

Reward-Dagger: Robot-Gated Interactive Imitation Learning with General-Purpose Progress-Based Reward Models

Ryan Li^{*1}, Yigit Korkmaz^{*2}, Erdem Bıyık²

¹Department of Mathematics, University of Southern California

²Thomas Lord Department of Computer Science, University of Southern California

Abstract—Recent advances in robot learning have enabled generalist control policies capable of completing a wide range of tasks. However, their performance degrades when deployed in unseen environments, making it critical to detect failures and teach recovery behaviors. Existing runtime monitoring methods often require task- and policy-specific training or hyperparameter tuning, limiting cross-task deployment and introducing additional overhead during iterative policy updates. We present Reward-Dagger, a robot-gated interactive imitation learning framework that uses dense progress signals from a general-purpose reward model to determine when human intervention is needed. Our approach is agnostic to the underlying policy architecture, requires no access to policy internals, and can be applied across tasks without retuning the gating mechanism. Our results show that Reward-Dagger achieves a better failure-detection accuracy–latency tradeoff than existing runtime monitoring baselines. Across eight simulated and real-world tasks, Reward-Dagger consistently improves the downstream policy’s autonomous success rate throughout interactive learning and achieves strong return on human effort, outperforming the baselines in most settings. Importantly, the same gating configuration is used across tasks without task-specific hyperparameter tuning, demonstrating transfer across tasks, environments, and policy architectures. Code and videos are available at <https://liralab.usc.edu/reward-dagger>.

I. INTRODUCTION

Modern learning-based robot policies, ranging from task-specific models [1, 2] to generalist vision-language-action (VLA) models [3, 4, 5], are increasingly capable of completing challenging manipulation tasks. Yet, these policies remain brittle during deployment: a missed grasp, a dropped object, or a misaligned insertion might cause an otherwise successful rollout to fail. Interactive imitation learning methods, such as DAgger [6] and its human-gated variants [7, 8, 9, 10, 11], address this problem by collecting expert interventions from states where the policy struggles and using these corrections to iteratively improve the policy. However, continuously monitoring robot execution and deciding when to intervene places a substantial burden on the human expert, making such approaches difficult to scale to robot fleets.

Robot-gated interactive imitation learning instead seeks to automate this decision by requesting human assistance when the policy is likely to fail, thereby making more efficient use of costly human supervision. Existing robot-gating methods, however, commonly rely on either learned failure-prediction

functions trained from explicit failure data [12] or policy-dependent uncertainty scores that require task- and policy-specific calibration [13, 14, 15, 16, 17]. The former requires collecting failures that are undesirable or infeasible during safety-critical deployment, while the latter couples the gating mechanism to a particular policy checkpoint. As the policy changes throughout interactive learning, the gate needs to be retrained or recalibrated, limiting its scalability.

Ideally, a scalable gating mechanism for interactive imitation learning should satisfy four properties: (1) detect failures early while avoiding unnecessary interruptions of successful behavior; (2) remain effective across iterations of interactive learning without repeated retraining or recalibration; (3) operate with different, potentially black-box, policy architectures; and (4) transfer across tasks without requiring a separate training or calibration procedure for each task.

Our key insight is that general-purpose progress-based reward models, which estimate task progress directly from visual observations and natural-language instructions [18, 19, 20], provide a natural signal for such a gate. Unlike policy-dependent uncertainty measures, task progress describes what the robot has accomplished rather than how the policy generates its actions. Consequently, progress can be evaluated independently of the underlying policy architecture or checkpoint. This decoupling allows the gating mechanism to be fixed as the policy is updated and to be applied to different policies without access to their internal representations. Moreover, by leveraging reward models pretrained across diverse tasks, progress provides a common signal that can support gating across tasks without task-specific gate training. Importantly, even under task distribution shift, the reward model only determines when to request assistance; the policy is still trained from human corrections, which provide valid task-specific supervision.

Based on this insight, we introduce **Reward-Dagger**, a robot-gated interactive imitation learning framework that uses a simple but effective two-timescale rule over progress estimates from a general-purpose progress-based reward model to trigger human takeover. A short-window gate detects rapid decreases in progress, capturing immediate failures such as dropping an object, while a long-window gate detects progress plateaus corresponding to stalled behavior, such as repeatedly failed grasps. When either gate fires, control is transferred to a human expert, and the resulting corrections are used in the next policy update. Importantly,

^{*}Equal Contribution

both the reward model and the gating configuration remain fixed throughout interactive learning, requiring no per-round retraining or recalibration.

We evaluate Reward-DAGger along three dimensions: failure-detection performance, cross-task transfer of the gating parameters, and downstream policy improvement. On four simulation tasks, Reward-DAGger achieves a strong accuracy–latency trade-off compared to existing robot-gating and runtime-monitoring baselines. We further find that a single gate configuration selected jointly on multiple source tasks transfers to held-out target tasks without reducing detection accuracy and with only a minor increase in detection latency. In simulation, Reward-DAGger consistently improves autonomous success rate across three rounds of interactive learning while achieving the highest return on human effort. Finally, we deploy the same simulation-selected gate configuration on four real-world manipulation tasks with large pretrained VLA policies. Without any task-specific hyperparameter tuning, Reward-DAGger achieves competitive performance across all tasks and outperforms the baselines in most settings in terms of autonomous success rate and return on human effort, demonstrating that the gating mechanism transfers across environments and policy architectures.

In summary, our main contributions are:

- We propose a two-timescale, progress-based gating mechanism that detects both abrupt failures and stalled behavior during robot execution. The gate is independent of the underlying policy, remains fixed across policy updates, and supports cross-task configuration transfer.
- We introduce Reward-DAGger, an interactive imitation learning framework that uses this gating mechanism to trigger human takeover and iteratively finetune the policy from the resulting expert corrections.
- We extensively evaluate Reward-DAGger across four simulated and four real-world manipulation tasks with both task-specific diffusion policies and large pretrained VLAs, studying failure detection, cross-task transfer, downstream policy improvement, and human-effort efficiency.

II. RELATED WORK

Interactive Imitation Learning. Interactive imitation learning mitigates distribution shift in behavioral cloning by iteratively collecting expert actions on states visited by the learned policy [6, 7, 8, 9, 12, 21]. In *human-gated* methods, a supervisor monitors execution and decides when to intervene [7, 8, 9, 11, 22]. This monitoring requirement becomes burdensome when supervision scales to many robots [23, 24, 25]. *Robot-gated* methods instead automate this decision using signals such as action disagreement, estimated risk, or policy uncertainty [12, 14, 15, 17, 26, 27]. However, these gates commonly depend on task-specific learned criteria or policy-dependent signals that require additional training or calibration across tasks or policy updates. In contrast, Reward-DAGger uses an external task-progress signal that can be applied zero-shot across tasks and policy architectures.

Runtime Monitoring in Robot Manipulation. Runtime monitoring provides a closely related approach for identifying policy failures during deployment. These methods typically reduce observations, predicted actions, or internal policy representations to a scalar score, and then calibrate a conformal threshold for failure detection. Representative methods derive these scores from temporal inconsistencies among consecutive actions [28], deviations from the nominal observation distribution [16, 29], action chunk entropy [29], or learned failure predictors using internal VLA features [13]. However, these methods are primarily studied as deployment-time mechanisms without updating the monitored policy. Reward-DAGger closes this loop between monitoring and learning by using gate firings to collect expert corrections for subsequent policy updates without inducing expensive retraining or recalibration overheads.

General-Purpose Reward Models. Reward functions provide scalar feedback about the desirability of states or actions and are commonly used to guide policy optimization. Recently, significant effort has focused on developing general-purpose reward models that provide dense reward signals across diverse tasks [18, 20, 30, 31, 32, 33]. A common approach is to estimate task progress from visual observations and language instructions and use this progress signal as a proxy for per-timestep reward [18, 19, 31, 32], with recent works demonstrating its use for reinforcement learning in real-world robotic manipulation. Despite these advances, directly optimizing policies with imperfect learned reward functions remains challenging, as they may exploit inaccuracies or spurious correlations in the learned reward, leading to reward hacking without completing the intended task [34, 35, 36, 37]. Rather than directly optimizing the progress signal, our key insight is to use it as a runtime monitoring signal for interactive imitation learning. Since progress is evaluated independently of the underlying policy and represented on a shared scale across tasks, it provides a natural signal for determining when expert intervention is needed without additional task- or policy-specific training or calibration.

III. METHOD

Reward-DAGger leverages dense progress signals obtained from general-purpose reward models to determine when a robot policy should transfer control to the expert. In this section, we first introduce how we construct a gating criterion from general-purpose reward models, and then discuss how we incorporate the criterion into an interactive imitation learning pipeline for fine-tuning the base policy.

A. Gating Criterion

A general-purpose reward model is typically constructed by prompting or fine-tuning a VLM to produce dense progress signals. Formally, let ℓ denote a natural-language task instruction and let $o_{\leq t} = (o_1, \dots, o_t)$ denote the visual observations available up to time t . A general-purpose progress-based reward model f_ψ produces a scalar estimate

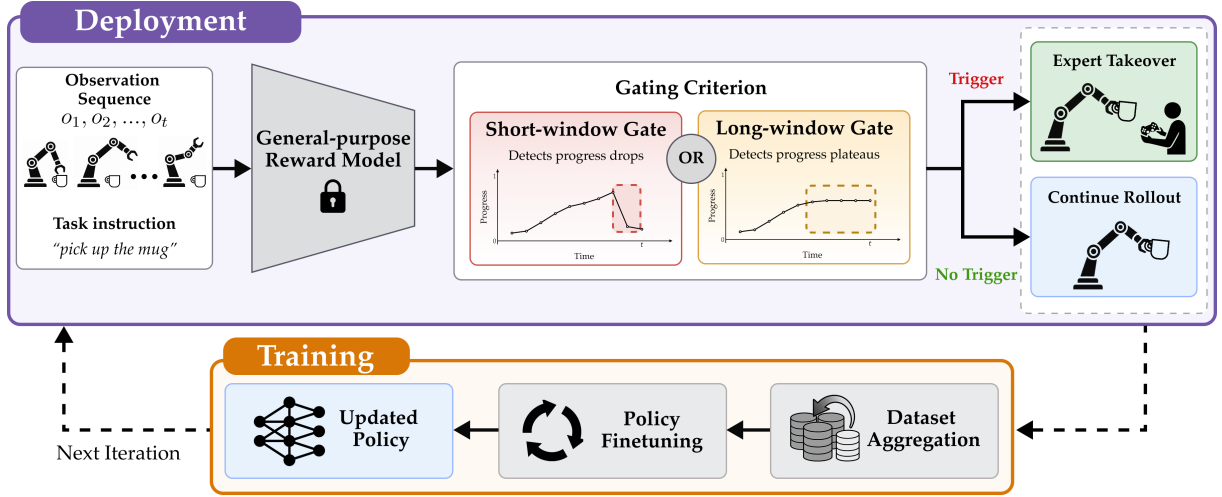


Fig. 1: **Reward-DAGger** is a robot-gated interactive imitation learning framework that uses a two-timescale rule over progress estimates from a general-purpose reward model to trigger human takeover. Collected corrections are appended to an initial dataset and the policy is finetuned in an iterative fashion.

of current task progress,

$$\phi_t = f_\psi(o_{\leq t}, \ell), \quad (1)$$

where larger values indicate greater predicted progress.

Inspired by [18], we target two common temporal signatures of policy failure in terms of the predicted progress signal. First, the policy may suddenly undo previously achieved progress. For example, a grasped object may slip from the gripper, or a partially inserted object may be pulled back out. These failures produce rapid decreases in predicted progress. Second, the policy may become stuck without advancing the task. For example, it may repeatedly miss a grasp or oscillate near the target pose without completing the placement. Such failures appear as plateaus in the predicted progress.

Following this observation, we design two complementary gates that constitute the overall *reward gate*. At time step t , the reward gate has access to the progress history $\phi_{1:t} = (\phi_1, \dots, \phi_t)$. The *short-window gate* examines the most recent L_s progress estimates, $\phi_{t-L_s+1:t}$, to detect sharp decreases in progress. The *long-window gate* examines the most recent L_l progress estimates, $\phi_{t-L_l+1:t}$, to detect progress plateaus, where $L_l > L_s$.

To quantify drops and plateaus, we use Spearman rank correlation: given a time-indexed window $\mathbf{v} = (v_1, \dots, v_n)$ of scalar values, we rank its values and compute the Pearson correlation between these ranks and the time indices $(1, \dots, n)$. We denote the resulting coefficient by $\rho(\mathbf{v})$. A coefficient near $+1$ indicates monotonically increasing progress, a coefficient near -1 indicates monotonically decreasing progress, and a coefficient near 0 indicates no clear monotonic trend. We compute the Spearman correlation within the short and long windows as

$$\rho_t^s = \rho(\phi_{t-L_s+1:t}), \quad \rho_t^l = \rho(\phi_{t-L_l+1:t}). \quad (2)$$

One problem with Spearman correlation is that it captures the direction of a progress trend but not the magnitude of the change. This is particularly important for the short-window gate, since a small fluctuation may produce a strongly

negative correlation. Therefore, we additionally define the maximum drop magnitude over the window as

$$\Delta_t = \max_{j \in \{t-L_s+1:t\}} \phi_j - \phi_t. \quad (3)$$

Using the window correlations and maximum drop magnitude, we define the complete gating criterion as

$$g_t^s = \mathbb{I}[(\rho_t^s < \tau_s) \wedge (\Delta_t \geq \delta_{\min})], \quad g_t^l = \mathbb{I}[\rho_t^l < \tau_l], \quad g_t = g_t^s \vee g_t^l, \quad (4)$$

where τ_s and τ_l are the respective correlation thresholds, while $\delta_{\min}$ sets the maximum drop magnitude threshold for the short-window gate. The reward gate fires when the short-window gate or long-window gate fires. A gate configuration is specified by $(L_s, \tau_s, \delta_{\min}, L_l, \tau_l)$. We set $\tau_s < \tau_l$ so that the short-window gate detects progress drops, while the long-window gate detects progress plateaus. Each gate is evaluated only after its corresponding window is full.

B. Reward-DAGger

We now summarize the Reward-DAGger procedure. We start with a robot policy π_0 and an offline dataset of expert demonstrations. During a rollout, the robot executes actions in the environment until the reward gate fires, after which the expert takes over and provides corrections. At each iteration i , we collect expert correction transitions until a fixed budget is reached. We then aggregate these expert interventions with the existing dataset and fine-tune π_i to obtain the next student policy π_{i+1} . This process is repeated for a fixed number of iterations without changing the reward gate's configuration. An overview of the framework is shown in Figure 1.

IV. EXPERIMENTS

Our experiments evaluate the effectiveness of Reward-DAGger as a robot-gated interactive imitation learning framework across different tasks and policy architectures. Specifically, we seek to answer the following research questions: **Q1.** Does the gating mechanism of Reward-DAGger accurately detect failures with low detection latency?

Q2. Does the gating configuration of Reward-Dagger transfer across tasks without additional calibration?

Q3. Does Reward-Dagger efficiently improve downstream policy performance while minimizing human burden?

Baselines. We compare **Reward-Dagger** against robot-gated Dagger methods as well as recent runtime monitoring approaches adapted to interactive imitation learning:

- **ThriftyDagger** [12] trains a policy ensemble to estimate state novelty, and a Q -function to estimate risk. The human takeover is triggered when a state is deemed novel or risky.
- **Diff-Dagger** [14] uses the diffusion policy’s denoising loss for the proposed action as an uncertainty score, which is then thresholded to trigger human takeover.
- **UCF** [15] samples candidate end-effector poses within a radius of the current pose, fits a Gaussian mixture model to the denoising vectors, and combines inter-mode divergence with intra-mode variance to estimate uncertainty.
- **SAFE-LSTM** [13] trains an LSTM on the final-layer features of a VLA policy to produce a scalar failure score, which is thresholded for detecting failures.
- **logpZO** [16] trains a flow matching model to map in-distribution observations to the standard normal distribution, and uses the distance between the current observation’s latent representation and the origin as the score.

Reward Model. We instantiate the reward model in Reward-Dagger using Robometer-4B [18], a general-purpose reward model that produces dense, frame-level estimates of task progress on a normalized $[0, 1]$ scale. We select Robometer because it outperforms prior general-purpose reward-model baselines across cross-domain robotic reward evaluations.

Simulation Setup. For our simulation experiments, we use LIBERO-10 [38], a suite of 10 long-horizon tasks with diverse settings. Simulation enables us to evaluate different methods across multiple tasks at scale, collecting enough rollouts to reliably characterize their accuracy–latency trade-offs. In these experiments, we use a frozen $\pi_{0.5}$ [39] LIBERO checkpoint as the expert policy. The student is a diffusion policy [1] trained on LIBERO demonstrations.

We first identify Tasks 0, 1, 4, 5, 6, and 8, where the initial student policies show intermediate performance, yielding sufficient numbers of both successful and failed trajectories for meaningful evaluation. For failure-detection and downstream interactive learning experiments, we use Tasks 0, 1, 4, and 6, as these are the tasks our expert policy consistently recovers from frequent failure states. For experiments that only require student-policy rollouts, including hyperparameter selection and cross-task transfer, we use all tasks.

Real Robot Setup. Our real-robot experiments use the DROID [40] setup, with a Franka FR3 robot arm equipped with a Robotiq 2F-85 gripper. We evaluate Reward-Dagger on four tasks: *stack* the red block on top of the blue block, *pick* up the red marker and put it into the blue cup, *put* the slice of bread from the toaster to the bowl, and *remove* the towel and screwdriver from the basket and put them on the table. In these experiments, the student is a pretrained $\pi_{0.5}$

policy, and the expert is a human teleoperator. An overview of the simulated and real-world tasks is provided in Figure 3.

Q1: Failure Detection

We first evaluate the effectiveness of Reward-Dagger’s gating mechanism on individual tasks based on two metrics:

- **Accuracy.** The gate should fire on failure trajectories but not on successful trajectories. Since the evaluation data may contain an unbalanced number of successful and failed trajectories, we measure balanced accuracy:

$$\text{BA}(c) = \frac{1}{2}(\text{TPR}(c) + \text{TNR}(c)) \quad (5)$$

where c denotes a gate configuration. TPR corresponds to the fraction of failed trajectories where the gate fires, and TNR corresponds to the fraction of successful trajectories where the gate does not fire.

- **Latency.** The gate should fire as early as possible when a failure occurs. Since it is difficult to manually label the precise onset of every failure, we use the *mean trigger time* on the failed trajectories as a surrogate for detection latency. For a failure trajectory where the gate does not fire, we assign the episode horizon as its trigger time. We normalize this value by the episode horizon.

We compare Reward-Dagger’s gate against two categories of baselines: (1) **Learned gates** train additional models to produce the gating signal; this category includes ThriftyDagger, SAFE-LSTM, and logpZO. (2) **Post-hoc gates** derive their scores directly from policies without additional offline training; Diff-Dagger and UCF fall into this category.

For each of Tasks 0, 1, 4, and 6 from LIBERO-10 suite, we collect 200 rollouts from the student policy. We reserve 50 rollouts for training the learned gates and evaluate every method on the remaining 150 rollouts. For ThriftyDagger, we use the policy ensemble and Q -risk estimator from its final checkpoint after three rounds of Dagger, which were trained on more transitions than the 50 reserved rollouts. We train SAFE-LSTM and logpZO using the 50 rollouts according to their respective training objectives. Reward-Dagger, Diff-Dagger, and UCF require no gate training.

Comparing methods at a single operating point is insufficient because they might be noisy and difficult to compare due to the tradeoff between accuracy and trigger time. We therefore sweep multiple gate configurations and construct a Pareto frontier for each method. Let T_{det} be the normalized trigger time averaged across all failure trajectories. A configuration c is Pareto-optimal if no other evaluated configuration c' achieves both higher accuracy $\text{BA}(c') \geq \text{BA}(c)$ and lower latency $T_{\text{det}}(c') \leq T_{\text{det}}(c)$, with at least one inequality strict.

To generate the candidate configurations, we sweep the score-quantile parameter α [41] for each baseline. For Reward-Dagger, we sweep the false-positive-rate cap β . For each β , we choose the gate configuration from a 3600-configuration pool with the lowest trigger time subject to a false-positive rate at most β . We evaluate each configuration using five-fold cross-validation repeated over five random partitions on the 150 evaluation rollouts. Gate parameters are selected on four folds and evaluated on the held-out fold.

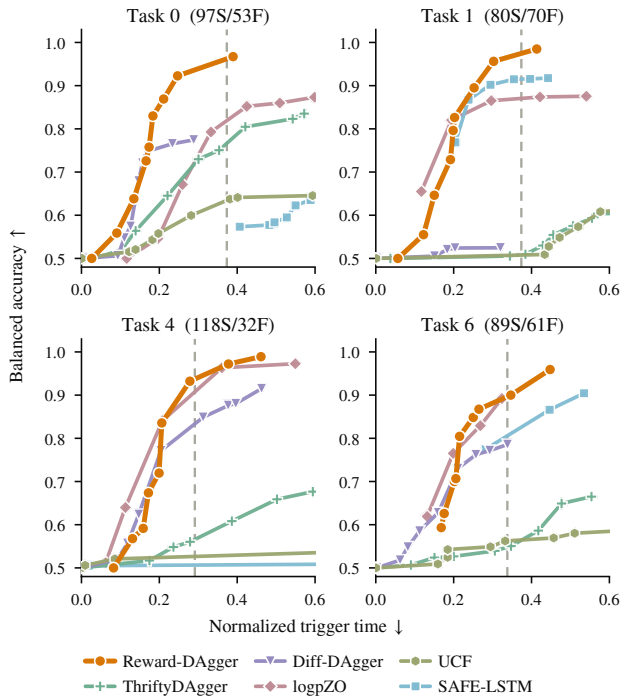


Fig. 2: **Detection frontier on LIBERO tasks.** Each curve shows the Pareto frontier between balanced accuracy and mean normalized trigger time. S and F denote the numbers of successful and failed trajectories; the dashed gray line indicates the mean normalized length of successful trajectories.

The results are shown in Figure 2. Reward-DAGger achieves the highest maximum balanced accuracy on every task, reaching at least 0.96. Its frontier also lies above or on par with the baselines, with large improvements on Tasks 0 and 1. logpZO remains competitive at some early-trigger operating points, but no baseline performs consistently as well as Reward-DAGger across all four tasks.

Q2: Transfer of Gating Configurations

Most existing baselines use *task-specific* gates, meaning that the same gating function or thresholds tuned on one task cannot be transferred directly to another. Several baselines are also *policy-specific*, meaning that updating a policy on the same task requires retraining or recalibrating the gate. In contrast, Reward-DAGger depends on generic progress signals, which enable better transfer across tasks and policies. We investigate its cross-task transfer performance on LIBERO-10 tasks 0, 1, 5, and 8.

We ask whether a single configuration tuned on one or more source tasks transfers to other tasks. We consider two settings: *one-to-three transfer*, where we tune the gate on one task and evaluate it on the other three tasks, and *three-to-one transfer*, where we tune the gate jointly on three tasks and evaluate it on the remaining task.

To select a single configuration from the source tasks, we define a balanced-accuracy tolerance ϵ . Given ϵ and a set of source tasks $\mathcal{S}$, let $\text{BA}_k^* = \max_c \text{BA}_k(c)$ denote the highest

TABLE I: **Configuration transfer across tolerance levels.** ΔBA is in percentage points; mean, median, and worst denote matched-accuracy latency penalties (%).

ϵ	One-to-three				Three-to-one			
	ΔBA	Mean	Median	Worst	ΔBA	Mean	Median	Worst
0.00	-1.6	21.0	16.7	53.2	—	—	—	—
0.01	-2.1	13.1	12.7	21.9	-0.7	21.7	18.9	33.7
0.02	-5.0	11.4	8.9	33.9	0.0	7.3	6.7	11.2
0.03	-7.3	11.6	8.0	42.9	0.0	7.6	7.8	13.7
0.05	-7.7	10.8	7.5	29.9	0.0	9.2	7.8	19.2

balanced accuracy obtained on task k . We select

$$c_\epsilon(\mathcal{S}) = \underset{c}{\operatorname{argmin}} \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} T_{\text{det},k}(c) \quad (6)$$

subject to $\text{BA}_k(c) \geq \text{BA}_k^* - \epsilon, \quad \forall k \in \mathcal{S}.$

The selected configuration c minimizes average trigger time while staying within ϵ of the highest balanced accuracy on every source task.

We quantify transfer using two metrics. To measure whether transfer preserves balanced accuracy, we use *change in balanced accuracy*, denoted by $\Delta\text{BA} = \text{BA}_{\text{transfer}} - \text{BA}_{\text{native}}$. Both quantities are measured on the target task, with $\text{BA}_{\text{native}}$ achieved by a configuration tuned on that task using the same ϵ . To measure whether transfer preserves latency, we use *matched-accuracy latency penalty*. It measures, at the same balanced accuracy, the percentage increase in trigger time relative to the configuration on the target task’s native frontier. A lower penalty indicates better transfer.

Table I reports the results across all tested balanced-accuracy tolerance values. We note that one-to-three task transfer is not consistently reliable; although its median latency penalties are moderate, no tolerance level preserves balanced accuracy and worst-case latency. We attribute this to possible overfitting of task-specific sampling noise when selecting among many configurations using rollouts from only one task. In contrast, selecting the gate jointly across multiple source tasks substantially improves transfer stability, preserving average balanced accuracy over a broad range of tolerances while incurring relatively small latency penalties.

Given the reliability of three-to-one transfer, we fix $\epsilon = 0.02$ and examine these configurations in greater detail. In Table II, we report the transfer metrics of a configuration tuned on three of Tasks 0, 1, 5, and 8 and evaluated on both the omitted task and independently held-out Tasks 4 and 6. The final row reports a configuration tuned jointly on all four source tasks and evaluated on Tasks 4 and 6.

Together, these results demonstrate that, by calibrating across a broad set of source tasks, reward gate configurations transfer reliably to target tasks, preserving balanced accuracy and inducing a small, controllable latency penalty.

Q3: Policy Improvement with Reward-DAGger

We next evaluate whether the interventions triggered by Reward-DAGger lead to efficient downstream policy improvement. We consider four simulated and four real-world tasks, visualized in Figure 3. Across all eight tasks, we

TABLE II: **Transfer results** at $\epsilon = 0.02$ for three-to-one transfer and held-out Tasks 4 and 6. T_{det} is mean normalized trigger time; Pen is matched-accuracy latency penalty.

Held-out task	Three-to-one			Eval on Task 4			Eval on Task 6		
	ΔBA	T_{det}	Pen	ΔBA	T_{det}	Pen	ΔBA	T_{det}	Pen
Task 0	-1.2	0.354	9.0	-4.4	0.286	0.8	-1.7	0.384	8.7
Task 1	-0.9	0.341	4.3	-11.0	0.260	11.7	-0.8	0.379	3.6
Task 5	1.4	0.336	11.2	-1.3	0.352	1.9	0.0	0.400	6.7
Task 8	0.5	0.485	4.5	-1.3	0.352	1.9	0.0	0.400	6.7
None (all four)	-	-	-	-1.3	0.352	1.9	0.4	0.412	3.2

use the same reward-gate configuration selected jointly on LIBERO Tasks 0, 1, 5, and 8:

$$(L_s, \tau_s, \delta_{\min}, L_l, \tau_l) = (75, -0.9, 0.05, 225, 0.0).$$

In simulation, we query the reward model at every environment step. For real-world experiments, we address querying latency by employing asynchronous scoring and querying the reward model every $k = 4$ environment steps.

For each method, we perform policy finetuning over multiple rounds. In each round, we collect a fixed budget of expert transitions using the corresponding gating method and then finetune the base policy on a replay buffer containing the original offline demonstrations and the aggregated expert corrections. Following [14], once takeover is triggered, the expert controls the robot for the remainder of the episode. After each round of training, we evaluate two metrics:

- **Autonomous success rate** evaluates the updated student policy over K autonomous rollouts. It reflects how effectively the gate triggers informative expert interventions.
- **Return on human effort (ROHE)** evaluates the performance of the combined student-expert system relative to the amount of expert interventions required. Following [26], ROHE is defined as

$$\text{ROHE} = \frac{\mathbb{P}(S_\tau = 1)}{1 + H/T}. \quad (7)$$

The numerator corresponds to the task’s success rate when the gate is active and the expert can take over. H/T denotes the fraction of rollout episodes in which the expert intervenes. A high ROHE indicates that the combined system achieves a high success rate while requiring expert intervention in relatively few episodes. We measure ROHE at the episode level as each intervention in our setting results in expert takeover for the remainder of the episode.

Simulation Results. We evaluate Reward-DAGger on LIBERO-10 tasks 0, 1, 4, and 6. Among the downstream tasks, Tasks 0 and 1 were included in gate-configuration tuning, whereas Tasks 4 and 6 were held out.

We perform three rounds of interactive learning and collect 3,000 successful expert state-action pairs for each round. We compute each round’s ROHE from the expert-gated rollouts during data collection. We then finetune the current student policy using a sampling mixture of 30% offline demonstrations and 70% aggregated expert interventions. After each round, we evaluate the updated policy’s autonomous success rate using 100 rollouts. Since the final policy is not used in



Fig. 3: **Overview of simulated and real-world tasks.** Left and right images show each task’s initial and final frames.

TABLE III: **Real-world results.**

Task	Method	Success rate (%)			ROHE		
		R0	R1	R2	R0	R1	R2
Stack	Reward-DAGger	65	75	80	.667	.731	.643
	ThriftyDAGger	65	70	55	.606	.556	.513
	Diff-DAGger	65	75	70	.588	.783	.600
Pick	Reward-DAGger	20	30	40	.541	.533	.588
	ThriftyDAGger	20	25	25	.526	.474	.424
	Diff-DAGger	20	35	40	.548	.435	.478
Put	Reward-DAGger	35	40	45	.594	.606	.645
	ThriftyDAGger	35	35	50	.559	.500	.526
	Diff-DAGger	35	20	30	.500	.346	.528
Remove	Reward-DAGger	75	85	90	.588	.714	.769
	ThriftyDAGger	75	70	80	.526	.486	.563
	Diff-DAGger	75	70	80	.545	.613	.773

a subsequent collection round, we do not report ROHE for the final checkpoint.

Because UCF, SAFE, and logpZO were not originally designed for iterative policy updates, we collect 50 additional rollouts from the updated student policy after each round. For UCF, these rollouts are used to recalibrate its uncertainty threshold; for SAFE and logpZO, they are used to retrain their scoring models and recalibrate their thresholds.

The results, averaged across three seeds, are reported in Figure 4. Reward-DAGger improves autonomous success on all four tasks, achieving the highest final success on Tasks 0 and 6 and remaining competitive on Tasks 1 and 4. It also achieves the highest or near-highest ROHE throughout data collection, indicating that these improvements are obtained with efficient use of expert interventions. Importantly, these gains extend to Tasks 4 and 6 despite neither task being used to select the gate configuration.

Real-robot Tasks. We then evaluate Reward-DAGger on

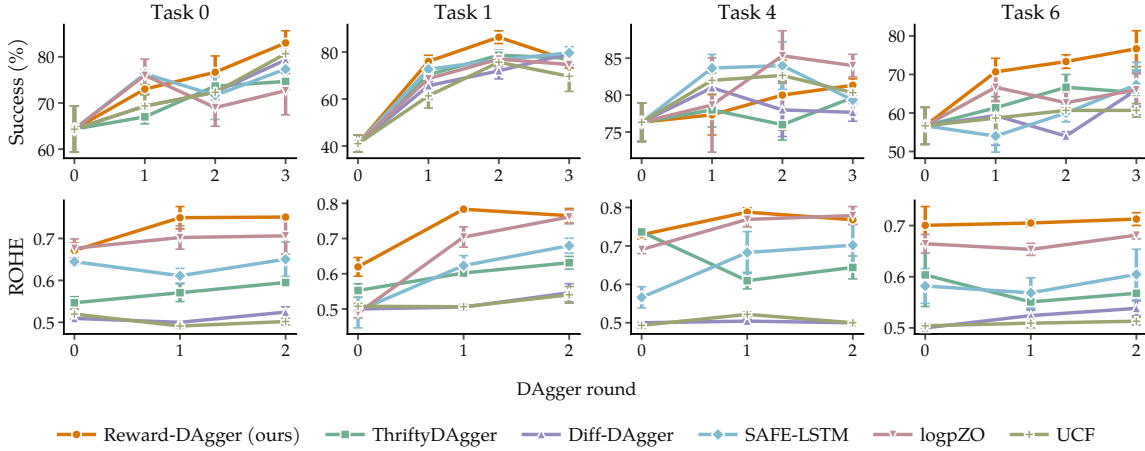


Fig. 4: **Policy improvement results** on LIBERO-10 Tasks 0, 1, 4, and 6. Error bars indicate standard error across 3 seeds.

four real-robot tasks. For *stack*, *pick*, and *put*, there exist exact or closely related examples in the dataset used to train Robometer-4B. In contrast, the *remove* task involves manipulating two distinct objects in sequence and serves as an out-of-distribution (OOD) task.

We use $\pi_{0.5}$ -DROID as the student policy and provide real human interventions through teleoperation. We deploy the same reward gate configuration selected in simulation without additional tuning on the real-world tasks. We compare Reward-DAGger with ThriftyDAGger and Diff-DAGger and evaluate every iteration’s policy checkpoint with 20 rollouts. We exclude UCF, SAFE, and logpZO from the real-world comparisons because these methods require obtaining fresh rollouts of the latest policy for recalibration or retraining. Under our 20-rollout evaluation budget, we find that calibration is highly unstable. To compute ROHE, we use the evaluation rollouts, where the human annotator judges whether the state at which the gate triggers is recoverable. A rollout is considered successful for the combined student-expert system if the student completes the task autonomously or the gate triggers while expert recovery remains feasible.

Results are reported in Table III. Reward-DAGger achieves strong overall performance, obtaining the highest ROHE in most settings and comparable performance in the others, while consistently improving autonomous success across DAGger rounds. This advantage extends to the *remove* task, demonstrating Reward DAGger’s effectiveness on OOD tasks. Empirically, ThriftyDAGger and Diff-DAGger frequently trigger during successful rollouts, particularly in early rounds, leading to unnecessary interventions; in contrast, Reward-DAGger rarely triggers on successful behavior.

Overall, our results show that *Reward-DAGger collects interventions that are both informative for downstream policy improvement and efficient in their use of human supervision*. Notably, the same reward model and gate configuration is used without additional tuning across eight simulated and real-world tasks, and different policy architectures.

Ablations

We conduct two ablations to evaluate our short- and long-window gate design and whether Reward-DAGger remains

TABLE IV: **Gate design ablation:** earliest normalized trigger time for each balanced-accuracy threshold.

Task	Detector	Trigger time at balanced accuracy $\geq$			
		0.80	0.85	0.90	0.95
Task 0	Full gate	.184	.211	.224	.389
	Short only	.194	.228	.255	–
	Long only	.219	.219	.234	.411
	Absolute thresh.	.205	.266	.331	.500
Task 1	Full gate	.199	.252	.261	.303
	Short only	.233	.241	.299	.324
	Long only	.243	.260	.279	.319
	Absolute thresh.	.270	.270	.323	.385
Task 4	Full gate	.206	.212	.279	.378
	Short only	.342	.342	–	–
	Long only	.242	.312	.337	.373
	Absolute thresh.	.313	.313	.313	.357
Task 6	Full gate	.215	.265	.347	.448
	Short only	.215	.249	.358	–
	Long only	.319	.319	.329	.551
	Absolute thresh.	.287	.314	.355	.445

effective across different reward models.

Gate Design. We compare the complete reward gate against three baselines: a short-window-only gate, a long-window-only gate, and an absolute-progress gate, which triggers when progress drops below a certain amount.

Table IV summarizes the resulting accuracy-latency trade-off. The full gate achieves the lowest or tied trigger time in 11 of the 16 task-accuracy combinations, highlighting the importance of combining both gates. The short-window-only gate detects some failures earlier but fails to reach high balanced accuracy on Tasks 0, 4, and 6, whereas the long-window-only gate provides broader coverage at the cost of later detection. The full gate also triggers earlier than the absolute-progress gate in 14 of the 16 comparisons.

Different Reward Model. To test whether our gating mechanism generalizes across progress-based reward models, we replace Robometer-4B with RynnValue-4B while keeping the gate construction and evaluation protocol unchanged. Fig. 5 shows that, under the same configuration-selection procedure on two simulation tasks, RynnValue achieves a comparable accuracy–latency frontier and the second-highest balanced

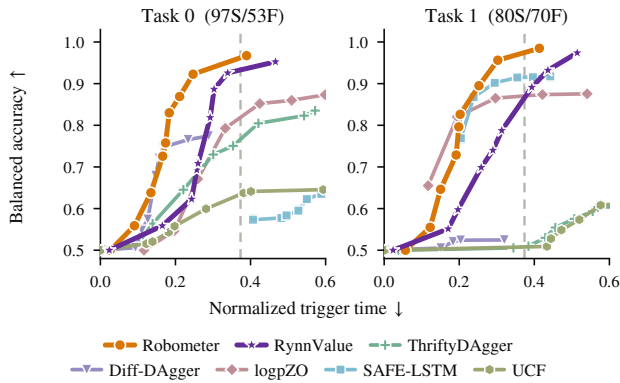


Fig. 5: **Reward model ablation.** Detection frontiers using Robometer and RynnValue as reward model backbones.

accuracy on both tasks, suggesting that the gating mechanism is not specific to a particular reward-model backbone.

V. CONCLUSION

Summary. We introduced Reward-DAGger, a robot-gated interactive imitation learning framework that detects progress drops and plateaus using estimates from a general-purpose reward model. By decoupling the gating mechanism from the underlying policy, Reward-DAGger operates across policy checkpoints and architectures, including black-box policies, without retraining or recalibrating the gate between DAGger rounds. In simulated and real-robot experiments, Reward-DAGger achieves effective failure detection, cross-task transfer, and efficient downstream policy improvement. Reward-DAGger is modular with respect to the reward model: more accurate general-purpose reward models can be incorporated without modifying the interactive imitation learning pipeline. **Limitations and Future Work.** Reward-DAGger is not without limitations. Its long-window gate must observe a sustained progress plateau, so failures such as repeated unsuccessful grasp attempts may persist for several seconds before triggering an intervention. More broadly, the framework is reactive rather than preemptive, requesting human assistance only after evidence of a failure emerges, which limits its applicability in high-risk settings or when failures are difficult to recover from. A promising direction is to integrate predictive world models [42, 43] into the gating framework, allowing the robot to anticipate possible failure outcomes and request intervention before executing them.

ACKNOWLEDGMENTS

The authors thank the USC Center for Advanced Research Computing (CARC) for providing us with compute resources, and acknowledge funding by the Airbus Institute for Engineering Research (AIER). This work was supported by the Office of Naval Research (ONR) Young Investigator Program (YIP) under grant number N00014-26-1-2473.

REFERENCES

- [1] C. Chi et al., “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion,” in *Robotics: Science and Systems (RSS)*, 2023.
- [2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, “Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware,” in *Robotics: Science and Systems (RSS)*, 2023.

- [3] M. J. Kim et al., “OpenVLA: An open-source vision-language-action model,” in *Conference on Robot Learning (CoRL)*, 2024.
- [4] K. Black et al., “ π_0 : A vision-language-action flow model for general robot control,” *arXiv preprint arXiv:2410.24164*, 2024.
- [5] O. M. Team et al., “Octo: An open-source generalist robot policy,” in *Robotics: Science and Systems (RSS)*, 2024.
- [6] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *International Conference on Artificial Intelligence and Statistics*, 2011.
- [7] H. Liu, S. Nasiriany, L. Zhang, Z. Bao, and Y. Zhu, “Robot learning on the job: Human-in-the-loop autonomy and learning during deployment,” in *Robotics: Science and Systems (RSS)*, 2023.
- [8] Y. Korkmaz and E. Bıyık, “Mile: Model-based intervention learning,” in *International Conference on Robotics and Automation (ICRA)*, 2025.
- [9] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, “Hg-dagger: Interactive imitation learning with human experts,” in *International Conference on Robotics and Automation (ICRA)*, 2019.
- [10] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese, “Human-in-the-loop imitation learning using remote teleoperation,” *arXiv preprint arXiv:2012.06733*, 2020.
- [11] Z. Peng, Z. Liu, and B. Zhou, “Data-efficient learning from human interventions for mobile robots,” in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2025, pp. 7407–7414.
- [12] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg, “Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning,” in *Conference on Robot Learning (CoRL)*, 2021.
- [13] Q. Gu et al., “Safe: Multitask failure detection for vision-language-action models,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [14] S.-W. Lee, X. Kang, and Y.-L. Kuo, “Diff-dagger: Uncertainty estimation with diffusion policy for robotic manipulation,” in *International Conference on Robotics and Automation (ICRA)*, 2025.
- [15] Z. He, Y. Cao, and M. Ciocarlie, “Uncertainty comes free: Learning human-in-the-loop policies with diffusion models,” in *International Conference on Robotics and Automation (ICRA)*, 2026.
- [16] C. Xu et al., “Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies,” in *Robotics: Science and Systems (RSS)*, 2025.
- [17] V. Myers, E. Bıyık, and D. Sadigh, “Active reward learning from online preferences,” in *International Conference on Robotics and Automation (ICRA)*, 2023.
- [18] A. Liang et al., “Robometer: Scaling general-purpose robotic reward models via trajectory comparisons,” in *Robotics: Science and Systems (RSS)*, 2026.
- [19] H. Tan et al., “Robo-dopamine: General process reward modeling for high-precision robotic manipulation,” in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2026.
- [20] T. Lee, A. Wagenmaker, K. Pertsch, P. Liang, S. Levine, and C. Finn, “Roboreward: General-purpose vision-language reward models for robotics,” *arXiv preprint arXiv:2601.00675*, 2026.
- [21] K. Baraka et al., “Human-interactive robot learning: Definition, challenges, and recommendations,” *ACM Transactions on Human-Robot Interaction*, 2026.
- [22] Z. Hu et al., “Rac: Robot learning for long-horizon tasks by scaling recovery and correction,” *arXiv preprint arXiv:2509.07953*, 2025.
- [23] H. Liu et al., “Multi-task interactive robot fleet learning with visual world models,” in *Conference on Robot Learning (CoRL)*, 2024.
- [24] R. Hoque et al., “Fleet-DAGger: Interactive robot fleet learning with scalable human supervision,” in *Conference on Robot Learning (CoRL)*, 2022.
- [25] W. Yu, J. Lv, Z. Ying, Y. Jin, C. Wen, and C. Lu, “Armada: Autonomous online failure detection and human shared control empower scalable real-world deployment and adaptation,” *IEEE Robotics and Automation Letters*, 2026.
- [26] H. Liu, S. Dass, R. Martín-Martín, and Y. Zhu, “Model-based runtime monitoring with interactive imitation learning,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- [27] H. Cai, Z. Peng, and B. Zhou, “Robot-gated interactive imitation learning with adaptive intervention mechanism,” in *International Conference on Machine Learning (ICML)*, 2025.

- [28] C. Agia et al., “Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress,” in *Conference on Robot Learning (CoRL)*, 2024.
- [29] R. Römer, A. Kobras, L. Worbis, and A. P. Schoellig, “Failure prediction at runtime for generative robot policies,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [30] S. Chen et al., “Topreward: Token probabilities as hidden zero-shot rewards for robotics,” *arXiv preprint arXiv:2602.19313*, 2026.
- [31] D. Huang et al., “Rynvalue: Scaling robotic value foundation models with temporal distance,” *arXiv preprint arXiv:2608.09853*, 2026.
- [32] J. Zhang et al., “Rewind: Language-guided rewards teach robot policies without new demonstrations,” in *Conference on Robot Learning (CoRL)*, 2025.
- [33] S. A. Sontakke et al., “Roboclip: One demonstration is enough to learn robot policies,” in *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [34] R. Tian, Y. Wu, and A. Bajcsy, “Position: Good embodied reward models need bad behavior data,” in *International Conference on Machine Learning (ICML)*, 2026.
- [35] P. Schroeder, T. Weng, K. Schmeckpeper, E. Rosen, S. Hart, and O. Biza, “Sole-rl: Video-language reasoning as the sole reward for on-robot reinforcement learning,” *arXiv preprint arXiv:2603.28730*, 2026.
- [36] J. Zhang et al., “Progress reward modeling for robotic learning: A comprehensive survey,” *arXiv preprint arXiv:2607.21655*, 2026.
- [37] J. Tien, J. Z.-Y. He, Z. Erickson, A. Dragan, and D. S. Brown, “Causal confusion and reward misidentification in preference-based reward learning,” in *International Conference on Learning Representations (ICLR)*, 2023.
- [38] B. Liu et al., “Libero: Benchmarking knowledge transfer for lifelong robot learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [39] K. Black et al., “ $\pi_{0.5}$: A vision-language-action model with open-world generalization,” in *Conference on Robot Learning (CoRL)*, 2025.
- [40] A. Khazatsky et al., “Droid: A large-scale in-the-wild robot manipulation dataset,” in *Robotics: Science and Systems (RSS)*, 2024.
- [41] R. Koenker and G. Bassett Jr, “Regression quantiles,” *Econometrica: journal of the Econometric Society*, 1978.
- [42] T. Yin et al., “Playworld: Learning robot world models from autonomous play,” in *Conference on Robot Learning (CoRL)*, 2026.
- [43] Y. Wang et al., “Interactive world simulator for robot policy training and evaluation,” in *Robotics: Science and Systems (RSS)*, 2026.

Additional Experiments

Frontier Transfer. We explore whether configurations that perform well on one task remain effective on other tasks. For each source task, we construct its task-specific Pareto frontier using 200 policy rollouts from a diffusion policy trained exclusively on that task. We then apply the configurations on this frontier to each of the other three tasks and retain the Pareto-optimal points to form a transferred frontier.

To measure how closely each transferred frontier matches the target task’s native frontier, we compute the hypervolume ratio (HVR). Specifically, we divide the area under the transferred frontier by the area under the native frontier. The trigger-time range extends from zero to the latest non-dominated point reached by either frontier. An HVR near one indicates that the transferred configurations retain most of the accuracy–latency tradeoff obtained by task-specific tuning.

The results for each source-target task pair are shown in Figure 6. Across the 12 ordered source-target pairs, Reward-Dagger achieves a mean HVR of 0.92 and a median HVR of 0.93, indicating that *high-performing regions of the gate configuration space are largely shared across tasks*.

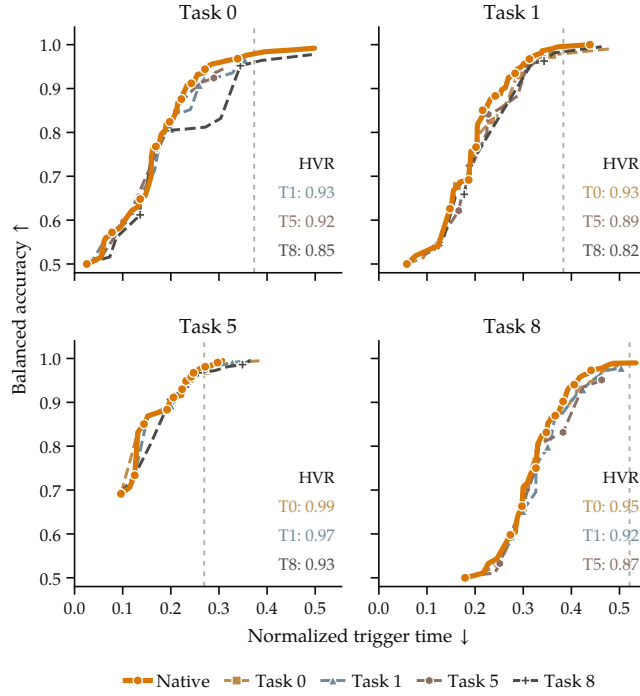


Fig. 6: **Frontier transfer of Reward-Dagger gate configurations.** Each panel corresponds to a target task. The solid orange curve denotes the native frontier obtained through task-specific tuning, and each dashed curve is obtained by applying configurations from another task’s frontier without retuning. Numbers indicate the hypervolume ratios (HVR) for each transferred curve. The dashed grey line indicates the mean normalized length of successful trajectories.

Different Reward Model Backbone in Real World Tasks. We further replace Robometer with RynnValue on the *Stack* and *Pick* policy-improvement experiments while directly reusing the Robometer-selected gating parameters, without additional tuning. As shown in Table V, RynnValue yields smaller gains in autonomous success but comparable ROHE overall, indicating that the proposed gating mechanism is not specific to a particular reward-model backbone.

TABLE V: **Different reward model backbone ablation on real-world tasks.**

Task	Method	Success rate (%)			ROHE		
		R0	R1	R2	R0	R1	R2
Stack	Robometer-4B	65	75	80	.667	.731	.643
	RynnValue-4B	65	65	70	.606	.645	.613
Pick	Robometer-4B	20	30	40	.541	.533	.588
	RynnValue-4B	20	25	25	.515	.556	.571

TABLE VI: Hyperparameter grids for detection frontier analysis

Method	Role	Parameter	Values	Grid size
	Frontier	FPR cap β	$\mathcal{B}$	22
Reward-DAGger	Inner search	Short-window length L_s	$\{20, 30, 50, 75, 100\}$	3,600
		Short-window threshold τ_s	$\{-0.9, -0.7, -0.5\}$	
		Minimum drop $\delta_{\min}$	$\{0, 0.05, 0.10, 0.20\}$	
		Long-window length L_l	$\{150, 175, \dots, 400, 450\}$	
		Long-window threshold τ_l	$\{-0.2, 0, 0.1, 0.3, 0.5\}$	
ThriftyDAGger	Frontier	Novelty/Q-risk quantile α	$\mathcal{A}$	43
Diff-DAGger	Frontier	Diffusion-loss quantile α	$\mathcal{A}$	43
UCF	Frontier	Uncertainty quantile α	$\mathcal{A}$	43
SAFE-LSTM	Frontier	Score quantile α	$\mathcal{A}$	43
logpZO	Frontier	Score quantile α	$\mathcal{A}$	43

$\mathcal{B} = \{0, 0.02\} \cup \{0.05, 0.10, \dots, 1.00\}$.
 $\mathcal{A} = \text{geomspace}(10^{-4}, 0.5, 41) \cup \{10^{-3}, 10^{-2}\}$.

TABLE VII: Training and evaluation details

	Simulated tasks				Real-robot tasks			
	Task 0	Task 1	Task 4	Task 6	Stack	Pick	Put	Remove
Successful offline transitions	9,293	12,702	10,353	9,245	4,265	3,536	3,436	7,454
Observations	Agent-view RGB, wrist RGB, and proprioception				Exterior RGB, wrist RGB, joint positions (7), and gripper state (1)			
Image crop size								
	224 × 224 (resize with padding)							
Action chunk (predicted/executed)	20/10				15/8			
Control frequency	20 Hz				15 Hz			
Episode horizon	800 steps				900 steps			
Student policy	Diffusion policy trained on offline expert demonstrations 3 rounds, 3000 transitions per round Full FT, 4k steps, batch size 256, constant lr 1e-4				$\pi_{0.5}$ -DROID			
Data Aggregation					2 rounds, 1500 transitions per round			
Training per round					LoRA, 3k steps, batch size 32, cosine lr 2.5×10^{-5}			
Expert	$\pi_{0.5}$ -LIBERO				Human teleoperator			
Evaluation rollouts per round	100				20			