SIM2CRITIC: LEARNING REWARD CRITICS FROM GAUSSIAN SPLATTING SIMULATION FOR REAL-WORLD ROBOT MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Simulation offers a key advantage for robot learning: privileged access to the underlying state enables accurate supervision of task success and progress. On real robots, however, this state is hidden behind visual observations, leaving post-training of generalist policies dependent on sparse, human-labeled outcomes. We introduce *Sim2Critic*, a vision-based critic trained with privileged supervision in a 3D Gaussian Splatting (3DGS) digital twin of the deployment scene and transferred to real-world observations. We study two variants that differ only in their supervision: Sim2Critic (Sparse) learns from sparse success signals, whereas Sim2Critic (Dense) learns from dense progress signals derived from the same simulator state. Across eight real-world manipulation tasks on two robot platforms, both variants transfer to real rollouts, requiring as few as ten real demonstrations per task. It outperforms existing generalist reward models in reward alignment and outcome detection, while using substantially fewer parameters and achieving lower inference latency. Beyond evaluation, the transferred critic supports policy improvement without per-episode reward annotation. It defines returns for offline reinforcement learning on a policy's own real-world rollouts and provides both rewards and success detection for online reinforcement learning on the robot. Through these two applications, we enhance the success rate of deployed policies by up to 2.2 times. Together, these results show that Sim2Critic turns a scanned workspace into a practical source of per-step reward for real-world robot learning.

1 INTRODUCTION

Generalist vision-language-action policies can execute a growing range of manipulation tasks, but deployment exposes a harder problem: how should the robot improve from its own experience? Recent methods post-train policies on real rollouts (Physical Intelligence et al., 2026; Xu et al., 2026; Fei et al., 2026; Xiao et al., 2026; Yang et al., 2026), moving toward systems that drive their own improvement (Pachocki, 2026). Such a loop needs an evaluator that answers two questions from visual observations: *Is the task progressing?* and *Did the rollout succeed?* These signals support not only policy learning, but also automatic evaluation, data selection, and failure diagnosis. Yet the state needed to compute them—object poses, contacts, containment, and deformable configurations—is hidden on the real robot.

Current approaches address this need through either human supervision or generalist reward models, but neither provides the desired combination of scene specificity and dense feedback. Human supervision provides reliable episode outcomes (Physical Intelligence et al., 2026; Xu et al., 2026) and can capture nuanced notions of progress, but dense frame-level annotation requires substantial labor, varies across annotators, and scales poorly to long-horizon tasks with multiple valid strategies (Zhang et al., 2026a). Generalist reward models instead learn broad semantic priors from cross-embodiment data (Zhang et al., 2026c; Lee et al., 2026; Liang et al., 2026; Schroeder et al., 2026). While these models eliminate the need for per-rollout annotation, their scores lack calibration to specific workspaces, may reveal perceptual shortcuts during optimization (Schroeder et al., 2026), and can be costly to evaluate in detail (Zhang et al., 2026a). Training a scene-specific evaluator from real data alone does not resolve the problem: failures are costly to collect, and dense progress remains

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.

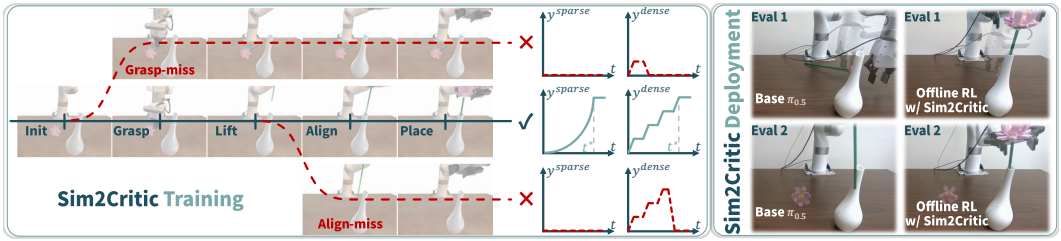


Figure 1: **Sim2Critic is a twin-grounded vision critic for real-robot evaluation and policy improvement.** *(Left)* In a physics-enabled 3DGS twin, expert rollouts and controlled failures are labeled with sparse success or dense progress targets. *(Right)* Transferred to real observations, the critic scores rollouts and supplies returns for reinforcement learning without per-episode reward annotation, improving flower placement over the base policy in both examples.

difficult to label consistently. What is missing is a signal that is scene-calibrated, dense, inexpensive per frame, and available without repeatedly annotating real rollouts.

Simulation appears to provide exactly this supervision. Privileged state makes success and progress measurable at every step, and scripted behavior can generate both successful and failed trajectories at scale. The difficulty is visual transfer: a critic trained on conventional rasterized simulation must cross a large appearance gap to the deployment cameras. Our ablations confirm that replacing a reconstruction-based renderer with synthetic meshes substantially degrades real-world reward prediction. Thus, simulation has the right labels but usually presents them through the wrong pixels.

We address the issue by learning a compact critic model in a photorealistic digital twin of the deployment scene. Specifically, we introduce Sim2Critic, which turns a scanned deployment workspace into a source of supervision for a real-world vision critic (Figure 1). From a short phone capture, we build a physics-enabled 3D Gaussian Splatting (3DGS) twin (Kerbl et al., 2023), align it with the robot and deployment cameras, and generate rollouts using scripted experts and controlled failure interventions. Privileged simulator state labels every rendered frame. Sim2Critic (Sparse) learns a discounted success return from the twin's success predicate, while Sim2Critic (Dense) learns task progress from an ordered ladder of state-based milestones. The two variants otherwise share the same data and architecture. Training also uses as few as ten successful real demonstrations per task, but no manually labeled real failures; at deployment, the critic receives only RGB observations and proprioception. In this sense, the twin acts as a *critic factory*: simulation supplies the supervision, while the scene reconstruction makes that supervision visually transferable.

We evaluate Sim2Critic on eight real-world tasks spanning rigid and deformable manipulation across an xArm 7 and an RM65-B bimanual robot (Figure 2). Both variants outperform zero-shot and few-shot generalist reward models in reward alignment and outcome discrimination, while using over $50\times$ fewer parameters than the smallest generalist baseline and achieving approximately $6.4\times$ lower per-value latency than the fastest. Beyond evaluation, the same fixed critic improves policy learning: when used to define returns for offline reinforcement learning on a policy's own real rollouts, it raises the eight-task average success rate of $\pi_{0.5}$ (Black et al., 2025a) from 32.0% to 49.0%. We further use the critic as both the reward and success detector for online reinforcement learning with DSRL (Wagenmaker et al., 2025), achieving the improvements in success rates by $1.7\times$ and $2.2\times$ on two tasks. In brief, our contributions are as follows:

- **Twin-grounded critic learning.** We show how privileged success and progress signals from a physics-enabled 3DGS twin can supervise a compact critic that transfers to real visual observations.
- **Real-world reward evaluation at scale.** Across eight tasks and two robot platforms, both critic variants outperform zero-shot and matched-data few-shot generalist reward models on reward alignment, trajectory ranking, and success–failure separation, while using substantially fewer parameters and achieving lower inference latency.
- **A reusable interface for policy improvement.** We use Sim2Critic to define returns for offline reinforcement learning on the policy's own real rollouts and to provide reward and success detection for online reinforcement learning in the frozen policy's latent-noise space, improving the base policy across diverse manipulation tasks.

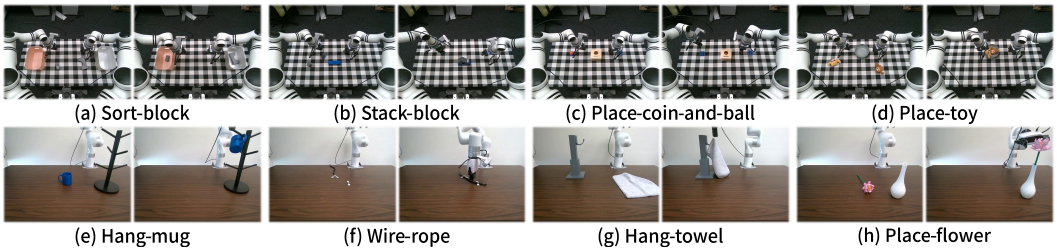


Figure 2: **Task suite.** We illustrate the initial and desired goal states for each task.

2 Related Work

Physical digital twins 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) has made it practical to transform a short capture of a robot workspace into a photorealistic replica. A growing body of real-to-sim work uses it to construct digital twins for manipulation. SplatSim (Qureshi et al., 2025) combines photorealistic rendering with physics simulation to transfer RGB policies zero-shot; RialTo (Torne et al., 2024) improves policy robustness through a real-to-sim-to-real pipeline; and RoboGSim (Li et al., 2024) synthesizes and evaluates demonstrations within the same loop. GSWorld (Jiang et al., 2026), PolaRiS (Jain et al., 2026), and Zhang et al. (2026b) use closed-loop splatting simulators as evaluation environments whose outcomes track real-robot performance. GS-Playground (Jia et al., 2026) combines parallel physics simulation with batched 3DGS rendering, enabling large-scale visual reinforcement learning across locomotion, navigation, and manipulation tasks. In short, prior work uses digital twins primarily to train policies, generate training data, or evaluate policy performance. In contrast, we use privileged simulator state to supervise a transferable vision critic, providing rollout outcomes, their timing, and per-step task progress without manual annotation. The twin thus serves as a *critic factory*, while also supporting policy diagnosis and targeted data generation.

Learned reward models Reward and value models for robot learning can be organized by their primary source of supervision: *human feedback*, *existing data*, or *simulation*.

Human feedback. These methods learn from real-world rollouts scored by human-specified rewards. Examples include the critics in $\pi^*_{0.6}$ (Physical Intelligence et al., 2026) and RL Token (Xu et al., 2026); the latter requires a human-assigned success label after each online episode.

Existing data. These methods derive reward or progress signals from demonstrations, video corpora (Tan et al., 2026), or pretrained models (Ma et al., 2025; Chen et al., 2026b) rather than requesting new rollout labels. SRPO (Fei et al., 2026) estimates progress relative to successful in-batch examples using frozen V-JEPA 2 embeddings (Assran et al., 2025), while other methods synthesize failures from demonstrations to learn dense rewards (Fang et al., 2026; Zhang et al., 2025). At broader scale, VLAC (Zhang et al., 2026c), Robometer (Liang et al., 2026), and RoboReward (Lee et al., 2026) rely on cross-embodiment pretraining. World Value Models (Wang et al., 2026) learn per-frame values from action-free videos using normalized elapsed-time targets, whereas SOLE-R1 (Schroeder et al., 2026) uses video–language reasoning to predict task progress as the sole reward for online RL.

Simulation. These methods transfer value signals learned in simulation to real-world policy improvement. Simulation-Guided Fine-Tuning (SGFT) (Yin et al., 2025) transfers an RL-trained value function for potential-based reward shaping and short-horizon policy optimization, assuming access to states and task rewards during real-world adaptation. Our approach falls within this category but targets the gap in observability: Sim2Critic learns a vision critic by supervised regression on scripted and perturbed digital-twin rollouts. Privileged success predicates or progress ladders provide per-step targets; the deployed critic uses only RGB observations and proprioception. All failure examples come from the twin, replacing manual rollout annotation with scene-specific simulator supervision without requiring a generalist reward-training corpus covering the deployment scene.

Improving generalist policies A frozen generalist policy can be improved through several interfaces to a value signal: $\pi^*_{0.6}$ conditions the policy on critic-estimated advantages (Physical Intelligence et al., 2026); Flow Reversal Steering (Tang et al., 2026) nudges a flow policy's sampling at inference time; Simulation Distillation (Levy et al., 2026) transfers sim-pretrained value heads that a planner consumes as rankings; RL Token (Xu et al., 2026) trains a lightweight actor–critic on a readout token

and refines the frozen VLA's action chunk online, anchored to it; and OmniReset (Yin et al., 2026) uses a privileged critic as scaffolding for simulation RL, discarding it after the actor is distilled. Each commits to one interface a priori. We instead hold a single twin-grounded critic fixed and ask whether it improves a frozen generalist policy through two interfaces. The critic defines the return for offline reinforcement learning on the policy's own real rollouts and, on a subset of tasks, supplies both the reward and the success detector for online reinforcement learning in the policy's latent-noise space.

3 Problem Setup

3.1 Critic Objective

We formulate each manipulation task as an episodic decision process $(\mathcal{S}, \mathcal{A}, P, r, H)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics P, reward r, and horizon H. Our approach exploits an asymmetry in observability: simulation exposes the full state s_t, including object poses, contacts, and deformable configurations, whereas deployment provides only observations $o_t = (I_t^{1:C}, q_t)$ from C calibrated RGB cameras and proprioception. We use privileged simulator state to supervise a vision critic $V_\phi(o_t)$ that estimates task success and progress without access to that state. We study two variants that differ only in their targets on simulated rollouts; Figure 1 (left) illustrates their target curves for successful trajectories and failures at different task stages.

Sparse success supervision For each task, the twin evaluates a success predicate $g(s_t) \in \{0, 1\}$. An episode is successful if the predicate holds over a sustained run ending at the final frame; t^* denotes the onset of this run. Sim2Critic (Sparse) uses the resulting outcome and success timing to define the per-step target

$$y_t^{\text{sparse}} = \begin{cases} R\,\gamma^{\max(t^*-t,\,0)} & \text{on a successful episode,} \\ 0 & \text{on every frame of a failed episode,} \end{cases} \tag{1}$$

where γ is the discount factor and R is the value assigned at success; t and t^* index stored frames, so γ is applied per stored frame. This target increases as success approaches, remains at R thereafter, and assigns zero to failed episodes. Predicate definitions, timing rules, the resulting discount horizon per task, and shared constants are provided in Section B.1.

Dense progress supervision Sim2Critic (Dense) additionally uses a progress ladder $h(s_t) \in [0, 1]$ that measures completion of task-specific sub-goals. Its per-step target is

$$y_t^{\text{dense}} = R\,h(s_t). \tag{2}$$

The terminal rung matches the task's success criterion, so both variants share the same definition of completion. Notably, the dense target credits partial progress even in failed episodes and decreases when progress is undone. More details about the task-specific ladders are presented in Section B.2.

Training and transfer Both critics are trained by supervised regression. Training combines simulated rollouts with a fixed budget of at most N successful real teleoperated demonstrations per task, without manual outcome annotation. Since real demonstrations lack privileged state, both variants assign them sparse success targets; all failure examples come from the twin. For each task, the architecture, data, hyperparameters, training schedule, and model-selection rule are otherwise identical, isolating the effect of simulator supervision. At deployment, neither g nor h is available: the critic operates on observations alone, and we evaluate its value traces for reward alignment and outcome discrimination on real rollouts.

3.2 Manipulation Tasks

As shown in Figure 2, we evaluate eight tabletop manipulation tasks using two real robot platforms: the UFACTORY xArm 7 and the Realman RM65-B bimanual robot. Further details about the robot systems are provided in Section A.1. Each platform has a designated workspace equipped with a fixed scene camera and wrist-mounted cameras. We collect real-world teleoperated demonstrations for each task in its corresponding workspace (Section A.2). The task suite includes both rigid and deformable manipulation: (1) *Sort-blocks* (RM65-B, rigid): pick blocks of two shapes and place them

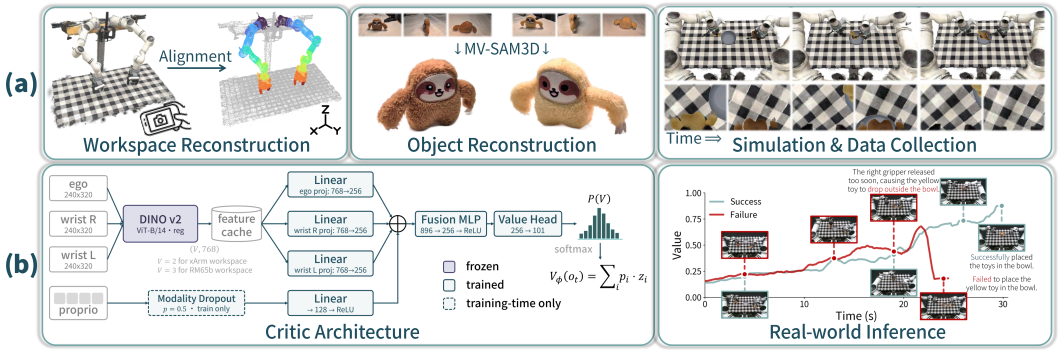


Figure 3: **Overview of Sim2Critic. (a) Twin construction and simulation.** We reconstruct and align the workspace, separately reconstruct task objects, and couple both to physics to collect photorealistic multi-view rollouts. **(b) Critic training and inference.** A frozen DINOv2 backbone encodes each camera view, and a lightweight head fuses the resulting features with proprioception to predict a value distribution. At deployment, its expectation provides a dense value that tracks task progress and distinguishes successful from failed real-world rollouts without privileged state.

into corresponding baskets. (2) *Stack-blocks* (RM65-B, rigid): stack two cuboids and a triangular roof onto a fixed base arch. (3) *Place-coin-and-ball* (RM65-B, rigid): post a coin through a slot and a ball through a circular opening of a square box, one object per arm. (4) *Place-toy* (RM65-B, soft): place two plush toys in a bowl. (5) *Hang-mug* (xArm, rigid): pick a mug and hang it on a branch of a mug tree. (6) *Wire-rope* (xArm, rope): thread the end of a rope through two table-mounted rings in sequence. (7) *Hang-towel* (xArm, cloth): drape a towel over a fixed hanger so that it remains hanging after release. (8) *Place-flower* (xArm, rigid): pick a flower by the stem and insert it into a vase.

Success for every task is a predicate on privileged simulator state (and operator labels on the real rig); the per-task success predicates, their hold requirements, and the simulated buffers each critic is trained on are described in Section B.1.

4 SIM2CRITIC

Sim2Critic is a framework for learning vision-based critics from privileged digital-twin supervision to assess task progress and outcomes on real robots. It comprises three stages: constructing a physics-enabled 3DGS twin, collecting and automatically labeling rollouts using scripted experts and failure templates, and training a critic that operates on real-world observations without privileged state. Figure 3 presents an overview of the proposed method.

4.1 3DGS TWIN

We construct a digital twin of each deployment workspace to train Sim2Critic. The twin has two purposes: it renders photorealistic observations from the calibrated deployment cameras, and it provides the privileged state and contact dynamics needed to evaluate task success at every step. Building the twin involves three steps. We first reconstruct the appearance and geometry of the workspace, align the reconstruction with the robot's metric coordinate frame, and then bind the reconstructed assets to physics in the Genesis simulator (Authors, 2024). Figure 3(a) illustrates this reconstruction-to-simulation pipeline.

Reconstruction We capture a short monocular video by moving an iPhone around the stationary robot platform. Scaniverse (Niantic, 2024) converts the video into a 3DGS reconstruction of the workspace. Using the SuperSplat interactive visualizer, we then isolate the Gaussian splats corresponding to the robot and table. Each task object is reconstructed separately using MV-SAM3D (Li et al., 2026). For example, the reconstructed mug used in *Hang-mug* consists of a collision mesh and Gaussian splats defined in the same local coordinate frame. The mesh provides collision geometry for physics simulation, whereas the splats represent the object's visual appearance.

Alignment We align the reconstructed workspace with the robot's metric frame and attach robot Gaussians to their corresponding links for motion driven by forward kinematics. We refine each arm's mount pose in the bimanual setup and calibrate object scales independently of task-specific poses. Registration, mount calibration, and scale estimation are detailed in Section B.4.

Physics binding Each task object has two aligned representations: a mesh used for collision detection and dynamics, and Gaussian splats used for rendering. As the physics engine updates the mesh pose and privileged state, the renderer applies the same pose to the splats and produces photorealistic observations from the calibrated deployment cameras. Our camera models include bundle-adjusted wrist-camera extrinsics, so that the rendered views stay faithful to the original scene. Simulated trajectories are rendered offline in batches, as detailed in Section B.5.

Deformables Cloth, rope, and plush toys require dynamics that are not captured by a rigid-body model. We simulate cloth using an incremental potential contact (IPC) solver (Li et al., 2021) and rope using a discrete elastic rod (DER) solver (Cao et al., 2026), and the plush toys of *Place-toy* using an elastic material-point-method (MPM) solver (Jiang et al., 2016). All are coupled to the same scene and rendered through the same pipeline.

4.2 Data Collection

Scripted experts We generate successful episodes using task-specific scripted experts. These waypoint-based controllers access the privileged twin state and execute each task across randomized object spawns. Before bulk data collection, we verify that each expert behaves deterministically under fixed initial conditions and exceeds a predefined success threshold across randomized spawns. This validation ensures sufficient buffer quality before committing substantial computational resources.

Failure templates Training a useful critic also requires diverse failure examples, but natural failures from the scripted experts are too rare and limited in variety. We introduce controlled interventions at task-specific phase anchors, such as offsetting a placement, releasing an object early, or deliberately missing a grasp. Importantly, the intended intervention does not determine the episode label. Instead, the simulator evaluates the measured outcome using the task's success predicate, because an intervention may have no effect or may still result in success. We measure the effectiveness of each intervention for every template–object pair and adjust the mixture of unmodified and intervened episodes for each task. The resulting buffer for each task contains between about 1K expert and intervened episodes, of which about $50 - 75\%$ are successful trajectories (see Table 3).

4.3 Critic Training

We train each critic by supervised regression using the targets defined in Section 3.1: the sparse success return of Equation (1) for Sim2Critic (Sparse) and the dense progress target of Equation (2) for Sim2Critic (Dense). Both variants share the architecture in Figure 3(b): a frozen DINOv2 ViT-B/14 backbone with registers (Oquab et al., 2024; Darcet et al., 2024) encodes each camera view, and a lightweight head fuses the multi-view features with proprioception to predict a value distribution. Its expectation is the critic value $V_\phi(o_t)$.

Training combines simulation rollouts with N successful real demonstrations per task. Simulation frames receive variant-specific targets, whereas real demonstrations receive sparse success targets in both variants. We provide more details on the critic architecture in Section B.6 and supplement additional information on optimization and model-selection in Section B.7.

5 Experiments

Our experiments address three questions: **(Q1) Reward evaluation**: does Sim2Critic evaluate progress and outcomes better than generalist reward models? **(Q2) Ablations**: what ingredients of the twin recipe does that quality rest on? and **(Q3) Policy improvement**: can Sim2Critic improve a generalist policy through offline and online RL? For Q1 and Q2, all models use the same operator-labeled banks of real rollouts from four generalist policies (Section D.1), none of which are used for critic training. Please refer to the supplementary materials for video results.

Metric	VLAC (Zhang et al., 2026c)	SOLE-R1 (Schroeder et al., 2026)	RoboReward (Lee et al., 2026)	Robometer (Liang et al., 2026)	RoboReward (Few-shot)	Robometer (Few-shot)	Sim2Critic (Sparse)	Sim2Critic (Dense)
VOC $\rho \uparrow$	0.775†	0.641	0.846	0.684	0.901	0.794	0.926	**0.932**
AUROC $\uparrow$	0.576	0.542	0.761	0.838	0.861	0.912	0.941	**0.943**
Succ–Fail $d \uparrow$	0.35†	0.43	1.23	1.26	1.67	1.85	2.45	**2.50**

Table 1: **Reward evaluation on real rollouts.** Results are averaged across all eight tasks. †: VLAC returns a constant value on the four RM65-B tasks, where VOC and Succ–Fail d are undefined, so those two values are taken over the remaining four tasks. Per-task results are reported in Table 8.

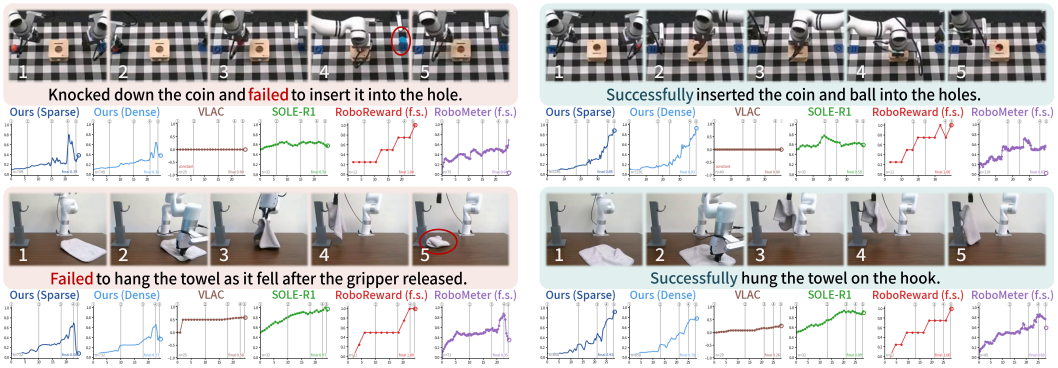


Figure 4: **Qualitative reward evaluation.** Top: *Place-coin-and-ball*; Bottom: *Hang-towel*.

5.1 Q1: Reward Evaluation

Reward models compared We compare Sim2Critic against four generalist reward models in their released zero-shot configurations: VLAC (8B) (Zhang et al., 2026c), SOLE-R1 (Schroeder et al., 2026), RoboReward (8B) (Lee et al., 2026), and Robometer (Liang et al., 2026). We also evaluate the few-shot adaptation modes of RoboReward and Robometer, denoted RoboReward (Few-shot) and Robometer (Few-shot). Both use exactly the same real demonstrations as Sim2Critic for fine-tuning the reward models. All models are evaluated on the same rollout episodes.

Metrics Following Robometer (Liang et al., 2026), we evaluate three complementary properties. *Reward alignment* is measured by Value Order Correlation (VOC), the rank correlation between predicted values and frame order on successful rollouts, introduced by GVL (Ma et al., 2025). *Trajectory ranking* is measured by outcome AUROC, which evaluates whether final scores distinguish successful from failed rollouts. *Success–failure separation* is measured by standardized Succ–Fail d, which quantifies the margin between their final-score distributions. Full definitions, model-specific outcome scores, and the scope of each metric are provided in Section D.1.

Analysis Both variants of Sim2Critic outperform all generalist baselines based on the aggregate metrics shown in Table 1, including few-shot models trained on the same real demonstrations. Among the baselines, RoboReward provides strong reward alignment, while Robometer excels in outcome discrimination. However, neither of these models matches our critics on both dimensions. Sim2Critic combine these strengths, with the dense variant holding a small advantage across all three metrics. Qualitatively, Figure 4 shows that both critics rise toward completion on successful *Place-coin-and-ball* and *Hang-towel* rollouts, but decline when the gripper knocks down the coin or the towel falls, correctly reflecting lost progress. RoboReward (Few-shot), by contrast, assigns high final values even to failed examples, while Robometer (Few-shot) produces similar progress traces for successful and failed rollouts despite its better overall outcome discrimination. We supplement more qualitative results for other tasks in Figure 10. These gains also come with a lower inference cost: Table 9 in Section D.2 reports approximately 87M parameters and 7.81 ms per value end to end—over $50\times$ fewer parameters and approximately $6.4\times$ lower per-value latency than Robometer, the smallest and fastest generalist baseline—while scoring every recorded frame.

5.2 Q2: Ablations

We conduct ablation studies on Sim2Critic (Dense) on four manipulation tasks: two from the RM65-B workspace (*Sort-blocks* and *Place-toy*); two from the xArm workspace (*Hang-mug* and *Place-flower*).

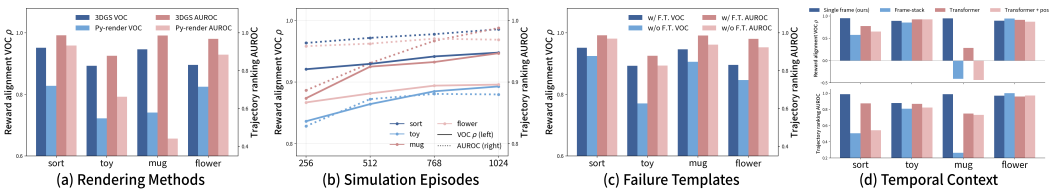


Figure 5: **Ablation analysis.**

Each ablation moves one factor of the recipe at a time and holds the rest of it fixed. We report reward alignment and trajectory ranking scores for each variant on the same banks of real rollouts.

(a) Does the photorealism of the twin's rendering matter? We replace the Gaussian-splat renderer of Section 4.1 with the simulator's mesh rasterizer, keeping the state trajectories, cameras, buffer, targets, and real demonstrations fixed. Only the training images change, from the reconstructed workspace to rasterized meshes on a synthetic ground plane. Figure 5(a) shows that 3DGS rendering improves both reward alignment and trajectory ranking on all four tasks. The largest degradation without 3DGS occurs on *Hang-mug*, where AUROC falls from 0.987 to 0.442. These results indicate that visual fidelity is important for transferring state-derived supervision to real observations.

(b) How many simulation episodes does a transferable critic need? We vary the number of simulation episodes used for training our critic from 256 to $1{,}024$, preserving the success-to-failure ratio. As shown in Figure 5(b), even 256 episodes yield useful progress and outcome predictions, while additional data improve reward alignment on all four tasks. Outcome discrimination also generally improves, with the largest gain on *Hang-mug*. However, AUROC on *Place-toy* and *Place-flower* largely plateaus by 768 episodes. Thus, a few hundred simulation episodes already suffice, with task-dependent benefits from scaling further.

(c) Do failure templates improve critic performance? We disable the failure templates used in data collection (Section 4.2) and collect the same number of simulation episodes, retaining only naturally occurring failures. All other training settings remain unchanged. Figure 5(c) shows that failure templates improve both metrics on every task. The largest reward-alignment gain occurs on *Place-toy*, where VOC increases from 0.771 to 0.893, alongside an AUROC increase from 0.827 to 0.879. Controlled failure interventions therefore provide useful supervision beyond natural simulation failures, improving both reward alignment and trajectory ranking rather than merely increasing the amount of training data.

(d) Does temporal modeling improve critic performance? We replace the frame-wise head (Section 4.3) with an ordered frame stack or a small transformer, with or without learned positional embeddings, using the same lookback window. Figure 5(d) shows no consistent benefit: temporal heads modestly improve reward alignment on *Place-toy* and *Place-flower*, but all underperform the frame-wise critic on both metrics on *Sort-blocks* and *Hang-mug*. On *Hang-mug*, the frame stack and positional transformer yield negative VOC, indicating declining values over successful rollouts. Removing positional information mitigates this degradation but does not recover frame-wise performance. One possible explanation is sensitivity to event timing rather than persistent task state: the transformer's values peak during hanging and decline afterward. Differences in simulation and real rollout pacing may further hinder transfer of fixed-window temporal features. We therefore retain the simpler frame-wise head for its more reliable performance across tasks, while leaving a dedicated temporal progress head to future work to explore the task-dependent gains in reward alignment.

5.3 Q3: Policy Improvement

Offline RL For each task, we fine-tune $\pi_{0.5}$ (Black et al., 2025a) on real teleoperated demonstrations and use the policy's own real-robot rollouts as offline experience. We estimate chunk-level Monte Carlo advantages using a cross-fitted expectile value baseline (Kostrikov et al., 2022), convert them into per-frame weights through advantage-weighted regression (Peng et al., 2019), and fine-tune the policy under its original flow-matching objective. A fixed subset of the teleoperated demonstrations is included only for rehearsal, not as RL experience. Section C.1 details the rollout pools, value fitting,

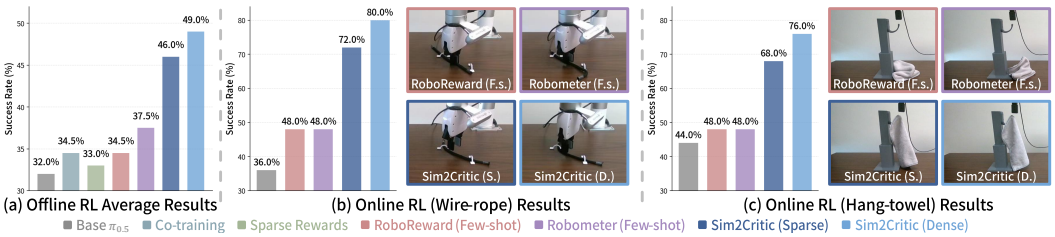


Figure 6: **Policy improvement results**.

reward construction, rehearsal, optimization, and evaluation protocol. The five offline-RL variants in Figure 6(a) differ only in the reward used to define returns. **Sparse Reward** uses operator-provided episode outcomes. RoboReward (Few-shot), Robometer (Few-shot), and the Sim2Critic variants instead combine their own dense progress values with terminal outcome scores, without human labels. We also report the unchanged base policy and a **Co-training** baseline trained with 500 additional successful twin demonstrations per task. All policies are evaluated blindly on the real robot with comparable object placements.

As shown in Figure 6(a), both Sim2Critic variants substantially outperform the base policy and all reward baselines. Sim2Critic (Dense) achieves the highest average success rate at 49.0%, compared with 32.0% for the base policy and 37.5% for the strongest generalist reward model. By contrast, operator-labeled sparse rewards and co-training provide only modest gains. We present per-task results in Section D.3: both Sim2Critic variants improve over the base policy on all eight tasks.

Online RL On *Wire-rope* and *Hang-towel*, we perform real-robot online RL with DSRL (Wagenmaker et al., 2025), which trains a soft actor–critic agent (Haarnoja et al., 2018) to modify the latent noise sampled by the frozen flow policy. Each reward model supplies both dense rewards and success detection, with all detectors calibrated on the same rollout data under the same objective. The operator provides no reward annotation and acts only as a supervisor: they may overrule a success declaration before the episode ends, and stop an episode for safety. Figure 6(b-c) compares the four reward models under this shared loop; Section C.2 provides more details on the experimental setting.

Both Sim2Critic variants outperform the base policy and the generalist reward models on both tasks. Sim2Critic (Dense) achieves 80% success on *Wire-rope* and 76% on *Hang-towel*, compared with base-policy rates of 36% and 44%—improvements of $2.2\times$ and $1.7\times$, respectively. Sim2Critic (Sparse) also reaches 72% and 68%, while both generalist-reward variants reach 48% on each task. The gain on *Wire-rope* is particularly notable because it is a long-horizon task that requires repeated replanning as the rope deforms during threading. The dense critic performs best on both tasks, suggesting that progress-sensitive feedback benefits online exploration; the strong sparse-critic results show that twin-grounded sparse success supervision also provides an effective learning signal.

6 Conclusion

We introduced *Sim2Critic*, a real-to-sim-to-real approach that turns a short workspace scan into a physics-enabled 3DGS twin, leveraging its privileged success and progress signals to supervise a compact real-world vision critic. With as few as ten successful demonstrations per task, both critic variants outperform zero-shot and matched-data few-shot generalist reward models across eight tasks and two robot platforms, while using over $50\times$ fewer parameters and approximately $6.4\times$ lower per-value latency. Furthermore, the fixed critic raises the average offline-RL success rate from 32.0% to 49.0% and achieves 80% and 76% success with online DSRL on *Wire-rope* and *Hang-towel*, respectively. Collectively, these results demonstrate that scene-grounded automatic evaluation effectively measures and drives policy improvement across diverse real-world manipulation tasks. By closing the loop between experience collection, outcome assessment, policy updates, and validation, such evaluators represent a critical step toward autonomous and potentially recursive robot self-improvement (Zhang et al., 2026a; Xiao et al., 2026; Yang et al., 2026; Pachocki, 2026; Chen et al., 2026a). Looking forward, Sim2Critic establishes a robust foundation for this paradigm, opening avenues for automatic task specification, continual twin updates in dynamic environments, and safeguards against reward exploitation across iterative policy-improvement cycles.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing the scripted experts and failure templates described in Section 4.2, providing feedback and interpreting results from the ablation and real-world experiments, polishing the manuscript, and cleaning and reformatting teleoperation and simulation data. We did not use these tools to develop theoretical or conceptual frameworks, formulate or prove mathematical claims, support qualitative or thematic data analysis, or propose or refine research hypotheses.

All AI-assisted outputs were reviewed by two authors. We verified that generated text was consistent with the experimental evidence and inspected the outputs of AI-assisted code to check its correctness. We take full responsibility for all text, claims, code, and artifacts presented in this work.

REFERENCES

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL `https://github.com/Genesis-Embodied-AI/genesis-world`.

Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE TPAMI*, 14(2): 239–256, 1992.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *CoRL*, 2025a.

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 *RSS*, 2025b.

Junyi Cao, Yian Wang, Ziyan Xiong, Chunru Lin, Zhehuan Chen, and Chuang Gan. DLO-lab: Benchmarking deformable linear object manipulations with differentiable physics. In *ICML*, 2026.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops. *arXiv preprint arXiv:2607.07663*, 2026a.

Shirui Chen, Cole Harrison, Ying-Chun Lee, Angela Jin Yang, Zhongzheng Ren, Lillian J. Ratliff, Jiafei Duan, Dieter Fox, and Ranjay Krishna. TOPReward: Token probabilities as hidden zero-shot rewards for robotics. *arXiv preprint arXiv:2602.19313*, 2026b.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *ICLR*, 2024.

Yu Fang, Wanxi Dong, Jiaqi Liu, Yue Yang, Mingxiao Huo, Yao Mu, Huaxiu Yao, Li Erran Li, Daniel Szafir, and Mingyu Ding. DenseReward: Dense reward learning via failure synthesis for robotic manipulation. *arXiv preprint arXiv:2607.13033*, 2026.

Senyu Fei, Siyin Wang, Li Ji, Ao Li, Shiduo Zhang, Liming Liu, Jinlong Hou, Jingjing Gong, Xianzhong Zhao, and Xipeng Qiu. SRPO: Self-referential policy optimization for vision-language-action models. In *CVPR*, 2026.

Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24 (6):381–395, 1981.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Zhi Hou, Tianyi Zhang, Yuwen Xiong, Haonan Duan, Hengjun Pu, Ronglei Tong, Chengyang Zhao, Xizhou Zhu, Yu Qiao, Jifeng Dai, et al. Dita: Scaling diffusion transformer for generalist vision-language-action policy. In *ICCV*, 2025.

Arhan Jain, Mingtong Zhang, Kanav Arora, William Chen, Marcel Torne, Muhammad Zubair Irshad, Sergey Zakharov, Yue Wang, Sergey Levine, Chelsea Finn, Wei-Chiu Ma, Dhruv Shah, Abhishek Gupta, and Karl Pertsch. PolaRiS: Scalable real-to-sim evaluations for generalist robot policies. In *RSS*, 2026.

Yufei Jia, Heng Zhang, Ziheng Zhang, Junzhe Wu, Mingrui Yu, Zifan Wang, Dixuan Jiang, Zheng Li, Chenyu Cao, Zhuoyuan Yu, et al. GS-playground: A high-throughput photorealistic simulator for vision-informed robot learning. In *RSS*, 2026.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *SIGGRAPH 2016 courses*. ACM, 2016.

Guangqi Jiang, Haoran Chang, Ri-Zhao Qiu, Yutong Liang, Mazeyu Ji, Jiyue Zhu, Zhao Dong, Xueyan Zou, and Xiaolong Wang. GSWorld: Closed-loop photo-realistic simulation suite for robotic manipulation. In *ICRA*, 2026.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM TOG*, 42(4), 2023.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *ICLR*, 2022.

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.

Jacob Levy, Tyler Westenbroek, Kevin Huang, Fernando Palafox, Patrick Yin, Shayegan Omidshafiei, Dong-Ki Kim, Abhishek Gupta, and David Fridovich-Keil. Simulation distillation: Pretraining world models in simulation for rapid real-world adaptation. In *RSS*, 2026.

Baicheng Li, Dong Wu, Jun Li, Shunkai Zhou, Zecui Zeng, Lusong Li, and Hongbin Zha. MV-SAM3D: Adaptive multi-view fusion for layout-aware 3D generation. *arXiv preprint arXiv:2603.11633*, 2026.

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. Codimensional incremental potential contact. *ACM TOG*, 40(4), 2021.

Xinhai Li, Jialin Li, Ziheng Zhang, Rui Zhang, Fan Jia, Tiancai Wang, Haoqiang Fan, Kuo-Kun Tseng, and Ruiping Wang. RoboGSim: A Real2Sim2Real robotic Gaussian splatting simulator. *arXiv preprint arXiv:2411.11839*, 2024.

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 Bıyık, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. In *RSS*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision language models are in-context value learners. In *ICLR*, 2025.

Niantic. Scaniverse, 2024. URL `https://scaniverse.com`.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *TMLR*, 2024.

Jakub Pachocki. An alien mind. OpenAI, `https://openai.com/index/an-alien-mind/`, 2026. September 6, 2026.

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.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, et al. $\pi^*_{0.6}$: a VLA that learns from experience. In *RSS*, 2026.

M Nomaan Qureshi, Sparsh Garg, Francisco Yandun, David Held, George Kantor, and Abhisesh Silwal. SplatSim: Zero-shot Sim2Real transfer of RGB manipulation policies using Gaussian splatting. In *ICRA*, 2025.

Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *ICRA*, 2009.

Philip Schroeder, Thomas Weng, Karl Schmeckpeper, Eric Rosen, Stephen Hart, and Ondrej Biza. SOLE-R1: Video-language reasoning as the sole reward for on-robot reinforcement learning. *arXiv preprint arXiv:2603.28730*, 2026.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

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 *CVPR*, 2026.

Andy Tang, William Chen, Andrew Wagenmaker, Chelsea Finn, and Sergey Levine. Improving robotic generalist policies via flow reversal steering. *arXiv preprint arXiv:2606.13675*, 2026.

Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. In *RSS*, 2024.

Andrew Wagenmaker, Yunchu Zhang, Mitsuhiko Nakamoto, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *CoRL*, 2025.

Ruicheng Wang, Sicheng Xu, Cassie Dai, Jianfeng Xiang, Yu Deng, Xin Tong, and Jiaolong Yang. MoGe: Unlocking accurate monocular geometry estimation for open-domain images with optimal training supervision. In *CVPR*, 2025.

Zhihao Wang, Jianxiong Li, Yu Cui, Yuan Gao, Xianyuan Zhan, Junzhi Yu, and Xiao Ma. World Value Models for robotic manipulation. *arXiv preprint arXiv:2606.24742*, 2026.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, et al. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

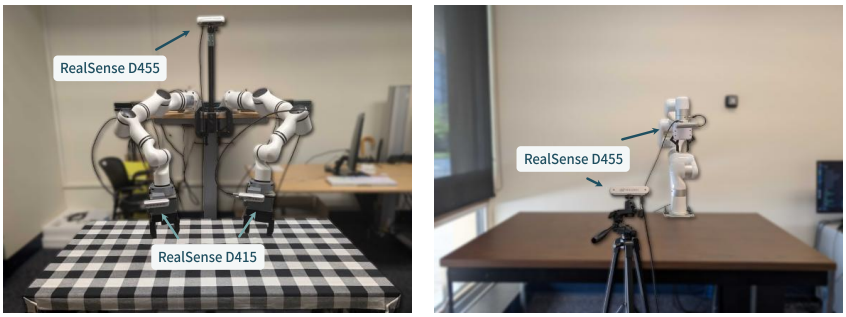
Charles Xu, Jost Tobias Springenberg, Michael Equi, Ali Amin, Adnan Esmail, Sergey Levine, and Liyiming Ke. RL token: Bootstrapping online RL with vision-language-action models. *arXiv preprint arXiv:2604.23073*, 2026.

Jiazhi Yang, Kunyang Lin, Jinwei Li, Wencong Zhang, Tianwei Lin, Longyan Wu, Zhizhong Su, Hao Zhao, Ya-Qin Zhang, Li Chen, et al. RISE: Self-improving robot policy with compositional world model. *arXiv preprint arXiv:2602.11075*, 2026.

Patrick Yin, Tyler Westenbroek, Ching-An Cheng, Andrey Kolobov, and Abhishek Gupta. Rapidly adapting policies to the real-world via simulation-guided fine-tuning. In *ICLR*, 2025.

Patrick Yin, Tyler Westenbroek, Zhengyu Zhang, Ignacio Dagnino, Eeshani Shilamkar, Numfor Mbiziwo-Tiapo, Simran Bagaria, Xinlei Liu, Galen Mullins, Andrey Kolobov, and Abhishek Gupta. Emergent dexterity via diverse resets and large-scale reinforcement learning. In *ICLR*, 2026.

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J. Lim, Jesse Thomason, Erdem Bıyık, and Jesse Zhang. ReWiND: Language-guided rewards teach robot policies without new demonstrations. In *CoRL*, 2025.



(a) Realman RM65-B (b) UFACTORY xArm 7

Figure 7: **Robot system overview.** In our real-world experiments, we use two robot platforms: (a) The Realman RM65-B, outfitted with a fixed scene camera and two wrist cameras. The scene camera is a RealSense D455, while the two wrist cameras are RealSense D415 models. (b) The UFACTORY xArm 7, which is equipped with a third-person scene camera and a wrist camera, both of which are RealSense D455 models.

Jianshu Zhang, Keliang Wu, Haoran Lu, Anbang Liu, Ce Zhang, Weijie Yin, Chengxuan Qian, Xiyuan Yang, Zhenyu Pan, Guo Ye, and Han Liu. Progress reward modeling for robotic learning: A comprehensive survey. *arXiv preprint arXiv:2607.21655*, 2026a.

Kaifeng Zhang, Shuo Sha, Hanxiao Jiang, Matthew Loper, Hyunjong Song, Guangyan Cai, Zhuo Xu, Xiaochen Hu, Changxi Zheng, and Yunzhu Li. Real-to-sim robot policy evaluation with Gaussian splatting simulation of soft-body interactions. In *ICRA*, 2026b.

Qi Zhang, Shaopeng Zhai, Shengzhe Zhang, Litao Liu, Tianyi Zhang, Fuxian Huang, and Ming Zhou. A generalist pair-wise progress critic model for vision-language-action robots. In *ICML*, 2026c.

A Experimental Setup

A.1 Robot System Details

As shown in Figure 7, our real-world experiments use two robot platforms. The Realman RM65-B platform is a bimanual system comprising two 6-DoF arms, for a total of 12 arm DoFs, with one gripper per arm. We equip this platform with a fixed RealSense D455 scene camera and two RealSense D415 wrist cameras, one mounted on each arm. Together, the scene and wrist views provide complementary global and local observations of object interactions. The UFACTORY xArm 7 is a single-arm manipulator with 7 degrees of freedom (DoFs) and a gripper. Its workspace is observed by two Intel RealSense D455 cameras: a fixed third-person scene camera and a wrist-mounted camera.

A.2 Teleoperation Details

We collect real-world demonstrations using a platform-specific teleoperation interface. For the bimanual Realman RM65-B platform, the operator uses a Meta Quest 2 and its two handheld controllers to command both robot arms simultaneously. The left and right controllers are mapped to the corresponding robot end effectors, with controller motion specifying relative 6-DoF pose changes. Holding a controller trigger engages motion and re-establishes the pose reference, allowing the operator to pause and resume without discontinuities; the grip input controls the corresponding gripper. We use this interface to collect demonstrations for *Sort-blocks*, *Stack-blocks*, *Place-coin-and-ball*, and *Place-toy*.

For the UFACTORY xArm 7 platform, we use an Android phone as a 6-DoF motion controller. The phone streams its pose through a WebXR interface over the local network, and its relative motion is mapped to end-effector translation and rotation. A press-and-hold input engages robot motion and resets the reference pose upon each engagement, while on-screen buttons control the gripper. We use this interface for *Hang-mug*, *Wire-rope*, *Hang-towel*, and *Place-flower*.

Tasks	Sort-blocks	Stack-blocks	Place-c.-b.	Place-toy	Hang-mug	Wire-rope	Hang-towel	Place-flower
#Episodes	240	240	240	200	200	206	201	206
#Frames	116,415	207,723	130,551	90,002	86,238	346,342	119,742	161,817
Duration	64.7 min	115.4 min	72.5 min	50.0 min	47.9 min	192.4 min	66.5 min	89.9 min

Table 2: **Statistics of real-world demonstrations for fine-tuning generalist policies.** We use our teleoperation system and manually collect 200-240 episodes for each manipulation task.

Task	Platform	Cam.	Success predicate on privileged state	Hold	Frame	Simulated buffer		Train / val	N
				(steps / s)	period (s)	episodes	successful	episodes	
Sort-blocks	RM65-B	3	both blocks overlap their own basket by at least 30%	2 / 0.04	0.02	1,264	791 (63%)	928 / 336	10
Stack-blocks	RM65-B	3	three blocks upright within pose tolerance on a fixed base arch	1 / 0.01	0.01	1,024	647 (63%)	816 / 208	10
Place-c.-b.	RM65-B	3	coin and ball each inside the box through their own opening	1 / 0.01	0.01	1,224	634 (52%)	979 / 245	10
Place-toy	RM65-B	3	both toys at least 80% inside the bowl, settled and released	1 / 0.01	0.01	1,068	664 (62%)	851 / 217	10
Hang-mug	xArm	2	mug lifted and its handle hole seated on the branch	4 / 0.12	0.03	1,024	690 (67%)	807 / 217	16
Wire-rope	xArm	2	both rings threaded in order and the rope released	30 / 0.9	0.09	1,024	614 (60%)	816 / 208	10
Hang-towel	xArm	2	towel still hanging on the hanger after the gripper releases	180 / 18	0.1	1,024	504 (49%)	780 / 244	10
Place-flower	xArm	2	flower stem inserted through the vase mouth to depth	1 / 0.03	0.03	1,024	772 (75%)	808 / 216	10

Table 3: **Success definitions, simulation buffers, and real-data budgets for each task.** *Cam.* denotes the number of calibrated camera views used by the critic; all observations are stored at 320×240. The twin evaluates each success predicate on privileged state at every simulation step. *Hold* gives the required consecutive duration in steps and seconds, with one step indicating immediate success. *Frame period* is the simulated time between stored observations, and *Successful* reports the number and fraction of buffer episodes satisfying the predicate. Training and validation are split by complete collection batches without overlap. N is the number of successful real teleoperated demonstrations used for critic training.

We record synchronized visual and proprioceptive observations at 30 Hz. For the RM65-B, each observation contains three RGB views—the scene camera and the left and right wrist cameras—together with a 28-dimensional robot state. The state concatenates, for each arm, the 3D tool-center-point (TCP) position, TCP orientation represented as a quaternion, six joint angles, and gripper state. For the xArm 7, each observation contains the scene and wrist RGB views and a 15-dimensional state comprising the TCP position and quaternion orientation, seven joint angles, and gripper state. All recorded RGB streams are stored at a resolution of 320×240 pixels. The corresponding actions specify target TCP poses and gripper commands for one or both arms, depending on the platform.

Across the eight tasks, we manually collect 200–240 episodes per task; the resulting numbers of episodes and frames, together with the total demonstration durations, are reported in Table 2.

B Implementation Details

This section supplies more implementation details on supervision targets and task-specific buffers, digital-twin alignment and rendering, and the critic training procedure.

B.1 Success Labels and Simulated Buffers

The twin evaluates task-specific success predicates on privileged simulator state. These include placement and containment conditions for resting configurations, as well as event-based conditions such as cloth retention. Table 3 lists each predicate, its hold requirement, the stored-frame period, the simulated buffer and its split, and the real demonstration budget.

Success timing and sparse targets All tasks use $\gamma = 0.995$ and $R = 1$ in Equation (1). In addition to satisfying the task-specific hold requirement in Table 3, a successful episode must end with at least five consecutive stored frames labeled true; the onset of this final run defines t^*. Although hold durations and frame periods vary by task, this five-frame rule is shared across all tasks. Because the discount is applied once per stored frame, its physical-time horizon depends on the frame period. The target half-life is 138 stored frames, corresponding to 0.28 s for *Place-coin-and-ball* ($\Delta t = 0.002$ s), 13.8 s for *Hang-towel* ($\Delta t = 0.1$ s), and 4.6 s for the 30 Hz real demonstrations. To test sensitivity to this choice, we retrained the sparse critic on all eight tasks using the physical-time discount

Task	Scored per	K_h	Rungs, in order
Sort-blocks	cuboid, arch	3	approach, lifted, transport, inside
Stack-blocks	two cuboids, triangular roof	2	pre-grasp, carry, placed
Place-c.-b.	coin, ball	4	approach, grasped, transport, aligned, inserted
Place-toy	two plush toys	3	approach, grasped, transport, inside
Hang-mug	the mug	3	approach, grasped, align, hung
Wire-rope	the rope	4	approach, grasped, first ring threaded, second ring threaded in order, released
Hang-towel	the towel	3	approach, lifted, aligned, hung
Place-flower	the flower	4	approach, grasped, transport, aligned, inserted

Table 4: **Task-specific progress ladders.** For each scored object, the ladder in Equation (2) contains K_h+1 ordered milestones defining K_h progress intervals. The initial milestone holds by construction, while the terminal milestone matches the success predicate in Table 3; therefore, the dense target reaches one exactly when the sparse success criterion is satisfied. Within each interval, a linearly mapped and clipped object-centric distance measures progress toward the next milestone. Object scores are then averaged. For *Wire-rope*, the two threading milestones are credited only when completed in order.

$0.995^{30\,\Delta t}$ per stored frame, which gives every task the same 4.6 s half-life as the real demonstrations. On average, this alternative decreases trajectory ranking by 0.013, reward alignment by 0.023, and Succ–Fail d by 0.11, with no task improving. We therefore retain the per-frame discount in the reported results.

Buffer construction and splits Simulation observations are rendered with 3DGS and stored at 320×240 pixels. Figures 8 and 9 show several failed and successful trajectories from the RM65-B and xArm workspaces, respectively. Training and validation are split by whole collection batch, with no batch shared between them.

B.2 Dense Progress Targets

The progress ladder $h(s_t)$ in Equation (2) depends only on the current privileged state, not on the frame index, episode history, or a latched record of previous progress. It is evaluated separately for each manipulated object and then averaged. Each object-specific ladder contains $K_h + 1$ ordered milestones defining K_h intervals. The stage is the index of the highest milestone currently satisfied, allowing the score to decrease immediately when progress is undone. Within the current interval, one object-centric distance is linearly normalized between task-specific bounds and clipped to $[0, 1]$ to obtain the fractional progress. The resulting object score is $(\text{stage} + \text{fraction})/K_h$. At the terminal milestone, the stage is K_h and the fraction is zero; because its predicate matches the task success predicate in Table 3, $h(s_t) = 1$ exactly when that predicate holds. Thus, the sparse and dense targets share the same definition of completion, although the hold requirement applies only to the sparse success label and not to the instantaneous dense target. For deformable tasks, the same procedure computes milestones and distances from cloth vertices or ring crossings. Table 4 lists the scored objects and milestones; for *Wire-rope*, the two threading milestones must be completed in order.

B.3 Real Demonstration Supervision

Real supervision is limited to the successful teleoperated demonstrations listed in Table 3: $N = 10$ for seven tasks and $N = 16$ for *Hang-mug*. These demonstrations are successful by construction and require no additional human outcome labels. Because they contain no privileged state or per-step success predicate, we set t^* to the final frame and apply the success branch of Equation (1) to every frame. This rule is identical for Sim2Critic (Sparse) and Sim2Critic (Dense); dense progress targets are used only for simulated observations. Thus, real supervision is one-sided, and all failure examples come from the twin. Baselines that use real demonstrations receive the same episodes and budget.

B.4 Digital-Twin Alignment

The workspace reconstruction initially lies in the scanner's coordinate frame, whereas simulation, robot kinematics, and camera calibration are defined in the robot's metric frame. We first level the workspace by fitting a plane to the tabletop with RANSAC (Fischler & Bolles, 1981), aligning it with the horizontal plane, and setting its height to zero. We identify the robot Gaussians using either

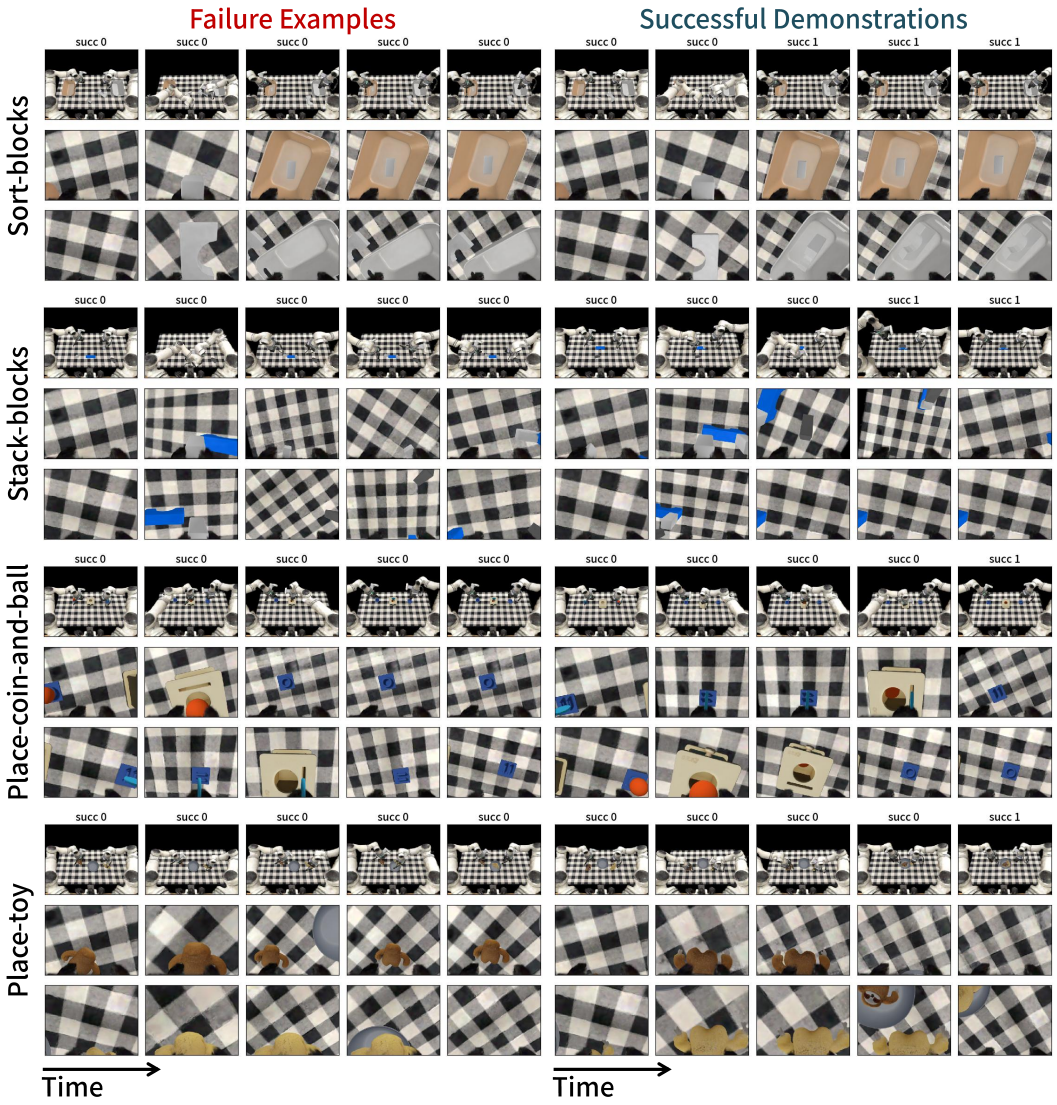


Figure 8: **Simulation data for the RM65-B workspace.** Example 3DGS renderings of failed trajectories (left) and successful demonstrations (right) for the four bimanual tasks. Each sequence shows synchronized scene and wrist-camera observations at five timestamps; *succ* indicates the privileged success predicate evaluated by the simulator.

a robot-only capture or a spatial bounding region. Next, we configure the simulated robot using the joint positions recorded during scanning and sample points from the robot's link meshes. We align the scanned robot with these sampled points using feature-based global registration (Rusu et al., 2009), followed by ICP refinement (Besl & McKay, 1992). Finally, we associate each robot Gaussian with its nearest link. Forward kinematics can then move the Gaussians with the corresponding links, producing consistent renderings of continuous arm and gripper motion.

The bimanual workspace requires an additional calibration step because the two arms are mounted independently and cannot be aligned accurately with a single rigid transform. We first register the scan against both arms jointly and then refine the mount pose of each arm separately to account for assembly error. Object reconstructions also require scale calibration: multi-view reconstruction produces objects in a canonical, non-metric scale. We recover the metric scale of each object by aligning its reconstruction with a metric monocular point map (Wang et al., 2025) estimated from a scene image. This procedure determines only object scale; object poses are specified independently by the task layouts and spawn randomization described in Section 4.2.

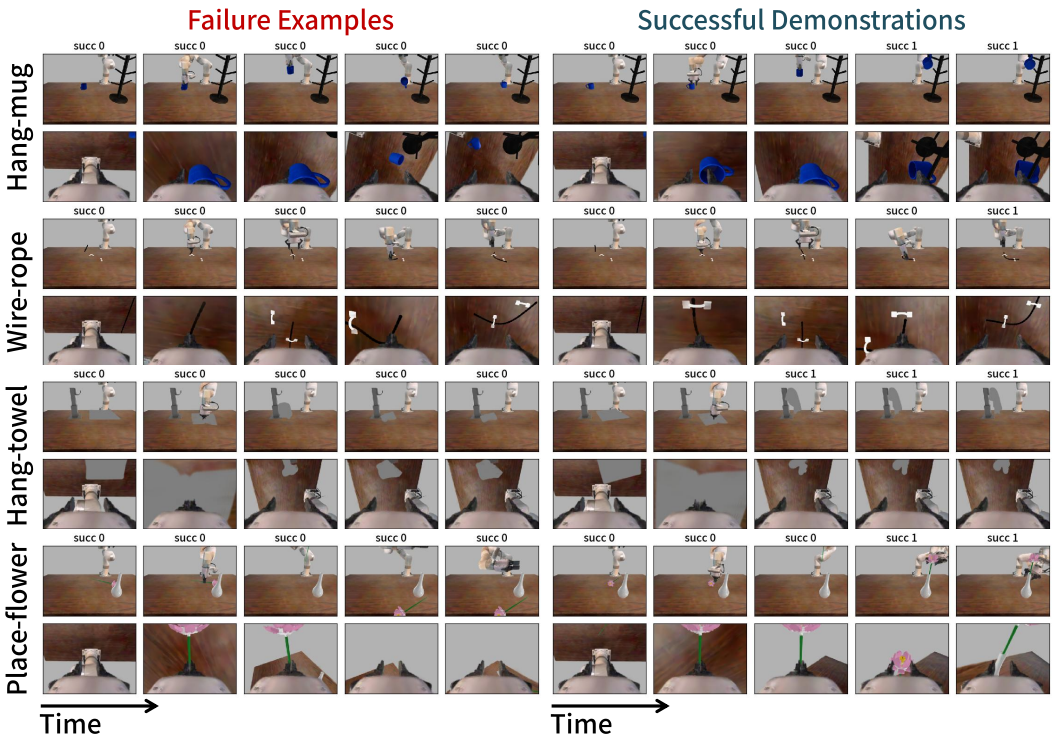


Figure 9: **Simulation data for the xArm workspace.** Example 3DGS renderings of failed trajectories (left) and successful demonstrations (right) for the four single-arm tasks. Each sequence shows synchronized scene and wrist-camera observations at five timestamps; *succ* indicates the privileged success predicate evaluated by the simulator.

B.5 Offline Rendering

We decouple rendering from physics by first collecting state trajectories and then rendering their observations offline, in batches. The reason is throughput: batched rendering is several times cheaper per frame than rendering one environment at a time. On a single NVIDIA L40S, the *Hang-mug* scene at 256^2 pixels or below renders at 77 frames per second with one environment and 385 with sixteen, so collecting state first and rendering afterwards converts a per-episode cost into a batched one. The buffers the critics are trained on are rendered at 320×240, and every simulated observation used to train the critic is rendered by the twin from the pose of a real deployment camera.

B.6 Critic Architecture

Each camera view is encoded into a 768-dimensional feature by the frozen DINOv2 backbone described in Section 4.3. We precompute these features and concatenate them across views, together with proprioception: 28 dimensions on the RM65-B and 15 on the xArm. Proprioception is dropped out with probability 0.5 during training. The head predicts a distribution over 101 bins spanning $[-0.05, 1.05]$ under a Gaussian-smoothed histogram loss; $V_\phi(o_t)$ is the expectation of this distribution. The architecture is shared between the sparse and dense variants.

B.7 Optimization and Model Selection

Training combines approximately 10^3 simulated episodes per task with N successful real demonstrations. Simulated frames receive the privileged success or progress targets defined in Section 3.1, whereas real demonstrations receive sparse success targets in both variants. Each step samples 8 episodes and 32 frames per episode, with relative real-to-simulated sampling weights of $1:2$, except on *Place-toy*, where the ratio is $1:20$. We train for 40 epochs with AdamW (Loshchilov & Hutter, 2019), using a learning rate of 3×10^{-4} and weight decay of 10^{-4}.

Task	Rollout pool		Rehearsal	Fine-tuning	Steps
	episodes	successful	episodes	frames	
Sort-blocks	218	76	72	173,617	3,000
Stack-blocks	152	35	72	266,361	4,200
Place-coin-and-ball	158	62	72	236,707	3,700
Place-toy	120	87	60	166,389	2,600
Hang-mug	153	53	60	142,324	2,250
Wire-rope	152	55	62	702,262	3,000
Hang-towel	158	75	60	219,644	3,450
Place-flower	150	63	62	325,435	5,100

Table 5: **Offline-RL pools and training length.** The *rollout pool* is the base policy's own real-robot rollouts, collected from multiple checkpoints of the same fine-tuning run; *successful* counts them by the operator's label, which only the Sparse Reward row reads. *Rehearsal episodes* is the fixed 30% slice of that task's teleoperated demonstrations, which rides at weight one and is never advantage-weighted. All rows of a task share this table and differ only in the reward that defines the return.

For each task and variant, we train with six random seeds and select the checkpoint with the lowest loss on a held-out simulated validation set containing approximately one fifth of the buffer's episodes. Selection uses no real-world episodes. For each task, both variants share the architecture, buffers, real demonstrations, optimization settings, and selection rule; only the targets on simulated frames differ.

C Reinforcement Learning Experiment Details

This section gives the full recipe behind the policy-improvement experiments of Section 5.3.

C.1 Offline RL

Procedure and data We study how the reward used to define returns affects what a generalist policy learns from its own real-world experience. For each task, we first fine-tune $\pi_{0.5}$ (Black et al., 2025a) on teleoperated demonstrations for $30{,}000$ steps and collect real-robot rollouts from multiple checkpoints of that run. Every offline-RL variant starts from the same last-step checkpoint and trains on the resulting pool, which contains 120–218 episodes per task. Table 5 gives detailed information. Each episode includes all camera views and proprioception recorded at 30 Hz. The teleoperated demonstrations are not used as RL experience; instead, a fixed 30% subset, sampled once and shared across all variants for a task, is included with unit weight for rehearsal. Only the **Sparse Reward** variant uses operator-provided outcome labels. The four model-reward variants are label-free throughout reward computation, value fitting, and sample weighting, and their cross-validation folds are stratified by rollout source rather than outcome.

Decision process and value learning Since the policy acts in chunks of 50 actions, we treat each chunk as one decision; a terminal chunk shorter than five frames is merged with its predecessor. The decision state x_k is constructed independently of the reward model from the chunk's first frame, and—unlike the privileged state s_t of Section 3.1—from observable quantities only: we concatenate the frozen DINOv2 features from all camera views with raw proprioception, yielding $3 \times 768 + 28 = 2{,}332$ dimensions on the RM65-B and $2 \times 768 + 15 = 1{,}551$ on the xArm. Features are z-scored within each fold using training-fold statistics only. For each task, we split whole episodes into five folds and fit V_ψ by expectile regression with $\tau = 0.7$ and three random seeds per fold. We retain only held-out predictions, ensuring that no decision is evaluated by a value function trained on its episode. Let $\bar{V}$ denote the mean prediction across seeds and K_e the final chunk index of an episode:

$$G_k = \sum_{j=k}^{K_e} \gamma^{j-k} r_j, \qquad A_k = G_k - \bar{V}(x_k), \qquad z_k = \frac{A_k}{\mathrm{sd}(A)}, \qquad w_k = \mathrm{clip}(e^{z_k}, 0, 20)\,, \quad (3)$$

where G_k is the discounted return-to-go from chunk k, A_k is its Monte Carlo advantage relative to the cross-fitted value baseline, z_k is the advantage standardized over the rollout pool, and w_k is the resulting clipped exponential training weight; $\gamma = 0.95$. The return-to-go captures the episode's

future rewards, while the learned value provides only a baseline. We assign each chunk weight to all frames in that chunk. Because each supervised frame predicts a 50-action horizon, its training weight is the mean of the corresponding chunk weights over that horizon. Finally, rollout-frame weights are normalized to have mean one, while rehearsal frames retain unit weight.

Reward specification All variants share the procedure above and differ only in the per-chunk reward r_k. **Sparse Reward** assigns the operator's binary outcome y_e to the final chunk and zero elsewhere: $r_k = y_e\,\mathbf{1}[k = K_e]$. The four model-reward variants use

$$r_k = c\,X(f_k) + \tilde{o}_e\,\mathbf{1}[k = K_e], \tag{4}$$

where f_k is the first frame of chunk k, $X(f_k)$ is the model's progress value, and o_e is its end-of-episode outcome score. Following Robometer's offline-RL setup (Liang et al., 2026), we use the absolute progress value as the dense reward and obtain RoboReward values through prefix queries (Liang et al., 2026); our additions are scale normalization and the terminal outcome term. We do not use progress differences: in Equation (3), a potential term $\gamma X(f_{k+1}) - X(f_k)$ telescopes to a terminal potential minus $X(f_k)$, whose state-dependent component is absorbed by the value baseline. The resulting advantages would therefore be nearly outcome-only, defeating the purpose of testing per-step progress feedback.

Absolute progress could in principle reward dwelling in a high-value intermediate state. However, dwelling also delays the discounted terminal reward, and the baseline subtracts the return expected from that state. Empirically, first-chunk advantage shows no systematic positive relationship with successful-episode length: $|\rho| \le 0.18$ on six tasks, $\rho = -0.47$ on *Wire-rope*, and $\rho = 0.46$ on *Place-toy*. We choose c so that the discounted dense return has unit standard deviation across episodes and set $\tilde{o}_e = o_e/\mathrm{sd}(o)$. Scaling without centering preserves the sign and zero point while placing progress and outcome on comparable scales. Although the critic predicts over bins spanning $[-0.05, 1.05]$, all outcome scores are nonnegative except for one negligible value of -5×10^{-5} on *Wire-rope*. Every pool has nonzero spread; the smallest outcome-score standard deviation among the 32 model-reward pools is 0.03. A zero-spread pool would be treated as reward-model failure rather than stabilized with an arbitrary floor. Neither normalization uses human labels.

For **w/ Sim2Critic (Sparse)** and **w/ Sim2Critic (Dense)**, X is the corresponding critic's raw per-frame value and o_e is the mean of its final two values, matching the outcome statistic in Section 5.1. For **w/ Robometer (Few-shot)**, X is the adapted model's progress score, forward-filled from its three-frames-per-second trace, and o_e is the mean success-head score over the final fifth of that trace. For **w/ RoboReward (Few-shot)**, we query each decision using the video prefix ending at the chunk's first frame, with the first chunk reusing the second chunk's readout; o_e is the full-video score. Both values are the normalized expected score $(\mathbb{E}[s] - 1)/4$ under RoboReward's five-level rubric. We leave its context limit unchanged: episodes exceeding the trained context are evaluated without truncation or subsampling.

Policy fine-tuning Each variant fine-tunes the task's base policy using its original flow-matching objective weighted by the per-frame values in Equation (3). The LeRobot training dataset concatenates the rollout pool with the shared rehearsal subset; all variants for a task use identical episodes and differ only in their weight files. We train with batch size 64 and a peak learning rate of 10^{-5}, equal to $0.40\times$ the peak rate used to train the base checkpoint. The learning rate warms up over the first tenth of training and then follows a cosine decay to 10^{-6}. We choose the number of optimization steps per task to provide approximately one pass over the data, yielding 1.00–1.11 passes for seven tasks. *Wire-rope* is the exception: its one-pass checkpoint produced visibly slower robot motion than the base policy, so we use the intermediate 3K-step checkpoint, whose step budget is comparable to those of the other tasks. We retain the base policy's normalization statistics. Full settings are reported in Table 6.

Evaluation We evaluate each fine-tuned policy on 25 real-robot attempts for its task. Object placements are matched as closely as possible across variants, and the policy identity is hidden from the operator during each evaluation round.

Hyperparameter	Value
Decision process	
Action chunk / decision length	50 actions
Minimum terminal chunk length	5 actions
Discount factor γ	0.95
Value-function state	frozen DINOv2 features of every view $\oplus$ proprioception
State dimension	2,332 (RM65-B) / 1,551 (xArm)
Value function	
Cross-fitting folds (by episode)	5
Seeds per fold	3 (predictions averaged)
Expectile τ	0.7
Batch size	256
Optimization steps	30,000
Learning rate (Adam)	3×10^{-4}
Sample weights	
Advantage standardization	unit standard deviation over the pool
Weight	$\mathrm{clip}(e^z, 0, 20)$, mean-normalized to one
Frame weight	mean over the 50-action horizon
Rehearsal fraction / weight	30% of the demonstrations / 1.0
Policy fine-tuning	
Batch size	64
Peak learning rate	10^{-5} ($0.40\times$ the base policy's peak)
Warmup	steps / 10
Schedule	cosine decay to 10^{-6} at the last step
Image augmentation	on

Table 6: **Offline-RL hyperparameters.** Identical for every row of every task; only the reward that defines the return in Equation (3) changes across rows.

C.2 Online RL

Platform and task We conduct the online RL experiments in the UFACTORY xArm 7 workspace on two tasks: *Wire-rope* and *Hang-towel*. For each task, the policy is a $\pi_{0.5}$ checkpoint fine-tuned on the same teleoperated demonstrations. It runs with 32 denoising steps at the robot's 30 Hz control rate and predicts a horizon of 50 actions. The first 25 actions form one execution chunk, and the next chunk is sampled when three actions remain. Each chunk corresponds to one RL decision, which begins when its first action is executed.

An episode terminates when the reward model's detector declares success and the operator does not overrule the declaration, or when a task-specific time limit is reached: 250 s for *Wire-rope* and 100 s for *Hang-towel* (7,500 and 3,000 control steps, respectively). These limits are chosen from the base policy's rollout-length distribution so that nearly all successful episodes finish within them.

The robot workstation is equipped with an NVIDIA RTX 5080 GPU. Although Sim2Critic is compact enough to run locally alongside the base $\pi_{0.5}$ policy, the two generalist reward models are not. For a common evaluation setting, we therefore run only the policy and RL networks on the robot workstation and serve all reward models—including our critic encoder and head—from a datacenter NVIDIA L40S GPU. The RL agent observes frozen DINOv2 features from the two 320×240 camera views, concatenated with z-scored proprioception and updated at 4 Hz. At the start of each decision, it stores the latest available observation. All four variants share this representation and frozen encoder, differing only in their reward signals.

The learning loop We apply DSRL (Wagenmaker et al., 2025), a latent-noise-space reinforcement-learning method, to the frozen policy. A soft actor–critic agent acts on all 32 latent noise dimensions of each action chunk, with every dimension bounded to 1.5. For chunk slots that are not executed, we repeat the final emitted action.

Each decision receives a base reward of -1, changed to 0 for the decision on which the detector declares success. We add the mean reward-model value, clamped to $[0, 1]$, over the frames executed by that decision. Frames on which the controller merely holds its previous command are excluded; if all frames are held, the decision receives only the base reward.

Hyperparameter	Value
Interface to the frozen policy	
Denoising steps	32
Plan length / slots kept per chunk	50 / 25 (re-planned when three remain)
Steered latent dimensions	all 32
Noise bound	1.5
Observation	DINOv2 features of two views $\oplus$ z-scored proprioception
Observation poll rate	4 Hz
Soft actor–critic	
Batch size	64
Discount per control step / per decision	0.995 / $0.995^{25} \approx 0.882$
Target update rate τ / interval	0.005 / 1
Critics / sampled per update	4 / 2
Actor updates per train step	10
Critic updates per actor update	4
Actor / critic / entropy-coefficient learning rate	5×10^{-5} / 10^{-4} / 10^{-4}
Target entropy	0
Encoder (actor and critic)	6-layer, 8-head, 256-wide transformer
Actor head	$[512, 512]$, $\tanh$, log-std range $[-20, 2]$
Critic head	layer norm, first-token pooling, $[768, 512]$
Replay capacity	50,000 decisions
Session (paired values are *Wire-rope* / *Hang-towel*)	
Seed session cap	40 episodes
Learning-off episodes per row	8
Learning episodes per row	32
Train steps per scene reset	25 (250 actor / 1,000 critic updates), learning block only
Episode time cap	7,500 / 3,000 control steps
Safety caps	12,000 decisions; 330,000 / 160,000 control steps
Termination	detector-declared success: terminal
	time cap: truncation, value bootstrapped
	operator may overrule a proposal; the episode resumes
	operator stop, unrecoverable scene: truncation, value bootstrapped
	emergency stop: episode discarded
Reward and success detector (paired values as above)	
Live detector poll	Sim2Critic (Sparse), Sim2Critic (Dense), Robometer: every observation poll (4 Hz)
	RoboReward: 0.5 Hz
Detector input	Sim2Critic (Sparse), Sim2Critic (Dense): all camera views
	Robometer, RoboReward: current scene frame
Reward of record	Sim2Critic (Sparse), Sim2Critic (Dense): every recorded frame
	Robometer: recorded video, 3 fps, $\leq$512 frames
	RoboReward: prefixes every 4th decision, 1 fps, $\leq$512 frames
Detector operating point	Sim2Critic (Sparse): 0.91 for 1 step / 0.34 for 45 steps
(threshold, consecutive control steps)	Sim2Critic (Dense): 0.82 for 15 steps / 0.67 for 120 steps
	Robometer: authors' rule (0.6, single poll) / success head above 0.27 for 8 polls (60 steps)
	RoboReward: top rubric score 5, single poll (both tasks)

Table 7: **Online-RL settings.** The soft actor–critic block follows the latent-noise-space recipe we compare against (Wagenmaker et al., 2025; Liang et al., 2026), converted to our decision unit of one action chunk. Every row shares these settings; only the reward server differs, and the two tasks differ only in the task-keyed rows.

Each transition is initially stored with the reward model's live output and later overwritten by a value recomputed from the complete recorded episode, which serves as the reward of record. For Sim2Critic (Sparse) and Sim2Critic (Dense), recomputation starts immediately after the episode and overlaps with the next episode. For the generalist models, it is requested after the episode, runs during the following episode, and is written back when that episode ends. Pending transitions remain available to the replay buffer with their live rewards. When a session resumes, any transition still lacking a reward of record is re-scored or discarded before training continues.

The discount is 0.995 per control step, corresponding to $0.995^{25} \approx 0.882$ per decision; it is a separate quantity from the critic-training discount γ of Equation (1), with which it shares only the symbol. Because decision durations vary slightly, we report their mean number of control ticks. Learning never occurs while the arm is moving: the agent performs a fixed number of updates at each scene reset during a learning block, and performs no updates during a learning-off block.

A sustained success detection pauses the arm and proposes episode termination. If the operator overrules the proposal, the episode resumes and further proposals are suppressed for 10 s. Otherwise, the episode terminates; the final decision receives base reward 0, and its value target is not bootstrapped. Reaching the task time limit or stopping because the scene is no longer recoverable—for example, after fixture damage—is instead treated as a truncation with a bootstrapped value target, and all collected decisions remain available for training. An emergency stop discards the entire episode: it is neither added to the replay buffer nor counted toward the session budget. A hardware fault only

pauses the arm, without issuing a gripper command, and the same episode resumes after recovery. Thus, the operator acts solely as a safety supervisor who may overrule the detector or stop an unsafe or unrecoverable run; no operator input supplies an outcome label or directly enters the reward or value target. Table 7 lists the agent settings.

Reward specification The four variants differ only in their reward server. **Sim2Critic (Sparse)** and **Sim2Critic (Dense)** use the latest critic value as both the provisional dense reward and the success-detector signal. At each control step, the detector proposes success when this value exceeds a task-specific threshold for a fixed number of consecutive steps. Table 7 reports these thresholds and hold durations. We select them on earlier operator-labeled rollouts passed through the same live observation path, maximizing recall subject to precision of at least 0.5; the 10 s refractory period after an overruled proposal limits excessive pauses. These labels are used only for calibration, not during online learning. Because the operating points are evaluated on the same rollouts used to select them, the reported detector statistics are in-sample.

Robometer (Few-shot) is queried at every 250 ms observation using the current scene-camera frame. Each query takes approximately 90 ms. Its progress output supplies the provisional transition reward, its success head generates proposals, and the complete episode is later re-scored to produce the reward of record. **RoboReward (Few-shot)** is queried at 0.5 Hz because its approximately 0.4 s latency exceeds the observation period. Its highest rubric level triggers a success proposal, while video-prefix queries every fourth decision produce the reward of record; the normalized score is held constant between queries.

We calibrate the generalist detectors on the same rollouts and with the same objective as ours. Each model receives the current scene frame through its released preprocessing, which resizes and duplicates the frame into the expected two-frame clip. This produces the Robometer operating point reported for *Hang-towel*. On *Wire-rope*, however, no threshold-and-hold rule for Robometer reaches the 0.5 precision requirement. RoboReward remains at its lowest greedy rubric level on nearly every calibration frame for both tasks, so it also provides no valid operating point. In these three cases, we retain the authors' released detection rule, which does not fire on the calibration rollouts. The remaining interface differences are that both generalist models use only the scene view and RoboReward operates at a lower polling rate.

Protocol Before running the four reward-model variants for a task, we collect a seed session with learning disabled, using the same interaction loop and termination rules for 40 episodes. Each reward model then relabels this shared replay buffer, ensuring that every variant starts from identical interaction data scored by its own reward. The seed session and all four variants use the same frozen policy checkpoint. Each variant subsequently runs 8 learning-off episodes under the same bounded random noise, providing its paired baseline for that day, followed by 32 learning-on episodes. Sessions end after these 40 episodes rather than at a step budget; the decision and control-step budgets in Table 7 serve only as safety limits. Including scene resets and reset-time updates, one session takes approximately 4 hours for *Wire-rope* and 2 hours for *Hang-towel*. We evaluate the frozen base policy with the four different learned SAC actors over 25 real-robot rollouts per task.

D Additional Results

D.1 Per-task reward evaluation

Rollout banks Each task's bank is the set of real robot rollouts of four generalist policies fine-tuned on that task's teleoperated demonstrations: π_0 (Black et al., 2025b), $\pi_{0.5}$ (Black et al., 2025a), SmolVLA (Shukor et al., 2025) and a diffusion-transformer policy (Hou et al., 2025), 30 to 44 episodes per policy. The operator labels every episode as a success or a failure, and that label is the only ground truth in Tables 1 and 8; the per-task episode and success counts are in the caption of Table 8. Task difficulty and policy quality vary widely across the banks—on Wire-rope one of the four policies produced no success at all (0 of 32 episodes)—so the columns are not comparable to one another, only the rows within a column are.

How each statistic is computed Each reward model converts an episode into a value trace containing one scalar per scored frame.

Reward model	Sort-blocks	Stack-blocks	Place-c.-b.	Place-toy	Hang-mug	Wire-rope	Hang-towel	Place-flower	Average
Reward alignment — VOC $\rho \uparrow$									
VLAC	—	—	—	—	**0.991**	0.327	0.931	0.849	0.775$^{(4)}$
SOLE-R1	0.149	0.416	0.465	0.751	0.819	**0.939**	0.926	0.663	0.641
RoboReward	0.892	0.832	0.832	0.938	0.953	0.480	0.959	0.879	0.846
Robometer	0.694	0.627	0.416	0.887	0.893	0.207	0.973	0.771	0.684
RoboReward (Few-shot)	0.945	0.818	0.895	0.884	0.942	0.832	0.963	**0.925**	0.901
Robometer (Few-shot)	0.768	0.633	0.756	0.880	0.938	0.648	0.956	0.769	0.794
Sim2Critic (Sparse)	0.945	0.922	**0.973**	**0.963**	0.895	0.867	0.941	0.904	0.926
Sim2Critic (Dense)	**0.952**	**0.923**	0.969	0.893	0.947	0.889	**0.989**	0.896	**0.932**
Trajectory ranking — Outcome AUROC $\uparrow$									
VLAC	0.500†	0.500†	0.500†	0.500†	0.819	0.532	0.640	0.619	0.576
SOLE-R1	0.305	0.329	0.373	0.554	0.643	0.570	0.831	0.729	0.542
RoboReward	0.853	0.876	0.531	0.896	0.797	0.494†	0.834	0.807	0.761
Robometer	0.924	0.827	0.599	0.904	0.897	0.694	0.887	0.969	0.838
RoboReward (Few-shot)	0.967	0.947	0.680	0.904	0.772	0.872	0.827	0.922	0.861
Robometer (Few-shot)	0.970	0.956	0.709	0.911	0.956	0.930	0.893	0.974	0.912
Sim2Critic (Sparse)	0.976	0.991	0.732	**0.919**	**0.990**	0.980	0.957	**0.979**	0.941
Sim2Critic (Dense)	**0.988**	**0.998**	**0.777**	0.879	0.987	**0.981**	**0.964**	0.969	**0.943**
Succ–Fail — $d \uparrow$									
VLAC	—	—	—	—	1.11	−0.04	0.10	0.22	0.35$^{(4)}$
SOLE-R1	−0.70	0.85	−0.26	0.57	0.71	0.31	1.35	0.63	0.43
RoboReward	1.95	1.82	0.39	1.60	1.30	−0.03	1.58	1.25	1.23
Robometer	1.59	1.01	0.17	1.78	1.12	0.74	1.99	1.69	1.26
RoboReward (Few-shot)	2.96	1.74	0.77	1.73	1.28	1.74	1.50	1.62	1.67
Robometer (Few-shot)	2.55	1.83	0.58	1.97	1.73	1.76	**2.09**	**2.29**	1.85
Sim2Critic (Sparse)	3.77	4.18	0.87	**2.07**	**1.88**	**3.33**	1.61	1.88	2.45
Sim2Critic (Dense)	**4.04**	**4.85**	**0.97**	2.02	1.79	3.08	1.72	1.56	**2.50**

Table 8: **Per-task reward evaluation.** All reward models score the same banks of real robot rollouts: Sort-blocks 134 episodes (41 successful), Stack-blocks 125 (12), Place-coin-and-ball 128 (29), Place-toy 131 (46), Hang-mug 136 (55), Wire-rope 129 (41), Hang-towel 138 (57), Place-flower 129 (24). "—" marks a cell that is undefined because the model returns the same value on every frame of every episode, so a within-episode rank correlation and a success–failure margin do not exist. † marks a cell the evaluation flags as degenerate: VLAC returns a constant value on every episode of the four RM65-B tasks, and on more than half of the *Wire-rope* episodes RoboReward returns a constant trace. In the *Average* column, $^{(4)}$ marks a mean taken over only the four tasks on which the statistic is defined; AUROC means are taken over all eight tasks, with the degenerate tasks entering at chance. Table 1 averages these values.

Reward alignment (VOC). For each successful episode, we compute the Spearman rank correlation between the raw value trace and frame index, following GVL (Ma et al., 2025); the task score is the median across successful episodes. Constant traces have undefined correlation and are omitted. VOC measures whether values increase monotonically during successful rollouts, but does not evaluate failures, state grounding, or responses to lost progress. These properties are instead examined by the outcome metrics and the qualitative traces in Figures 4 and 10. Our critic cannot directly read elapsed time because each observation is scored independently, without frame index or history. Coverage is therefore the number of successful episodes with nonconstant traces. It falls materially below the episode count only for the degenerate cases identified in Table 8: VLAC on the four RM65-B tasks and RoboReward on *Wire-rope*.

Trajectory ranking. We compute AUROC between one final score per episode and the operator's outcome label over all episodes of a task. For models with a success head, the final score is the head's output over the episode's closing window; for all other models, including ours, it is the mean of the last two raw values.

Success–failure separation. We smooth each trace, average its final fifth, and compute the difference between the successful and failed means divided by their pooled standard deviation. Unlike Robometer's unnormalized gap, this standardized Succ–Fail d is comparable across reward scales. A model that outputs the same value for every frame and episode has undefined reward alignment, zero separation, and chance-level trajectory ranking; these cases are marked in Table 8.

Generalist rows VLAC is the 8B checkpoint on every column. SOLE-R1 is run per task in its configuration of record: both camera views on the xArm tasks (Hang-mug, Wire-rope, Hang-towel, Place-flower) and the single scene view on the RM65-B tasks. RoboReward and Robometer are

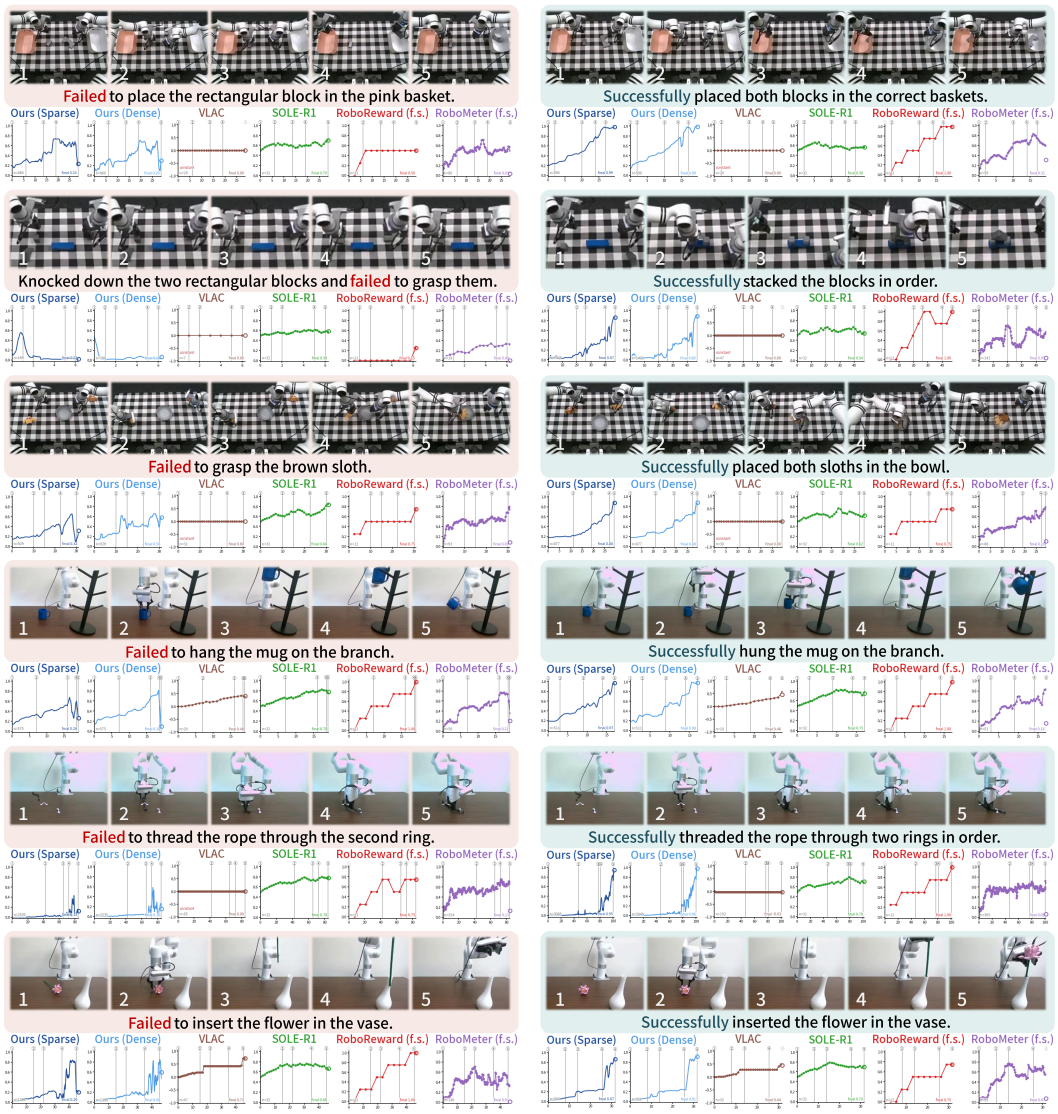


Figure 10: **Additional qualitative reward traces on the remaining six tasks.** Each task shows one failed rollout (left) and one successful rollout (right), with five representative observations above the value traces from Sim2Critic (Sparse), Sim2Critic (Dense), and the generalist baselines. Our critics generally increase toward successful completion and remain low or decrease after failure, while the generalist models often capture either progress or outcome without consistently capturing both.

used zero-shot at the checkpoints their authors release. Robometer (Few-shot) is Robometer adapted per task for 100 optimization steps on the same real demonstrations the critic of record for that task receives, with everything else in the released recipe left at its default; its success head is then read exactly as the zero-shot model's is. RoboReward (Few-shot) is the same idea applied to RoboReward: a rank-16 LoRA on the language layers alone, the vision tower frozen, trained for 100 optimization steps on those same demonstrations. RoboReward is trained to score a video against a five-level rubric rather than to regress progress, so the adaptation targets are synthesized from the demonstrations themselves the way its own training data is: the full demonstration takes the top score; prefixes of it cut at 25%, 50% and 80% of its length take the three intermediate scores in that order; and the demonstration paired with each of two other tasks' instructions takes the failure score. Those cut points are a fixed proxy for progress and are not validated against any annotation, and no outcome label beyond the demonstrations enters. Its success head is read exactly as the zero-shot model's is.

Reward model	Parameters		Inference				Setting
	total	per task	ms / value	values / ep.	values / frame	values / s	
VLAC	8.075B	0.00M	93.01	51.3	0.034	1.02	bf16, flash attn.
SOLE-R1	8.767B	0.00M	576.40	32.0	0.035	1.05	batch 8, one pass
RoboReward	8.767B	0.00M	454.79	12.0	0.013	0.39	bf16, 13 gen.
RoboReward (Few-shot)	8.811B	43.65M	617.04	12.0	0.013	0.39	bf16 + fp32 LoRA, 13 gen.
Robometer	4.447B	0.00M	49.86	105.6	0.089	2.66	bf16
Robometer (Few-shot)	4.513B	75.92M	53.97	126.3	0.096	2.89	fp32
Sim2Critic	87.17–87.43M	0.59–0.85M	7.81	1,524.3	1.000	30.00	fp32, end to end
w/ cached features			0.0024				fp32, head only

Table 9: **Parameters and inference latency.** Every row is timed on one NVIDIA L40S over the same fixed set of 64 real episodes, in the configuration and the precision that produced its Table 1 scores; each cell is the median of three timing passes, except SOLE-R1's, which is a single pass. *Total* counts the parameters a scoring call instantiates and *per task* those trained per task. The two Sim2Critic variants share one architecture, so one row covers both, and its parameter range spans the two-camera and the three-camera head. *values / frame* is how many values a model emits per recorded camera frame, *values / s* how many per second of episode time. The row with "Few-shot" is its zero-shot parent with a LoRA attached, and attaching it costs 1.36× the parent's time per emitted value for RoboReward and 1.08× for Robometer. Section D.2 gives the protocol.

Additional qualitative results. Figure 10 extends the qualitative comparison of Figure 4 to the remaining six tasks. Across tasks, both Sim2Critic variants generally rise with successful execution and terminate near one, while remaining low or decreasing after failures such as knocked-down blocks, a missed mug hang, incomplete rope threading, or a missed flower insertion. The dense critic additionally reflects partial progress in several failed rollouts, whereas the sparse critic more sharply separates terminal outcomes. The generalist baselines exhibit the same trade-off observed in the main text: RoboReward (Few-shot) often tracks stage transitions but can assign high terminal values to failures, while Robometer (Few-shot) produces plausible intermediate traces yet frequently gives similar or low terminal scores for both outcomes. VLAC is constant on the four RM65-B tasks, and SOLE-R1 often remains high on failures. These results illustrate how our critics combine progress sensitivity with outcome discrimination across both robot platforms and diverse manipulation tasks.

D.2 Parameters and inference latency

Timing protocol We time every row of Table 9 on one NVIDIA L40S with 46 GB of memory and a single model loaded. All rows use the same fixed set of 64 real episodes—eight per task, evenly split between successes and failures—sampled before timing and used for no other purpose. Each model runs in the same configuration and precision used to produce its results in Tables 1 and 8. We report the median of three timing passes, except for SOLE-R1, which is measured once at its evaluation batch size.

Total parameters counts everything instantiated for one scoring call. The Robometer rows instantiate slightly fewer parameters than their checkpoint files contain because an unused head is not constructed. *Per-task parameters* counts only parameters trained for an individual task: zero for zero-shot models, one adapter for each few-shot generalist, and the lightweight critic head for Sim2Critic. Most of Sim2Critic's total parameter count belongs to its frozen backbone; the reported range reflects the two- and three-camera heads.

ms / value is the wall-clock time to score an episode divided by the number of values emitted. The remaining columns express the same output density as values per episode, per recorded camera frame, and per second of episode time. These densities differ because each model follows its released scoring recipe. Our critic emits one value for every frame at the 30 Hz camera rate. Robometer scores at three frames per second, up to 512 frames per episode; on 15 of the 64 episodes, its out-of-memory fallback reduced this limit further.

RoboReward is measured in *generations* rather than forward passes because it returns rubric scores as decoded text. We run thirteen generations per episode: one on the full video for its native end-state score and twelve on prefixes ending at successive twelfths of the episode. Videos are sampled at approximately one frame per second, up to 512 frames. The twelve prefix scores form the value trace counted in the table.

Method		Sort-blocks	Stack-blocks	Place-c.-b.	Place-toy	Hang-mug	Wire-rope	Hang-towel	Place-flower	Average
Base – $\pi_{0.5}$		40%	12%	20%	48%	32%	36%	44%	24%	32.0%
Co-training		48%	16%	20%	48%	40%	36%	36%	32%	34.5%
Offline RL	Sparse Reward	44%	12%	16%	56%	28%	40%	48%	20%	33.0%
	w/ RoboReward (F.s.)	52%	12%	20%	52%	28%	40%	48%	24%	34.5%
	w/ Robometer (F.s.)	48%	12%	20%	56%	40%	40%	56%	28%	37.5%
	w/ Sim2Critic (Sparse)	**68%**	20%	24%	**68%**	40%	48%	60%	**40%**	46.0%
	w/ Sim2Critic (Dense)	64%	**28%**	**28%**	**68%**	**52%**	**52%**	**64%**	36%	**49.0%**

Table 10: **Policy learning with offline RL.** Success rate on the real robot over 25 attempts per task. The five offline-RL rows share the pipeline described above and differ only in the reward models.

For precision, zero-shot Robometer uses the half-precision path selected by its released runner. A paired comparison on the same GPU found this path no faster than full precision, but we report the configuration that produced its evaluation scores. Adapted Robometer is evaluated directly in full precision. Adapted RoboReward uses an unmerged FP32 adapter attached to the half-precision base and the same thirteen-generation scoring procedure as its zero-shot parent; thus, the adapter is the only difference between the two RoboReward rows. Finally, the cached-feature row reports only the cost of the critic head when backbone features are already available. It is included as a disclosure and is not used in place of the end-to-end row in any claim in Section 5.1.

Analysis Table 9 shows that Sim2Critic is substantially smaller and faster than the generalist reward models while producing much denser feedback. Its approximately 87M parameters are over $50\times$ fewer than Robometer, the smallest generalist baseline, and its 7.81 ms end-to-end latency is about $6.4\times$ lower than the fastest generalist row. Moreover, Sim2Critic emits values at the full 30 Hz camera rate, more than $10\times$ the output rate of the densest generalist model. Its per-task trainable head contains only 0.59–0.85M parameters, at least $51\times$ fewer than the adapters used by the few-shot generalists. When backbone features are already available, head-only inference takes 0.0024 ms per value, indicating that nearly all end-to-end cost comes from the shared visual encoder rather than the task-specific critic.

D.3 Per-task offline RL results

As shown in Table 10, both Sim2Critic variants improve over the base policy on every task, but their relative advantage depends on task structure. Sim2Critic (Dense) performs better on several multi-stage tasks, like *Stack-blocks*, *Place-coin-and-ball*, and *Wire-rope*. The difference is largest on *Stack-blocks*, where success requires placing multiple blocks in the correct order: dense milestones can assign credit to each valid intermediate placement, providing a more informative learning signal long before final success. More generally, progress supervision may improve credit assignment when a rollout contains several ordered subgoals or when partial completion remains useful despite eventual failure. In contrast, Sim2Critic (Sparse) performs better on *Sort-blocks* and *Place-flower*, which are closer to direct pick-and-place or insertion tasks with clear terminal geometry. In these settings, the discounted success target may provide a sharper distinction between correct and incorrect final placements, while avoiding bias or noise from hand-designed intermediate milestones. Despite this task dependence, at least one Sim2Critic variant outperforms the strongest baseline on every task, and both improve over the base across all eight tasks. Overall, these results demonstrate Sim2Critic's potential to enhance a generalist policy's performance across various manipulation tasks.

E Future Directions

Several directions could further improve the scalability and capability of Sim2Critic. First, twin construction could be streamlined by combining automated reconstruction with an agentic pipeline for workspace and object capture, asset isolation, metric registration, calibration, and physics binding, leaving only uncertain cases for human review. Second, task-conditioned video generation could provide synthetic adaptation demonstrations and reduce reliance on real data. In our current setting, the roughly ten successful demonstrations used by each critic add little overhead because they can be drawn from the hundreds of teleoperated demonstrations already collected to fine-tune the base policy; generated videos could extend the approach to settings where such data are unavailable. Third, future critics could incorporate temporal context more effectively. The fixed-window models evaluated in Section 5.2 did not consistently improve performance, potentially because they were sensitive

to simulation-specific event timing and execution speed. Event-aligned recurrent or state-space models trained with variable-speed trajectories may better retain evidence of contacts, releases, and deformable-object dynamics while remaining robust to differences in rollout pacing.