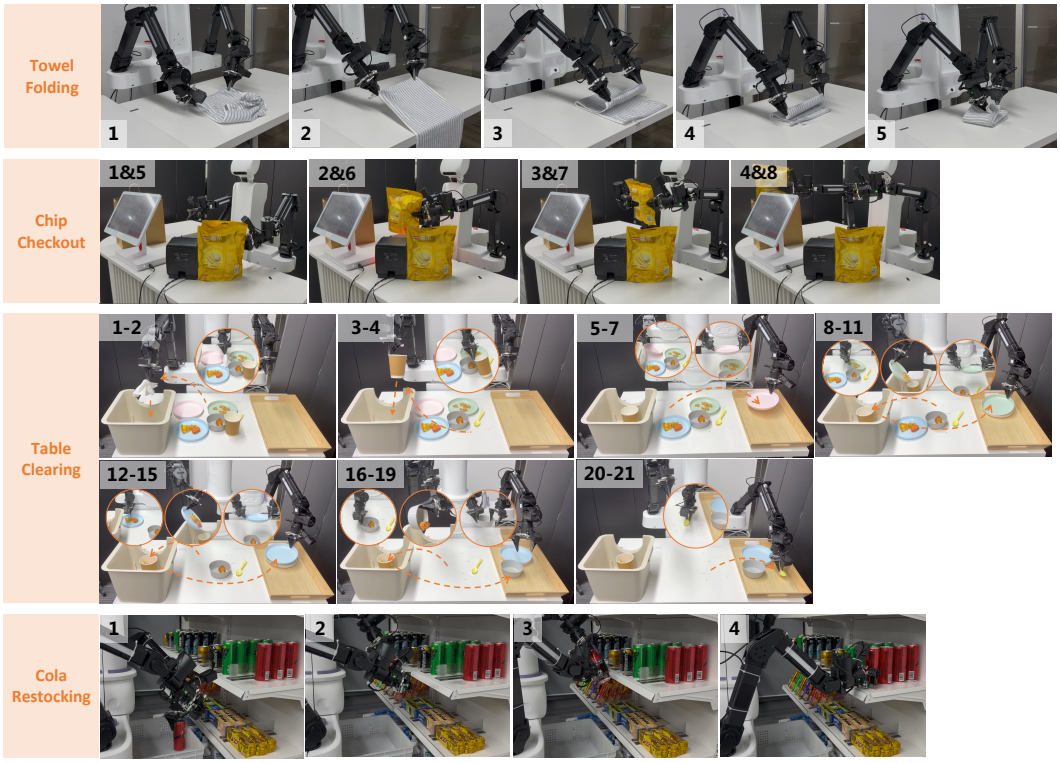


Figure 7: **Robot setup and tasks.** The four bimanual tasks used throughout the paper — towel folding, chip checkout, table clearing, and cola restocking — with the stages of each task and the sub-step indices each panel covers.

dataset collected for this task includes 240 expert demonstrations, 125 autonomous rollouts, and 20 human correction episodes. The folding sequence is divided into five stages:

- **Stage 1 (Pick up):** The robot arms reach down and grasp the corners of the towel on the table.
- **Stage 2 (Flatten):** The arms lift and stretch the towel to flatten the fabric, preparing it for folding.
- **Stage 3 (First fold):** The robot executes the first longitudinal fold.
- **Stage 4 (Second fold):** The robot executes the second fold, reducing the towel's width.
- **Stage 5 (Third fold):** The robot performs the final fold to complete the folding sequence.

Chip Checkout. This dual-arm coordination task simulates a retail checkout process, where two bags of chips must be scanned and bagged sequentially. The training dataset for this task includes 200 expert demonstrations, 63 autonomous rollouts, and 20 human correction episodes. The task involves 8 stages:

- **Stage 1 (Pick up first bag):** The left arm reaches and grasps the first bag of chips from the counter.
- **Stage 2 (Scan first bag):** The left arm moves the chip bag to the barcode scanner and performs the scanning action.
- **Stage 3 (Handover first bag):** The left arm coordinates with the right arm to handover the first chip bag.
- **Stage 4 (Bag first bag):** The right arm receives the chip bag and places it into the shopping bag.
- **Stage 5 (Pick up second bag):** The left arm reaches and grasps the second bag of chips.
- **Stage 6 (Scan second bag):** The left arm moves the second chip bag to the scanner for barcode recognition.
- **Stage 7 (Handover second bag):** The left arm handovers the second chip bag to the right arm.
- **Stage 8 (Bag second bag):** The right arm receives the second chip bag and completes the bagging process.

Table Clearing. This is the longest task in our suite: the dual-arm robot has to clear a dinner table of seven objects, discarding the disposable items, emptying the leftover food from every container, and stacking the containers on the side. It is also the most demanding one in terms of coordination, since every plate and bowl is handed over from the right arm to the left arm before being put down. The training dataset contains 203 expert demonstrations, 94 autonomous rollouts, and 42 human correction episodes. The task is divided into 21 stages:

- **Stage 1 (Pick up tissue):** The right arm grasps the used tissue from the table.
- **Stage 2 (Discard tissue):** The right arm drops the tissue into the trash bin.
- **Stage 3 (Pick up paper cup):** The right arm grasps the paper cup.
- **Stage 4 (Discard paper cup):** The right arm drops the paper cup into the trash bin.
- **Stage 5 (Pick up pink plate):** The right arm grasps the pink plate.
- **Stage 6 (Handover pink plate):** The right arm hands the pink plate over to the left arm.
- **Stage 7 (Place pink plate):** The left arm places the pink plate onto the stacking area.
- **Stage 8 (Pick up green plate):** The right arm grasps the green plate.
- **Stage 9 (Empty green plate):** The right arm tips the leftover food from the green plate into the trash bin.
- **Stage 10 (Handover green plate):** The right arm hands the green plate over to the left arm.
- **Stage 11 (Place green plate):** The left arm places the green plate onto the stack.
- **Stage 12 (Reposition and pick up blue plate):** The right arm pushes the blue plate into a graspable pose and picks it up.
- **Stage 13 (Empty blue plate):** The right arm tips the leftover food from the blue plate into the trash bin.
- **Stage 14 (Handover blue plate):** The right arm hands the blue plate over to the left arm.
- **Stage 15 (Place blue plate):** The left arm places the blue plate onto the stack.
- **Stage 16 (Pick up grey bowl):** The right arm grasps the grey bowl.
- **Stage 17 (Empty grey bowl):** The right arm tips the leftover food from the grey bowl into the trash bin.
- **Stage 18 (Handover grey bowl):** The right arm hands the grey bowl over to the left arm.
- **Stage 19 (Place grey bowl):** The left arm places the grey bowl onto the stack.
- **Stage 20 (Pick up yellow spoon):** The right arm grasps the yellow spoon.
- **Stage 21 (Place yellow spoon):** The right arm places the yellow spoon at its target position, completing the task.

Cola Restocking. This task requires the dual-arm robot to transfer a cola bottle from a crate to a shelf, which demands active collision avoidance. The training dataset consists of 89 expert demonstrations, 27 human corrections, and 49 autonomous rollouts. The task is divided into 4 stages:

- **Stage 1 (Pick up):** The left arm grasps a cola bottle from the inventory crate.
- **Stage 2 (Camera adjustment):** The left arm rotates its wrist to adjust the wrist-camera's viewpoint, preventing potential hardware and camera collisions between the two arms.
- **Stage 3 (Handover):** The left arm handovers the cola bottle to the right arm.
- **Stage 4 (Place):** The right arm moves the cola bottle and places it stably onto the target shelf.

A.3 Per-Task Advantage Visualizations

Section 4.1 compares STEAM against the baselines on two towel-folding episodes. Here we show the frame-level A_{STEAM} curve on all four tasks and on all four episode types we collect (expert demonstrations, successful rollouts, failed rollouts, and human-correction episodes) to document what the signal looks like across the full range of data STEAM is applied to.

An effective advantage signal must reflect the nuances of task execution: remaining high during proficient segments, decreasing during hesitation or failure, and recovering upon effective progress.

Towel Folding. Figure 8 visualizes the four representative episode types in the towel folding task. For expert demonstrations, the advantage values remain consistently high throughout the majority of the episode, experiencing only minor and transient drops around frames that involve necessary retries or slight adjustments. Successful rollouts show lower and more fluctuating advantages than expert demonstrations, reflecting slower yet still effective task execution. In contrast, failed rollouts rapidly drop to near-zero advantage after entering failure states, where the policy becomes stuck without recovering. Human correction episodes present an initial drop similar to failures, but critically, show a clear recovery of advantage after human intervention. STEAM therefore provides a fine-grained, frame-level assessment capable of distinguishing proficient execution from hesitation, failure, and recovery within a single episode.

Chip Checkout. For the **chip checkout** task (Figure 9), the advantage curves accurately capture the complex dual-arm coordination and timing. In the *expert demonstration*, a brief grasping failure by the right arm during the handover phase (around frames 200–300) causes a temporary drop in the advantage; once a successful retry is executed, the advantage recovers to high values. In the *successful rollout*, the overall execution speed is visibly slower, particularly during the handover phase (indicated by the dark blue region), leading to a more fluctuating advantage curve. The *failed rollout* exhibits a persistent decline, with the advantage remaining extremely low after frame 400. In the *human-correction episode*, the policy stagnates around frame 800 and fails to grasp the second chip bag, causing the advantage to dip significantly into negative values. Following human takeover and corrective action, the task resumes progress, and the advantage recovers.

Table Clearing. For the **table clearing** task (Figure 10), the advantage curves track pauses and stalls across the 21 stages. In the *expert demonstration*, after the green plate is put down, the right arm pauses out of view for about 7 s (frames 2185–2325) before reaching for the blue plate in Stage 12; the advantage falls to around -0.9 during the pause and returns to 0.7–0.9 once the arm reaches and lifts the plate. In the *successful rollout*, the policy clears the blue plate, the grey bowl, and the spoon at roughly half the expert pace, and the advantage stays at moderate positive values (0.2–0.6) apart from brief dips. In the *failed rollout*, the grey bowl is emptied and handed over to the left arm, but the left arm then holds it in the air and never puts it down (Stage 19); the advantage falls from 0.3–0.7 to below zero after frame 4950 and stays low until the episode ends. In the *human-correction episode*, the robot fails to move at the start of Stage 20 and the advantage stays at -1; once the operator takes over and grasps the spoon, the advantage recovers to high values.

Cola Restocking. For the **cola restocking** task (Figure 11), we observe similar temporal sensitivity to coordination and placement errors. In the *expert demonstration*, the robot fails to place the cola bottle onto the narrow shelf due to initial misalignment between frames 400 and 600, causing a

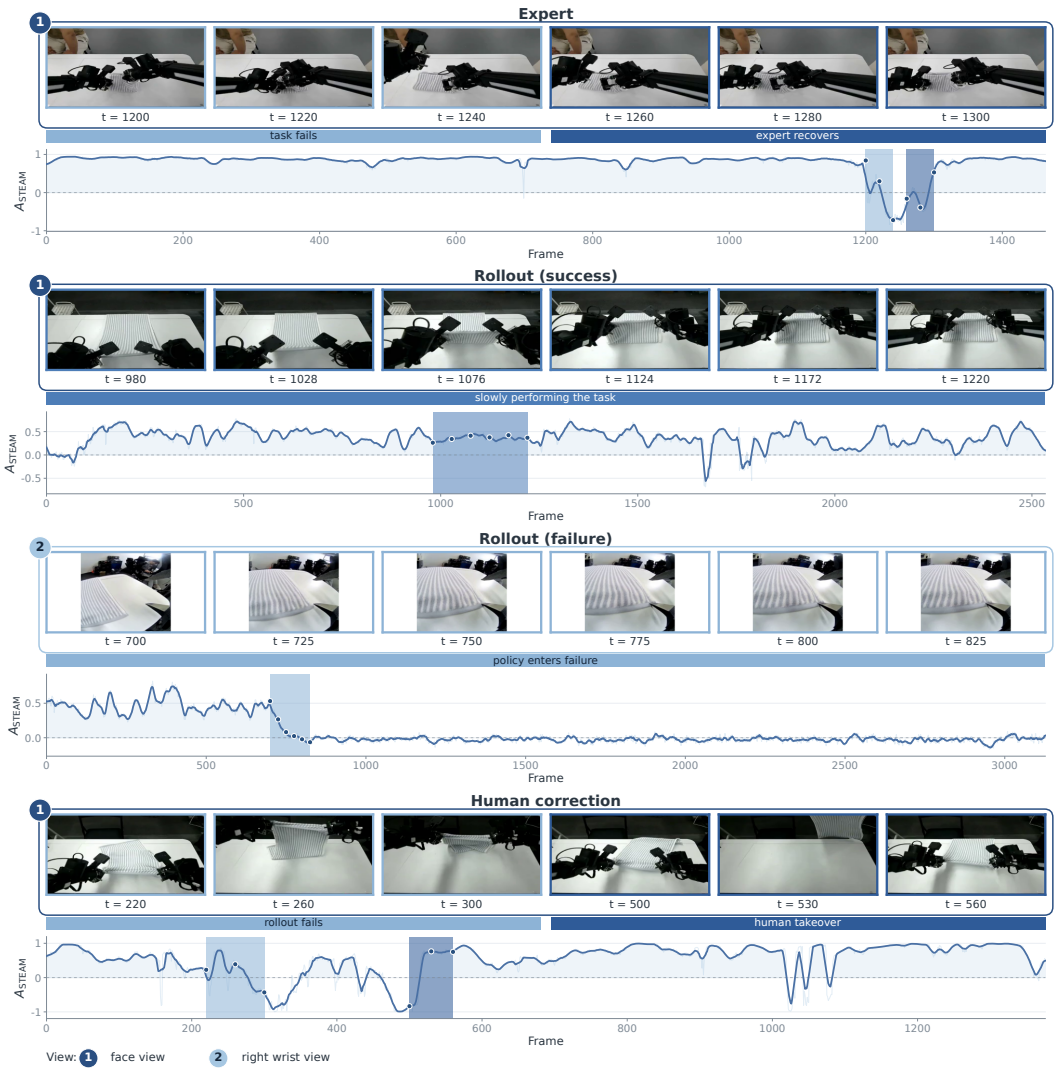


Figure 8: **Towel folding advantage curves.** Frame-level A_{STEAM} on four representative episode types: expert demonstrations, successful rollouts, failed rollouts, and human correction episodes. Images on top show corresponding frames from each episode, while shaded regions highlight segments with retry, slow progress, failure, or human takeover.

temporary advantage drop. After a successful realignment and retry after frame 600, the advantage rises back to a high level. In the *successful rollout*, the right arm makes multiple consecutive attempts to grasp the cola during the handover phase, leading to slower progress and a corresponding decline in the advantage curve. In the *failed rollout*, the advantage drops and remains consistently low after frame 200, reflecting unrecoverable policy errors.

A.4 Advantage Distributions Across Data Sources

Section 4.1 scores each episode by its cumulative advantage. This section reports the two distributions behind that number: how frame-level advantages are distributed within each data source, and how the per-episode sums separate.

Figure 12 presents the probability density of frame-level A_{STEAM} across different data sources. Consistent trends are observed across all domains: expert demonstrations exhibit a strong concentration near $+1$, indicating that most frames correspond to high-quality execution that contributes directly to task completion. Successful rollouts systematically shift toward lower positive values,

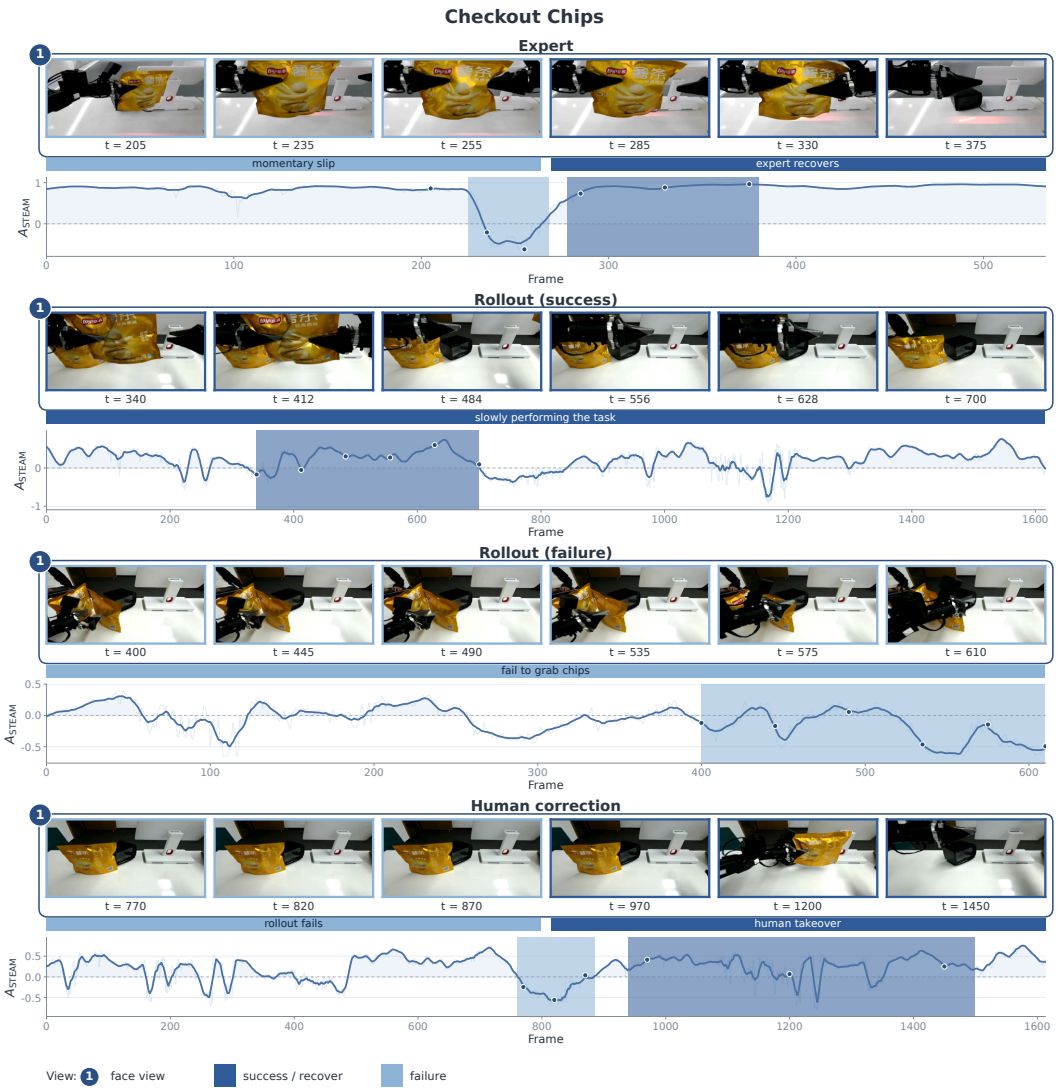


Figure 9: **Chip checkout advantage curves.** Expert, successful-rollout, failed-rollout, and human-correction episodes.

reflecting slower or more cautious execution. In contrast, failed rollouts place a prominent density peak around or below zero, which corresponds to the stagnation and lack of progress after a failure occurs. Human-correction episodes exhibit a broader, more distributed profile, combining the lower positive advantages from slow autonomous phases with the higher advantages recovered immediately after human takeover.

Figure 13 reports the episode-wise sum of frame-level advantages, the quantity clustered in Table 1. While successful episodes yield comparable cumulative advantages, failed rollouts are clearly separated by significantly lower sums. Notably, cumulative advantages are higher for towel folding and chip checkout than for cola restocking, particularly among failures. This discrepancy stems from their failure modes: towel and chip failures typically occur in middle or late stages, allowing positive advantages to accumulate early on, whereas cola failures occur during the initial grasp, preventing any advantage accumulation. The absolute scale of the sum therefore carries task-specific information and is not comparable across tasks, which is why Table 1 clusters episodes within a single task.

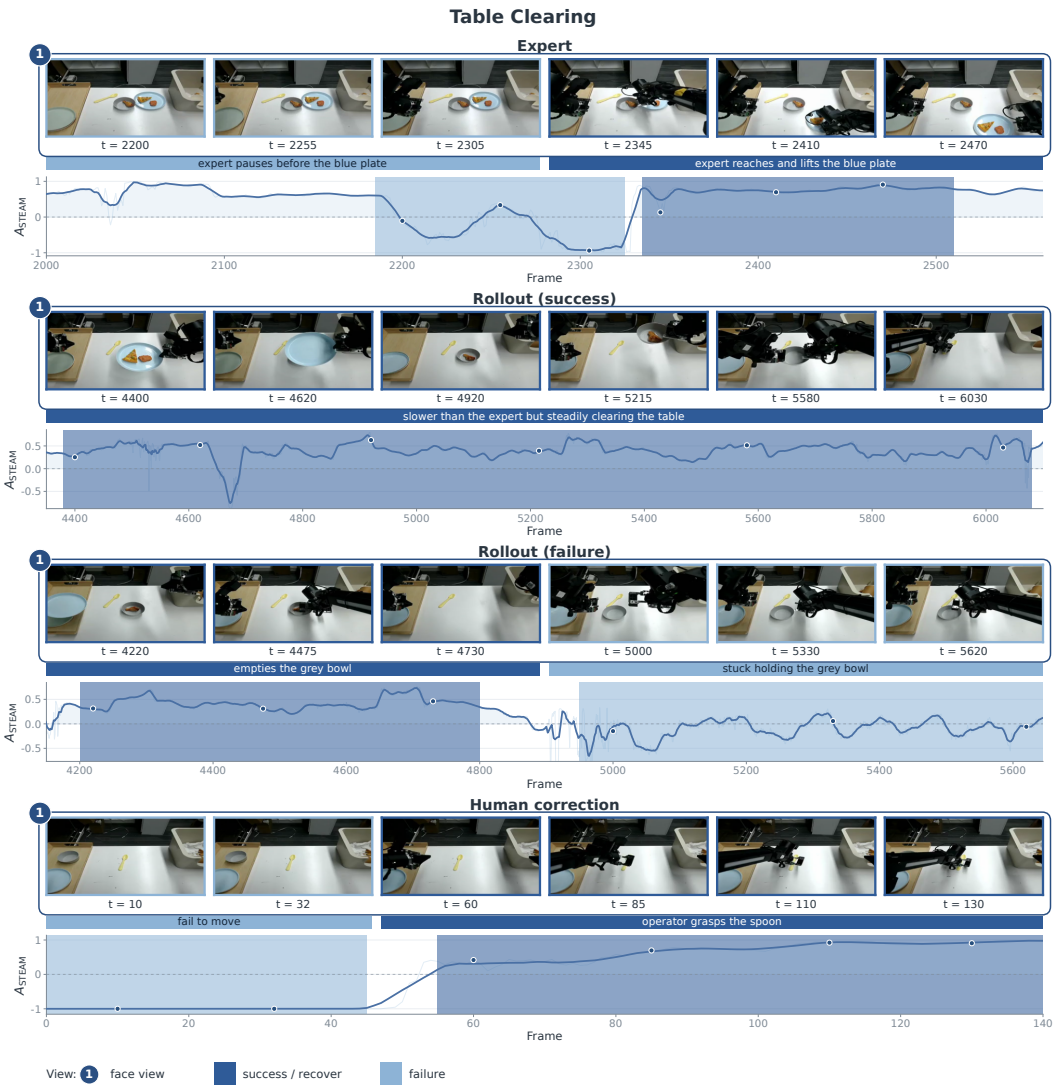


Figure 10: **Table clearing advantage curves.** Expert, successful-rollout, failed-rollout, and human-correction episodes.

A.5 Implementation Details: Advantage Modeling and CFGRL Policy Extraction

This section expands the policy-training step sketched in Equation 6: how the frame-level advantages learned by STEAM enter the classifier-free guidance reinforcement learning framework, and how training and inference proceed once they do.

CFGRL Training. Once the STEAM advantage predictors are trained on the expert dataset, we compute the advantage score $A_{\text{STEAM}}(f_{k,i};\Theta)$ for each frame $f_{k,i} \in \mathcal{D}_{\text{exp}} \cup \mathcal{D}_{\text{nexp}}$ using a fixed lookahead H. Here, $\mathcal{D}_{\text{exp}}$ represents the expert demonstration dataset, and $\mathcal{D}_{\text{nexp}}$ denotes the non-expert dataset, comprising both autonomous rollouts and human corrections. We then map these continuous advantages into binary optimality labels $o \in \{0,1\}$ by thresholding each data pool separately:

$$o = \begin{cases} \mathbf{1}[A_{\text{STEAM}}(f_{k,i};\Theta) \geq \delta^{\text{exp}}], & f_{k,i} \in \mathcal{D}_{\text{exp}}, \\ \mathbf{1}[A_{\text{STEAM}}(f_{k,i};\Theta) \geq \delta^{\text{non-exp}}], & f_{k,i} \in \mathcal{D}_{\text{nexp}}, \end{cases} \tag{12}$$

where δ^{exp} and $\delta^{\text{non-exp}}$ label the top-q^{exp} and top-$q^{\text{non-exp}}$ fractions of $\mathcal{D}_{\text{exp}}$ and $\mathcal{D}_{\text{nexp}}$ as optimal.

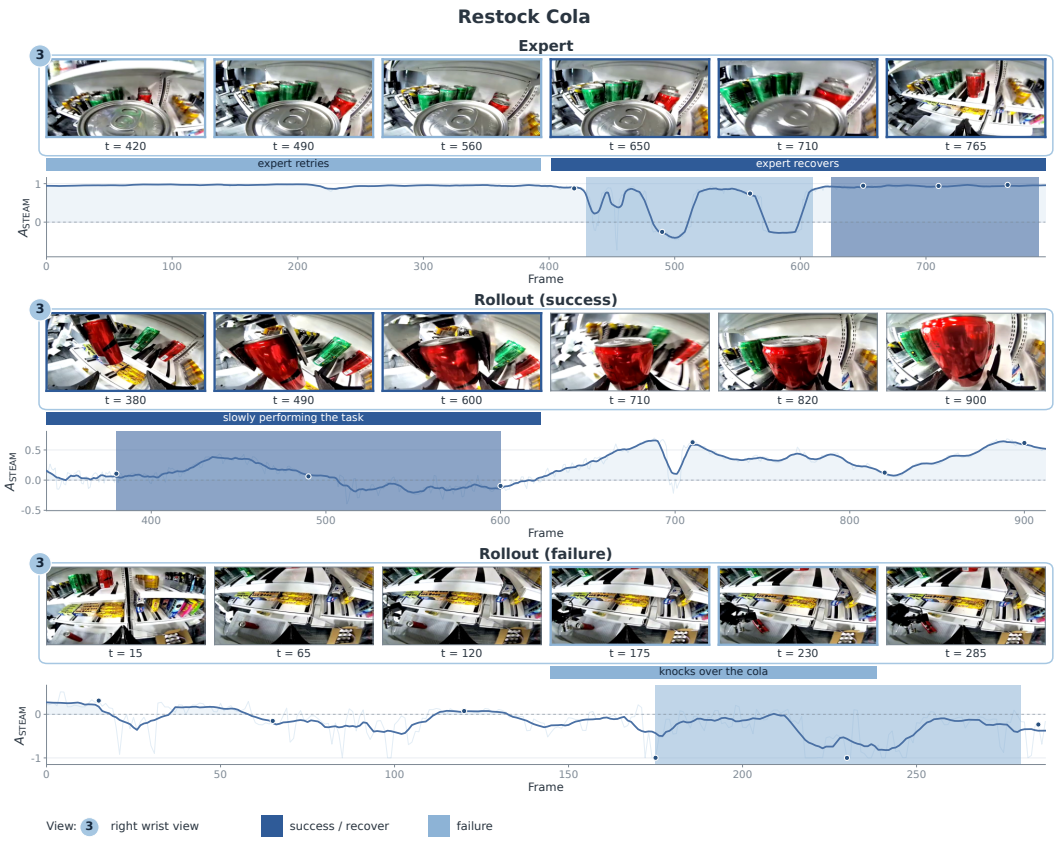
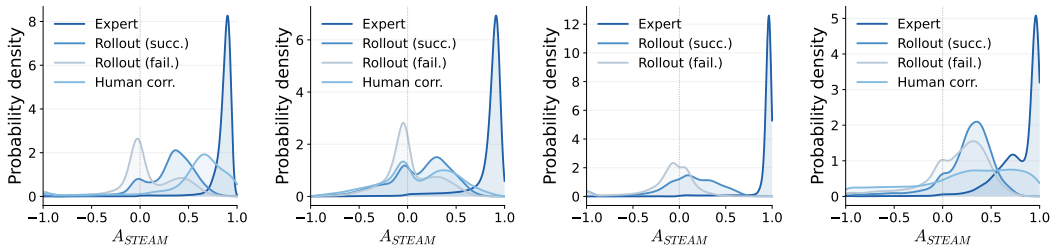


Figure 11: **Cola restocking advantage curves.** Expert, successful-rollout, and failed-rollout episodes.



(a) Towel Folding (b) Chip Checkout (c) Cola Restocking (d) Table Clearing

Figure 12: Probability density of frame-level A_{STEAM} across data types.

These binary optimality labels condition a flow-matching policy $v_\phi(a_t, t, s, o)$ parameterizing a velocity field. During training, we construct a noisy action interpolant a_t at timestep t as:

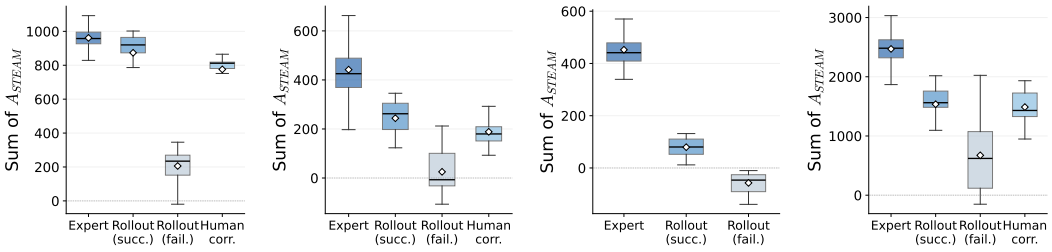
$$a_t = (1 - t)a_0 + ta, \tag{13}$$

where a is the target action chunk from the dataset, $a_0 \sim \mathcal{N}(0, I)$ is standard Gaussian noise, and t is sampled from a uniform distribution over $[0, 1]$, denoted as $t \sim \mathcal{U}[0, 1]$.

To support classifier-free guidance, we randomly drop the conditioning label o with a dropout probability p_{drop}. The conditioned label input $\tilde{o}$ is defined as $\tilde{o} = \varnothing$ (representing the unconditioned state) with probability p_{drop}, and $\tilde{o} = o$ otherwise. The resulting flow matching objective is formulated as:

$$\mathcal{L}_\pi(\phi) = \mathbb{E}_{(s,a,o)\sim\mathcal{D}_{\text{exp}}\cup\mathcal{D}_{\text{nexp}},\ t\sim\mathcal{U}[0,1],\ a_0\sim\mathcal{N}(0,I)} \left[\left\| v_\phi(a_t, t, s, \tilde{o}) - (a - a_0) \right\|^2 \right], \tag{14}$$

where s denotes the policy's observation input of the current frame $f_{k,i}$.



(a) Towel Folding (b) Chip Checkout (c) Cola Restocking (d) Table Clearing

Figure 13: Sum of frame-level A_{STEAM} over each episode across data types.

CFGRL Inference. During deployment, we generate guided actions by integrating the ODE defined by the velocity field. Starting from pure Gaussian noise $a_0 \sim \mathcal{N}(0, I)$ at $t = 0$, we perform T fixed-size Euler integration steps with step size $\Delta t = 1/T$:

$$a_{t+\Delta t} = a_t + \Delta t \cdot v_{\text{cfg}}(a_t, t, s), \tag{15}$$

where the guided velocity field $v_{\text{cfg}}(a_t, t, s)$ is computed by extrapolating towards the high-optimality label ($o = 1$):

$$v_{\text{cfg}}(a_t, t, s) = v_\phi(a_t, t, s, \varnothing) + w\Big[v_\phi(a_t, t, s, o{=}1) - v_\phi(a_t, t, s, \varnothing)\Big]. \tag{16}$$

Here, $w \geq 1$ is the guidance scale that controls the strength of the optimality bias. Setting $w > 1$ actively steers the generated actions toward trajectories classified as highly progress-advancing by STEAM. After T integration steps, the resulting endpoint a_1 is sent to the robot controller as the target action chunk.

Pseudocode. We provide the formal algorithmic procedures for our framework. Specifically, Algorithm 1 details the self-supervised training process for the STEAM advantage predictor ensemble. Algorithm 2 describes the policy learning pipeline, showing how learned advantages are translated into binary optimality labels to guide a classifier-free flow-matching policy.

Algorithm 1 Training STEAM

Input: Expert dataset $\mathcal{D}_{\text{exp}}$, ensemble size M, bin count N, max offset $k_{\max}$, reference length $L_{\max}$, offset bound $\tilde{\Delta}_{\max}$
Output: Trained ensemble $\Theta = \{\theta_m\}_{m=1}^{M}$

for $m = 1$ **to** M **do**
  Initialize θ_m with an independent random seed
  repeat
    Sample expert episode $\tau_k = (f_{k,1}, \ldots, f_{k,L_{\tau_k}})$ with instruction ℓ from $\mathcal{D}_{\text{exp}}$
    Sample offset $\Delta \sim \text{Uniform}\{\pm 1, \ldots, \pm k_{\max}\}$
    Sample frame index $i \sim \text{Uniform}\{\, i' : 1 \leq i' \leq L_{\tau_k},\ 1 \leq i' + \Delta \leq L_{\tau_k} \,\}$; set $j \leftarrow i + \Delta$
    $\tilde{\Delta}_{\tau_k}(i, j) \leftarrow (j - i) \cdot L_{\max}/L_{\tau_k}$ ▷ Eq. 2
    $y \leftarrow \tilde{\Delta}^{\mathcal{B}}_{\tau_k}(i, j)$ ▷ one-hot bin of $\text{clip}(\tilde{\Delta}_{\tau_k}(i, j), \pm\tilde{\Delta}_{\max})$
    $\mathcal{L} \leftarrow H\big(y,\ p_{\theta_m}(\Delta \mid f_{k,i}, f_{k,j}, \ell)\big)$ ▷ Eq. 3
    Update θ_m via gradient step on $\mathcal{L}$
  until convergence
end for
return $\Theta = \{\theta_m\}_{m=1}^{M}$

Hyperparameters. Table 6 lists the hyperparameters of the STEAM advantage predictor and the CFGRL policy training, which are shared by all four tasks. The lookahead H is the one setting that is worth calling out: it fixes the temporal span over which progress is measured, and it has to be long enough for the scene to change visibly. Our default $H = 32$ spans 1.6 s at the 20 fps the data

Algorithm 2 CFGRL Policy Learning with STEAM

Input: Trained STEAM ensemble $\Theta = \{\theta_m\}_{m=1}^{M}$, expert dataset $\mathcal{D}_{\text{exp}}$, non-expert dataset $\mathcal{D}_{\text{nexp}}$, policy lookahead H, top fractions q^{exp} and $q^{\text{non-exp}}$, conditioning dropout p_{drop}
Output: Trained flow-matching policy v_ϕ

Stage 1: label data with STEAM

for each $(s_t, a_t, s_{t+H}) \in \mathcal{D}_{\text{exp}} \cup \mathcal{D}_{\text{nexp}}$ **do**
  $A_{\text{STEAM}}(s_t, s_{t+H}) \leftarrow \min_{m=1,\dots,M} A(s_t, s_{t+H};\, \theta_m)$ ▷ Eqs. 4, 5
end for
$\delta^{\text{exp}} \leftarrow$ top-q^{exp} threshold of $\{A_{\text{STEAM}}(s_t, s_{t+H})\}$ over $\mathcal{D}_{\text{exp}}$
$\delta^{\text{non-exp}} \leftarrow$ top-$q^{\text{non-exp}}$ threshold of $\{A_{\text{STEAM}}(s_t, s_{t+H})\}$ over $\mathcal{D}_{\text{nexp}}$
for each $(s_t, a_t, s_{t+H}) \in \mathcal{D}_{\text{exp}} \cup \mathcal{D}_{\text{nexp}}$ **do**
  if $(s_t, a_t, s_{t+H}) \in \mathcal{D}_{\text{exp}}$ **then**
    $o_t \leftarrow \mathbf{1}[A_{\text{STEAM}}(s_t, s_{t+H}) \geq \delta^{\text{exp}}]$
  else
    $o_t \leftarrow \mathbf{1}[A_{\text{STEAM}}(s_t, s_{t+H}) \geq \delta^{\text{non-exp}}]$
  end if
end for
$\mathcal{D}_o \leftarrow \{(s_t, a_t, o_t)\}$ ▷ Eq. 12

Stage 2: classifier-free guidance policy training

Initialize flow-matching policy parameters ϕ
repeat
  Sample $(s, a, o) \sim \mathcal{D}_o$
  Sample flow time $t \sim \mathcal{U}[0, 1]$ and noise $a_0 \sim \mathcal{N}(0, I)$
  $a_t \leftarrow (1 - t)\, a_0 + t\, a$
  $\tilde{o} \leftarrow \varnothing$ with probability p_{drop}, otherwise $\tilde{o} \leftarrow o$
  $\mathcal{L}_\pi \leftarrow \left\| v_\phi(a_t, t, s, \tilde{o}) - (a - a_0) \right\|^2$ ▷ Eq. 14
  Update ϕ via gradient step on $\mathcal{L}_\pi$
until convergence
return v_ϕ

is recorded at; halving it to $H = 16$ (0.8 s) lowers the towel-folding success rate from 92.3% to 80.0%, because over such a short interval the visual difference between two frames is too subtle to signify progress.

Table 6: Hyperparameters for STEAM advantage prediction and CFGRL policy optimization, shared by all four tasks.

Hyperparameter	**Value**
Maximum offset	32
Number of bins N	32
Ensemble size M	3
Advantage lookahead H	32
Top fraction q^{exp} over $\mathcal{D}_{\text{exp}}$	0.8 (top 80%)
Top fraction $q^{\text{non-exp}}$ over $\mathcal{D}_{\text{nexp}}$	0.3 (top 30%)
Conditioning dropout p_{drop}	0.1
Guidance scale w	2.5
Learning rate	5×10^{-5}
Training steps	30,000
Batch size	512

A.6 Continuous Advantage Weighting as an Alternative to CFGRL

CFGRL turns A_{STEAM} into a binary optimality label and conditions the policy on it (Eq. 12). A natural alternative is to keep the advantage continuous and let it reweight the imitation loss directly, as in advantage-weighted regression (Peng et al., 2019), where each transition is weighted by $\exp(A/\beta)$.

This is a strictly finer-grained use of the same signal, so it is worth asking whether the binarization in CFGRL throws information away.

Table 7 answers this on towel folding, with the advantage model, the dataset, and the policy backbone held fixed and only the policy-extraction rule changed. AWR recovers a large part of the gap over behavior cloning, from 33.3% to 73.3%, which confirms that the STEAM advantage is informative under either extraction rule. But it remains 19 points below CFGRL.

We attribute the gap to the bounded range of the advantage. Since $A_{\mathrm{STEAM}} \in [-1, 1]$ by construction, $\exp(A/\beta)$ spans a narrow band of weights, and continuous weighting is not sharp enough to filter failed transitions: a stalled or regressing frame is down-weighted, but it still contributes a non-trivial gradient, and in datasets where such frames are a large fraction of the non-expert data their aggregate pull is substantial. Quantile thresholding instead removes those frames from the optimal mode entirely, and the guidance scale w then controls at inference time how strongly the policy is pushed toward that mode, decoupling the strength of the optimality bias from the scale of the advantage. This is consistent with the finding in RECAP (Amin et al., 2025) that advantage-conditioned policy extraction outperforms advantage-weighted regression.

Table 7: **Policy extraction rule on towel folding.** Success rate (%) with the STEAM advantage held fixed and only the extraction rule changed.

	BC	**STEAM (AWR)**	**STEAM (CFGRL, ours)**
Success rate (%)	33.3	73.3	**92.3**

A.7 When Expert-Only Filtering Does Not Help: The Pick-and-Place Case

STEAM improves policy performance by filtering out low-quality, non-progressing frames and emphasizing the most task-advancing segments. The size of that benefit depends on what there is to filter. When the available data are both scarce and already of high quality, filtering removes samples that a supervised policy could still have learned from, effectively shrinking an already small training set. We study this regime on the **pick-and-place** task, a short-horizon task on a single Franka arm in which the gripper relocates objects between two plates. Expert trajectories here are highly consistent, and the dataset contains 50 expert demonstrations and 594 autonomous rollouts. The task has 2 stages:

- **Stage 1 (Grasp):** The gripper descends to grasp a target object from the left plate.
- **Stage 2 (Place):** The arm transfers and places the object onto the designated position on the right plate.

Because the episodes are much shorter, we halve the maximum offset, the bin count N, and the lookahead H to 16 on this task; all other hyperparameters follow Table 6.

Figure 14 reports the policy performance, and the four-way comparison isolates two effects. First, STEAM trained on expert demonstrations only (STEAM (Exp)) slightly underperforms the already-strong BC baseline (63.8%). The expert set is small (50 trajectories) and already high-quality, so filtering it only removes useful supervision and lowers performance. This near-optimality is directly visible in Figure 16, where the frame-level A_{STEAM} of expert demonstrations concentrates sharply near $+1$, leaving almost no low-quality mass to remove. Second, once autonomous rollouts are added (STEAM (Full)), STEAM selects the high-quality frames from this rollout data and raises the success rate to 80%, surpassing both BC and RECAP.

The limitation is therefore specific, and narrower than the section title suggests on its own: what does not help on this task is filtering an expert set that is already clean and small. Given the full data it is applied to everywhere else in this paper, expert demonstrations together with autonomous rollouts and human corrections, STEAM still outperforms BC on pick-and-place, by 16 points. The regime to be careful about is not short-horizon tasks in general but datasets that offer nothing to prune.

Figure 15 shows the frame-level advantage curves on this task for completeness. Even at this much shorter horizon the signal remains informative. The *expert demonstration* shows smooth, proficient

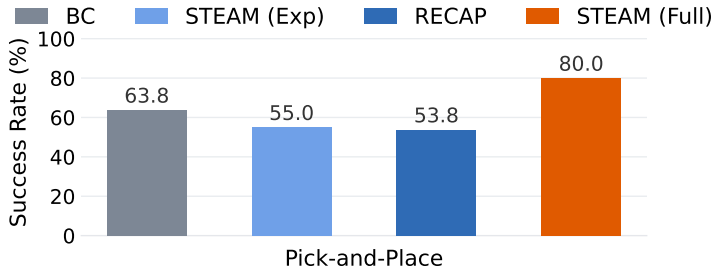


Figure 14: **Policy performance on the pick-and-place task.** STEAM (Exp) filters expert demonstrations only, whereas STEAM (Full) additionally uses autonomous rollouts. BC vs. STEAM (Exp) shows that filtering the already-clean expert data alone does not help, while RECAP vs. STEAM (Full) shows that STEAM captures the useful signal in rollout data more effectively than RECAP.

execution, maintaining a high advantage near $+1$ throughout. In the *successful rollout*, the robot makes multiple unsuccessful attempts to grasp the target object (a toy duck) between frames 40 and 100, pushing the advantage below zero. After a successful grasp after frame 100, the advantage recovers to nearly $+1$. In the *failed rollout*, the advantage is initially high (frames 0–50) as the gripper approaches the duck, but plunges below zero once the robot repeatedly fails to grasp the object.

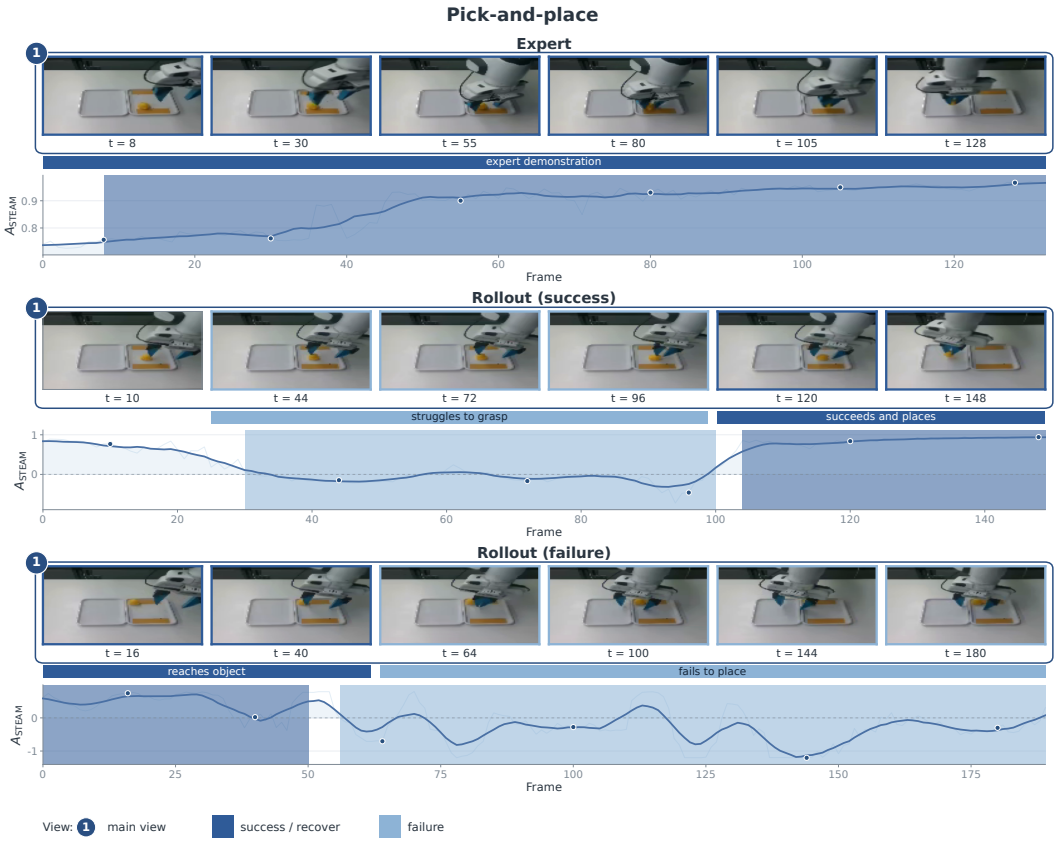
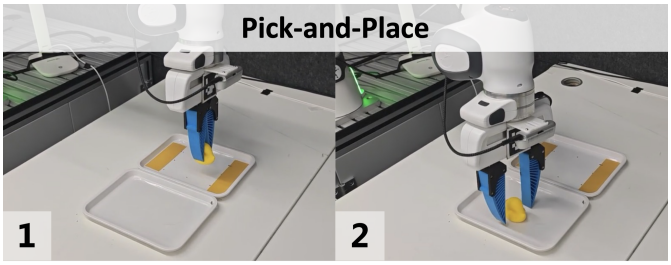
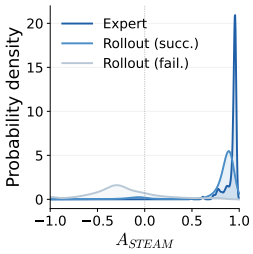


Figure 15: **Pick-and-place advantage curves.** Expert, successful-rollout, and failed-rollout episodes.



(a) Task setup



(b) Advantage density

Figure 16: **The pick-and-place task and its frame-level advantage density. (a)** Pick-and-place is a short, two-stage task (grasp the object, then place it), so expert trajectories are highly consistent. **(b)** The frame-level A_{STEAM} of expert demonstrations concentrates sharply near $+1$, indicating that trajectories are already near-optimal and leave little low-quality mass for filtering to prune. This is why STEAM (Exp) slightly underperforms BC in this scarce, already-clean regime.