Value Sim-to-Real: Simulation Values for Efficient Real-World RL

Weijun Dong[1,*] **Yilin Wang**[1,*] **Jiawei Fu**[1] **Yuchen Zhou**[2]
Henrik I. Christensen[1] **Zhao-Heng Yin**[3,†] **Hao Su**[4,†]
[1] University of California, San Diego [2] University of Pennsylvania
[3] University of California, Berkeley [4] Sudo AI GmbH

Abstract: Real-world reinforcement learning (RL) is a promising approach for advancing robotic manipulation, but how to obtain informative and robust reward signals remains a central challenge. Recent works have explored reward models trained on real robotic data to enable efficient real-world RL. However, this formulation can be expensive and infeasible due to the high-dimensional state space and its dependence on real-world expert data. In this paper, we address this limitation with Value Sim-to-Real (VSR), a novel framework that leverages simulation-trained value functions to define informative, high-quality reward functions for efficient real-world RL. We evaluate our method with both sim-to-sim and sim-to-real experiments on several vision-based contact-rich tasks. We show that VSR learns a smooth value function that strongly correlates with task progress and improves the performance and sample efficiency of downstream RL with its induced reward function.

Keywords: Robot Manipulation, Sim-to-Real, Reinforcement Learning

1 Introduction

Real-world Reinforcement Learning (RL) allows robots to master complex manipulation skills on their own, bypassing the limitations of imperfect human demonstrations [1, 2, 3, 4, 5]. This approach optimizes policies through real-world trial and error by maximizing cumulative task rewards. However, designing these reward functions remains a major bottleneck. Manually defining rewards is challenging, often requiring complex systems engineering (such as detecting precise object states) and risking unintended reward-hacking behaviors.

To address this challenge, recent works have explored data-driven reward models trained on real-world robotics datasets [6, 7, 8, 9, 10, 11]. The core paradigm relies on collecting extensive expert robot trajectories or video demonstrations to learn a *progress-based model* that outputs task-progress signals, which subsequently serve as the reward signal. Nevertheless, this paradigm faces fundamental limitations. First, robot manipulation tasks have large state spaces that span numerous combinations of object poses. For N objects in the state space, there are at least $6N$ independent state dimensions, and real-world data collection for reward training can only cover a very low-dimensional subset. Due to this, a progress-based reward model trained on real-world data often struggles to generalize to unseen object states, where it may produce noisy or even erroneous guidance. Moreover, some critical states (e.g., stuck objects) are difficult to deliberately capture during real-world data collection, further exacerbating this issue. Second, training a reliable reward model assumes access to high-quality trajectory supervision. This requires near-optimal demonstrations or careful post-hoc manual progress labeling, an assumption that is often expensive and impractical. Without such careful curation, the progress signal can also become noisy or suboptimal, which harms the final policy learning outcome as a result.

[*] Equal Contribution [†] Equal Advising

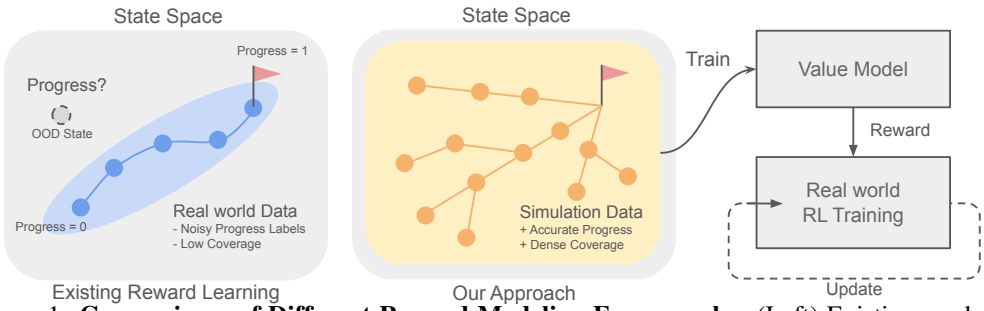


Figure 1: **Comparisons of Different Reward Modeling Frameworks**. (Left) Existing works directly collect real world videos or robot trajectories for learning a progress reward. However, this suffers from several issues including noisy progress labeling and low state space coverage, and the OOD critical states can not be guided effectively. (Right) Therefore, we propose to learn reward function from simulation. This formulation overcomes the drawback of prior methods.

Inspired by recent progress in scalable sim-to-real learning [12, 13], we propose Value Sim-to-Real (VSR), a framework that transfers simulation-trained value functions to provide dense reward guidance for real-world RL. Since all value-model training data are collected in simulation, we obtain substantially broader state coverage through iterative state expansion, mitigating data challenges arising from high dimensionality and rare critical states. Furthermore, by rolling out well-trained simulation policies, we obtain automatically computed, fine-grained progress supervision, avoiding the noise and ambiguity of manually annotated progress labels in real-world datasets. Therefore, this simulation-based framework can help to overcome the limitations of existing reward models.

A key challenge in implementing this framework is exhaustively searching over task-relevant states and determining their associated progress or value. We address this by developing a systematic data generation pipeline that iteratively expands a set of feasible states while optimizing the simulation policy on the evolving state distribution. Using the expanded states and trained policy, we construct a value-learning dataset and distill it into a point-cloud value function for transfer to the real world. Across three real-world tasks, VSR outperforms real-world RL baselines using alternative rewards by at least 14.0 percentage points on average. Both quantitative and qualitative evaluations further show that the transferred value function reliably reflects real-world task progress. In sim-to-sim experiments, VSR also achieves the highest average final success rate across the three tasks.

2 Related Work

Learning Reward Models. Early attempts to learn rather than hand-design rewards center on inverse RL and generative adversarial imitation [14, 15, 16, 17]. However, their recovered rewards are often just non-stationary byproducts of distribution matching and cannot be reused for downstream RL. More recently, large vision-language models have made it possible to elicit rewards zero-shot from pretrained models without task-specific data [18, 19, 20], yet these signals are weakly grounded in physical dynamics and often too coarse or noisy for precise manipulation. The most prevalent paradigm trains progress- or value-based reward models on large-scale real-robot and human trajectories, yielding dense, task-grounded rewards for real-world RL [6, 7, 8, 9, 10, 11]. However, real-world data covers only a low-dimensional slice of the manipulation state space and rarely includes critical states such as stuck objects, so these models generalize poorly off-distribution. They also presuppose near-optimal demonstrations or careful progress labeling. Although recent methods reduce this dependence via trajectory preferences or in-context transfer [9, 10], the resulting supervision remains noisy and the quality bottleneck persists [21]. In contrast, VSR sources its reward from simulation, which provides both broad state coverage and precise, fine-grained progress supervision, thereby alleviating both bottlenecks at once.

Sim-to-Real for Robotic Manipulation. Sim-to-real transfer leverages large-scale, low-cost simulation to acquire knowledge for real-world deployment, and existing methods differ mainly in what they transfer. The most common target is the policy itself, typically transferred via domain and dynamics randomization [22, 23], which has enabled tasks ranging from generalizable grasping and

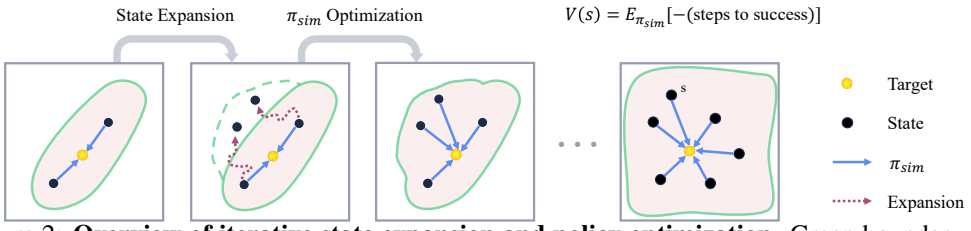


Figure 2: **Overview of iterative state expansion and policy optimization.** Green boundary indicates the current coverage of the state buffer. Starting from the initial reset range, simulation policy π_{sim} explores novel states by noisy rollouts. Accepted states are added to the state buffer and used to further optimize π_{sim}, forming an iterative loop between state expansion and policy improvement.

pick-and-place across diverse objects and embodiments [12, 24, 25, 26] to contact-rich dexterous manipulation [27, 28, 29, 30, 31]. As direct policy transfer is brittle under unmodeled contacts and dynamics mismatch, another line transfers auxiliary structure to bootstrap real-world learning, including exploration priors [32, 33], latent action spaces [34], and world models [35].

Our work is most closely related to SGFT [36] and SimDist [37], which likewise transfer a value learned in simulation to guide real-world learning. VSR differs from them in two main ways. (1) SGFT trains its value on the task's default state distribution and SimDist diversifies its data by perturbing actions along policy rollouts. In contrast, VSR starts from the task's initial states and iteratively expands the set of states it trains on, co-training a proficient policy over them, so the value stays reliable across the broad range of states a real-world policy might reach. (2) SimDist's value is built on a perception encoder that must bridge the visual appearance gap and SGFT's is built on a low-dimensional state that requires real-world state estimation, whereas VSR defines the value over segmentation-labeled point clouds, a geometry-centric representation that sidesteps both.

Real World RL. Reinforcement learning lets robots surpass demonstration performance, discovering faster and more reliable behaviors through real-robot practice. A first line studies the RL algorithm itself [38, 4, 39] or how to fine-tune pretrained policies through online RL, including offline-RL policies [40] and large-scale VLAs [41, 42, 3, 5]. Another thread introduces a human-in-the-loop setting, where operator interventions during autonomous execution make real-world training safe and sample-efficient [1, 2]. From another angle, RLFP focuses on the source of guiding priors rather than the RL mechanism itself, deriving policy, value, and reward priors from pretrained foundation models [43]. We share this perspective but derive value signals from simulation, which offers broader state coverage and more task-grounded guidance than internet-pretrained priors.

3 Method

Learning a reliable value model for real-world reward guidance requires broad coverage of task-relevant states and informative value supervision. VSR first alternates state expansion with policy optimization to build a diverse state buffer and train a simulation policy over this expanding distribution (Sec. 3.1.1). We then refine the policy under a sparse task-completion objective and use its rollout returns to supervise a vision-based value model (Sec. 3.1.2). Finally, we transfer the learned value model to shape rewardsfor real-world RL (Sec. 3.2).

3.1 Learning Value Model in Simulation

Problem formulation. We formulate robotic manipulation as a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \mathcal{G}, \mathcal{S}_0)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, and $P(s'|s, a)$ the transition dynamics. The reward function r provides the objective for RL policy training in simulation, and $\gamma \in [0, 1)$ is the discount factor. Let $\mathcal{G} \subseteq \mathcal{S}$ denote the set of task-success states, and $\mathcal{S}_0 \subseteq \mathcal{S}$ the set of initial states constructed to mirror the configurations from which the real-world task might start. In simulation, $\mathcal{S}$ corresponds to privileged simulator states, which are fully observable and serve as the basis for both our state expansion and the input to the simulation policy.

3.1.1 Iterative State Expansion and Policy Optimization

Overview. We maintain a feasible-state buffer $\mathcal{B}$, initialized from $\mathcal{S}0$, and a policy πsim trained from states in the buffer. The procedure proceeds in repeated expand-then-optimize cycles, as shown in Fig. 2. At cycle i, we first use the current policy $\pi_{\text{sim}}^{(i)}$ to explore from states in $\mathcal{B}^{(i)}$ and add accepted candidate states to obtain an expanded buffer $\mathcal{B}^{(i+1)}$. We then update the policy with PPO using initial states partially from $\mathcal{B}^{(i+1)}$, producing $\pi_{\text{sim}}^{(i+1)}$. Details are shown in Appendix Algorithm 1.

Expanding the buffer. As generating an arbitrary yet valid simulation state is hard, our idea is to explore novel states by starting from states already in $\mathcal{B}$. Concretely, we sample a seed state s_{seed} from $\mathcal{B}$ and, starting from it, roll out π_{sim} with added Gaussian noise $\mathcal{N}(0, 1)$ to each action dimension for a random number of steps; we take the resulting final state as a candidate s_{cand}. To avoid accumulating near-duplicates, we add s_{cand} to $\mathcal{B}$ only when its k-nearest-neighbor distance to existing states based on object and end-effector poses in the privileged state space exceeds a threshold. We also discard candidates with task objects outside a predefined 3D workspace whose horizontal bounds approximately match the reachable range of the robot end effector, preventing the buffer from being dominated by extreme states from which task completion is unlikely. Additionally, to prevent expanding from states that are too hard or unrealistic for the current policy, we require that π_{sim} has a non-zero success rate from s_{seed}.

Optimizing the policy. We train π_{sim} with PPO [44]. To facilitate policy learning during state expansion, we use $r = r_{\text{dense}}$, a task-specific dense reward function consisting of stage-dependent terms (e.g., reaching, goal-directed movement, and task completion) with pose/contact penalties.

Instead of sampling initial states uniformly from the entire buffer, we use a balanced reset distribution. Let $\mathcal{B}^{\text{new}}$ denote the states added in the most recent expansion round, and let $\mathcal{B}^{\text{old}} = \mathcal{B} \setminus \mathcal{B}^{\text{new}}$ denote the previously collected states. During PPO training, half of the rollouts are initialized from $\mathcal{B}^{\text{new}}$ and the other half from $\mathcal{B}^{\text{old}}$. In other words, the initial state distribution is defined as: $\rho_{\mathcal{B}} = \frac{1}{2}\text{Unif}(\mathcal{B}^{\text{new}}) + \frac{1}{2}\text{Unif}(\mathcal{B}^{\text{old}})$. The policy is optimized to maximize the expected discounted return under this reset distribution:

$$J(\pi_{\text{sim}}) = \mathbb{E}_{s_0 \sim \rho_{\mathcal{B}},\, \pi_{\text{sim}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{\text{dense}}(s_t, a_t) \right]$$

This balanced sampling emphasizes adaptation to newly discovered states while maintaining training coverage of previously collected states.

The two steps reinforce each other. The progressively expanding $\mathcal{B}$ acts as a natural curriculum for π_{sim}: rather than confronting RL with the full range of states at once, π_{sim} is trained on a gradually widening state distribution, which facilitates policy learning [45]. The balanced reset sampling further emphasizes newly added states while retaining coverage of older states, which stabilizes policy optimization. Conversely, π_{sim} biases state expansion toward task-relevant regions of the state space. Through this loop, $\mathcal{B}$ grows broadly while π_{sim} becomes a strong rollout policy for value labeling in the next stage. Details are provided in the Appendix Sec. A.1.

3.1.2 Value Labeling and Distillation

To formalize the value labels, we consider an undiscounted shortest-path MDP with the same state space, action spaces and transition dynamics defined above, and with goal set $\mathcal{G}$. Its reward r_{sparse} is -1 at every step, and the episode terminates upon reaching the success set $\mathcal{G}$. The optimal value $V^*(s)$ hence measures the negative expected number of transitions before completion under an optimal policy. We aim to approximate $V^*(s)$ with our value network.

To this end, we further fine-tune π_{sim} with PPO using r_{sparse} and $\gamma = 1$, initializing episodes from the expanded state buffer $\mathcal{B}$. We then roll out the refined policy from each $s \in \mathcal{B}$ and use the accumulated undiscounted reward as its label $y(s)$. These returns estimate $V^{\pi_{\text{sim}}}(s)$, which serves as a practical approximation to $V^*(s)$. Failed rollouts within horizon H are clipped to $y(s) = -H$.

Then, we distill these values into a vision-based value function V_ψ. Denoting the corresponding observation of s as o_s, we train V_ψ to predict $y(s)$ from o_s. To reduce the visual sim-to-real gap, o_s is represented as a point cloud augmented with per-point object segmentation labels, obtained from a segmentation tracker [46]. Rather than regressing a scalar value directly, we adopt distributional value prediction [47]: the target range is partitioned into K uniform bins, the network outputs a categorical distribution over them and is trained by minimizing the cross-entropy loss. The scalar estimate is taken as the expectation over the bin centers. The resulting V_ψ transfers to the real world, where it guides downstream RL (Section 3.2). Details are provided in the Appendix Sec. A.2.

3.2 Real-world Reinforcement Learning

In the real world, we assume only a sparse binary task signal, provided in practice by a human operator: every step receives $r_{\text{binary}} = -1$, and the episode terminates when the operator marks success. Used alone, this signal makes credit assignment difficult. Similar to the additive reward formulation used in Robometer [10], we augment the task reward with the learned value function V_ψ: at each step, the shaped reward is $r_{\text{real}}(o) = V_\psi(o) + \alpha\, r_{\text{binary}}$, where $V_\psi(o)$ is normalized to $[-1, 0]$, and α is a trade-off coefficient.

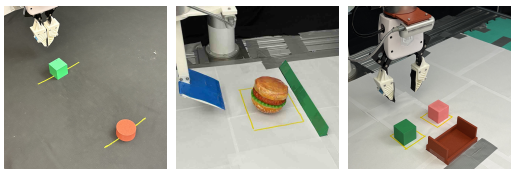
We instantiate online RL with DSRL [39], which keeps a pretrained base diffusion policy frozen and trains a steering policy to bias its sampling. The real-world policies take RGB images and the robot's proprioception as input, while V_ψ provides the value from a separate point-cloud observation. Implementation details are provided in the Appendix Sec. A.3.

4 Experiments

We evaluate the effectiveness and efficiency of VSR on three manipulation tasks. Following the experimental setup in Section 4.1, we present real-world and sim-to-sim RL results in Section 4.2 and assess real-world value transfer quantitatively and qualitatively in Section 4.3. Finally, Section 4.4 examines the coverage and characteristics of the expanded states and ablates key components of state expansion to assess their impact on downstream RL performance.

4.1 Experimental Setup

We use a 7-dof xArm7 robot with a parallel-jaw gripper. The robot is controlled by a differential IK controller, while the action for gripper is the absolute target position. Our perception system consists of three Intel RealSense D435 cameras: two fixed workspace cameras mounted on the table and one wrist-mounted camera on the arm.



(a) Push Cube (b) Hamburger (c) Pack Cubes

Figure 3: Illustration of three real-world tasks.

We evaluate our method and the baselines on three manipulation tasks in both sim-to-real and sim-to-sim transfer settings. The real-world tasks are shown in Fig. 3 (additional details in the Appendix Sec. B), and the simulation environments are built in SAPIEN [48]. For real-world experiments, the DSRL base diffusion policy is pretrained on 15, 10, and 30 demonstrations for Push Cube, Flip Hamburger, and Pack Cubes, respectively, before further improvement via online RL.

Task 1: Push Cube. The robot must push a cube to a target cylinder using the gripper. The cube and the target cylinder are randomized along two parallel $15\ \text{cm}$ lines separated by $20\ \text{cm}$. In the sim-to-sim setting, we calibrated the frictions in the environment to mimic sim-to-real gap.

Task 2: Flip Hamburger. The robot must use the shovel to push an initially upside-down hamburger toward a triangular wall, flip it over the wall, and make it land upright. The hamburger is randomly initialized within a $15\ \text{cm} \times 20\ \text{cm}$ rectangular region. In the sim-to-sim setting, we adjust the height of the wall to imitate sim-to-real gap.

Table 1: Evaluation results on 3 real-world tasks. All real-world RL methods use 150 online trajectories for Push Cube and Flip Hamburger, and 120 for Pack Cubes. We report *success rate* (SR) and *average completion time over successful trials* (CT, in seconds); for Pack Cubes, we additionally report the *phase-1 success rate* ($\text{SR}_{\text{Phase 1}}$). We evaluate the last two checkpoints, each with 25 trials.

Task	**Push Cube**		**Flip Hamburger**		**Pack Cubes**		
	SR (↑)	CT (↓)	SR (↑)	CT (↓)	SR (↑)	$\text{SR}_{\text{Phase 1}}$ (↑)	CT (↓)
Sim-to-real policy	24%	14.4	48%	6.2	0%	0%	-
Base diffusion policy	32%	16.8	48%	7.6	36%	84%	16.4
DSRL + Binary rewards	46%	13.3	66%	5.4	74%	98%	13.9
DSRL + Robometer	64%	14.3	68%	5.6	70%	**100%**	**13.7**
DSRL + Ours	**86%**	**13.0**	**74%**	**5.1**	**84%**	**100%**	14.2

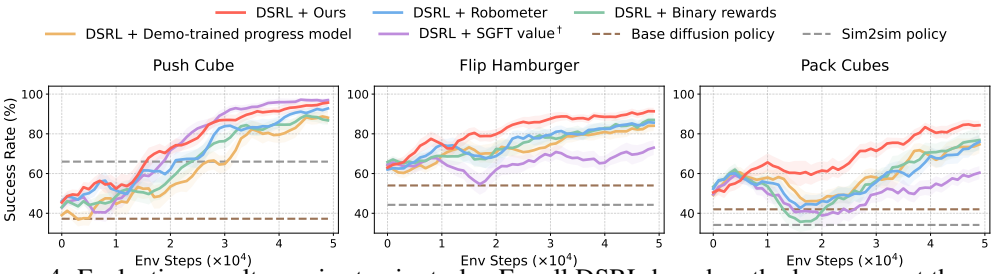


Figure 4: Evaluation results on sim-to-sim tasks. For all DSRL-based methods, we report the mean and standard error over eight random seeds. †DSRL + SGFT value is not strictly comparable to VSR, as its value function receives privileged target-simulator states unavailable to VSR.

Task 3: Pack Cubes. The robot must pick up two 4 cm cubes and place them into a 10 cm-wide container, with the pink cube on the right and the green cube on the left. Each cube is randomly initialized within a 6 cm $\times$ 6 cm region. For sim-to-sim transfer, we reduce the target controller's stiffness and damping to create a dynamics gap.

Baselines and Metrics. We compare against: (1) A direct sim-to-real policy trained on the same trajectories and using the same backbone as our value model, with its categorical value head replaced by an MLP action head; (2) The base diffusion policy trained with provided target-domain demonstrations. (3) DSRL with binary or off-the-shelf Robometer rewards. For sim-to-sim, we additionally include: (4) DSRL + Demo-trained Progress Model, whose progress model shares our value-model architecture and is trained on target-simulator demonstrations; and (5) DSRL + SGFT Value, which replaces our value function with one trained by SGFT [36] in the source simulator. Since SGFT's codebase does not support point-cloud inputs, its value function instead takes privileged states from the target simulator as input.

For real-world experiments, we report success rate (SR) and task completion time (CT). For sim-to-sim experiments, we report success-rate curves over $50{,}000$ environment steps.

4.2 Real-World and Sim-to-Sim RL Results

As shown in Tab. 1, with 120-150 online trajectories per task, VSR improves the average success rate across the three tasks by 42.7 percentage points over the base diffusion policy and reduces successful-trial completion time by 23.0% on average. It achieves the highest success rate on all tasks and outperforms each DSRL baseline by at least 14.0 percentage points on average. Despite substantial sim-to-real discrepancies such as the gripper and shovel (soft in real-world and rigid in simulation), and object friction differs between the two domains, VSR achieves strong real-world performance, whereas direct policy transfer remains less successful. Appendix Sec. C.2 shows the curve for real-world training.

The sim-to-sim results in Fig. 4 show that VSR maintains the highest success rates for most of training on Flip Hamburger and Pack Cubes, indicating strong sample efficiency, while remaining com-

Table 2: Quantifying value functions on real-world data. VOC is reported as mean$_{\pm \text{standard deviation}}$ across demonstration trajectories; terminal-outcome ranking accuracy is reported as a percentage.

	VOC on demos (Pearson r) ↑			Pairwise ranking acc. (%) ↑		
Method	Push Cube	Flip Hamburger	Pack Cubes	Push Cube	Flip Hamburger	Pack Cubes
Ours	**0.97**$_{\pm 0.01}$	0.78$_{\pm 0.17}$	**0.86**$_{\pm 0.11}$	**89.8**	**96.0**	**89.8**
Ours (w/o state expansion)	0.78$_{\pm 0.30}$	0.69$_{\pm 0.42}$	0.83$_{\pm 0.12}$	58.4	93.9	81.7
Robometer	0.74$_{\pm 0.09}$	**0.85**$_{\pm 0.10}$	0.79$_{\pm 0.07}$	**89.5**	84.3	83.8

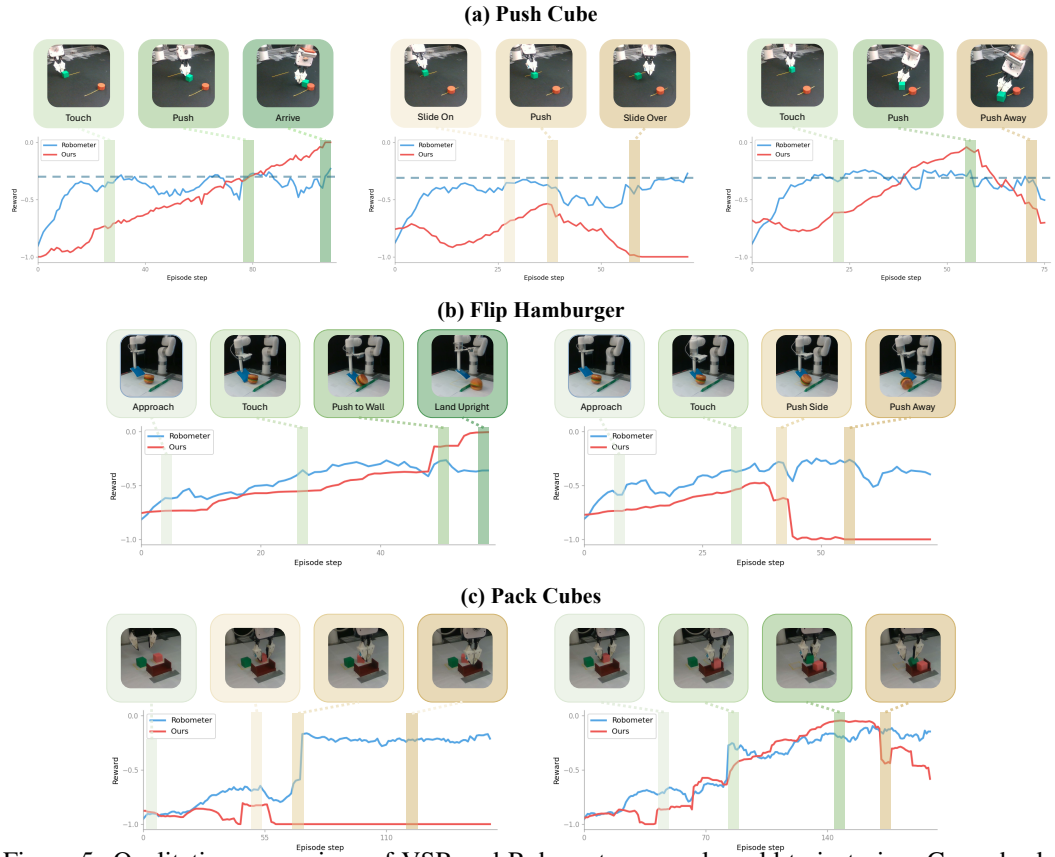


Figure 5: Qualitative comparison of VSR and Robometer on real-world trajectories. Green backgrounds mark states reflecting progress toward task success, while yellow backgrounds mark execution errors or deviations from successful execution. Across these examples, VSR tracks task progress and responds to setbacks due to manipulation errors, whereas Robometer sometimes assigns high scores prematurely or fails to reflect further progress or errors.

petitive with the best baselines on Push Cube. Among these baselines, SGFT is particularly relevant as it also transfers a source-simulation value to target-domain. Under the same DSRL framework, VSR achieves an average final success rate that is about 13.7 percentage points higher across the three tasks despite the SGFT baseline's access to privileged target-simulator states, consistent with our hypothesis that explicitly expanding state coverage can improve the quality of value guidance.

4.3 Evaluating Value Transfer to the Real World

In this section, we evaluate the quality of the transferred value function in the real world through both quantitative metrics and qualitative trajectory analysis.

Quantitative evaluation. We report two metrics in Tab. 2. (i) *VOC on human expert demonstrations* in Robometer [10]: the per-trajectory Pearson correlation ($\in [-1, 1]$, higher is better) between predicted values and expert timestep value (i.e., negative remaining steps-to-success). We outperform Robometer on PushCube and PackCube, while Robometer is slightly better on Hamburger.

(ii) *Pairwise terminal-outcome ranking accuracy*. We construct a shared evaluation set by subsampling real-world training and evaluation rollouts collected with DSRL + binary rewards and the base diffusion policy. Neither VSR nor Robometer is used to generate these rollouts, separating evaluation-data collection from the models being compared. We evaluate only the final state of each trajectory, assigning an outcome label of failure / success for Push Cube and Flip Hamburger, and phase-1 unfinished / phase-1 only / full success for Pack Cubes. Only pairs with different outcome labels are considered. Ranking accuracy is the fraction of these pairs for which the model assigns a higher value to the state with the higher outcome label. VSR achieves comparable accuracy to Robometer on Push Cube and higher accuracy on Flip Hamburger and Pack Cubes. Finally, removing state expansion degrades both metrics on all task, supporting that the broad simulated coverage is also important for the reliability of the value in the real world.

Qualitative evaluation. Fig. 5 compares VSR and Robometer on representative real-world trajectories. On smooth successful trajectories in Push Cube (a, left) and Flip Hamburger (b, left), VSR's value increases as the task progresses toward success. VSR also reflects manipulation errors and the resulting loss of progress. In Flip Hamburger (b, right), the value drops when the hamburger is pushed aside, while Robometer remains relatively high. VSR exhibits similar value drops when the gripper slides past the cube or pushes it away from the target in Push Cube (a, middle and right), and when the green cube is dropped outside the container in Pack Cubes (c, right). VSR also distinguishes task progress from gripper motion: in Pack Cubes (c, left), it stays low when the gripper moves above the container without grasping the pink cube, whereas Robometer rises substantially.

Together, these results suggest that VSR provides informative real-world reward guidance by reflecting task progress and errors, helping explain the improved downstream RL performance.

4.4 Analysis of State Expansion

Here, we analyze the role of iterative state expansion through experiments in simulation. We first examine the coverage and characteristics of the collected states, and then use controlled ablations to evaluate how individual components contribute to downstream RL performance.

Coverage & characteristics of expanded states. On Push Cube, Appendix Fig. 7 shows that the collected states broadly cover the task workspace, encompassing diverse combinations of cube poses and robot arm configurations. Appendix Fig. 8 further illustrates representative states and the performance of π_{sim} when initialized from them. Except for one state where the arm is at its reach limit, π_{sim} achieves a 100% success rate from all other illustrated states. The average steps to success provide reasonable dense supervision for value learning.

Controlled ablations. We compare the full method with variants removing: (1) the kNN distance threshold; (2) the non-zero success-rate filter on seed states, (3) the balanced reset distribution, instead sampling uniformly from $\mathcal{B}$; (4) co-training of π_{sim}, keeping the policy fixed during expansion; and (5) state expansion entirely. For (5), we collect value-training states along rollouts of π_{sim} from the original task reset distribution, matching the training-set size of the full method.

Table 3: State expansion ablations on three sim-to-sim tasks. We report the mean (standard error) deterministic success rate (%) over eight random seeds.

Setting	Push Cube	Flip Hamburger	Pack Cubes
Full method	**87.4** (1.8)	86.7 (2.7)	**85.2** (2.1)
w/o kNN threshold	83.3 (2.3)	87.1 (1.5)	83.0 (3.3)
w/o success filter	81.5 (3.7)	76.9 (1.9)	84.3 (1.0)
w/o balanced resets	83.6 (2.5)	86.0 (2.4)	80.5 (2.8)
w/o policy co-training	80.3 (3.4)	**89.4** (1.0)	79.8 (3.1)
w/o state expansion	81.8 (3.0)	81.6 (2.1)	81.0 (1.6)

We evaluate all variants after 30,000 DSRL environment steps on Push Cube and Flip Hamburger, and 50,000 steps on Pack Cubes in their respective target simulations. As shown in Table 3, removing state expansion consistently lowers mean success rates across all three tasks, supporting our central claim that expanding training-state coverage improves downstream RL through learned value guidance. Individually removing any of components (1)–(4) also lowers mean success rates on at least two tasks.

5 Conclusion

We presented VSR, a framework that learns value functions in simulation and uses them as reward guidance for real-world reinforcement learning. By combining iterative state expansion, simulation policy optimization, and point-cloud value learning, VSR provides dense task-progress signals without requiring real-world reward-model training. Across sim-to-sim and sim-to-real experiments, VSR improves sample efficiency and task success, showing that simulation-trained values can be an effective reward guidance for real-world robot learning. We further discuss the limitations of the current framework in the Appendix Sec. D.

References

[1] J. Luo, C. Xu, J. Wu, and S. Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

[2] P. Intelligence. $\pi^*_{0.6}$: a VLA That Learns From Experience, 2025. URL https://arxiv.org/abs/2511.14759.

[3] Y. Li, X. Ma, J. Xu, Y. Cui, Z. Cui, Z. Han, L. Huang, T. Kong, Y. Liu, H. Niu, et al. Gr-rl: Going dexterous and precise for long-horizon robotic manipulation. *arXiv preprint arXiv:2512.01801*, 2025.

[4] K. Lei, H. Li, D. Yu, Z. Wei, L. Guo, Z. Jiang, Z. Wang, S. Liang, and H. Xu. Rl-100: Performant robotic manipulation with real-world reinforcement learning. *arXiv preprint arXiv:2510.14830*, 2025.

[5] C. Xu, J. T. Springenberg, M. Equi, A. Amin, A. Esmail, S. Levine, and L. Ke. Rl token: Bootstrapping online rl with vision-language-action models. *arXiv preprint arXiv:2604.23073*, 2026.

[6] J. Zhang, Y. Luo, A. Anwar, S. A. Sontakke, J. J. Lim, J. Thomason, E. Biyik, and J. Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. In *Conference on Robot Learning*, pages 460–488. PMLR, 2025.

[7] Q. Chen, J. Yu, M. Schwager, P. Abbeel, Y. Shentu, and P. Wu. Sarm: Stage-aware reward modeling for long horizon robot manipulation. *arXiv preprint arXiv:2509.25358*, 2025.

[8] H. Tan, S. Chen, Y. Xu, Z. Wang, Y. Ji, C. Chi, Y. Lyu, Z. Zhao, X. Chen, P. Co, et al. Robodopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025.

[9] S. Zhai, Q. Zhang, T. Zhang, F. Huang, H. Zhang, M. Zhou, S. Zhang, L. Liu, S. Lin, and J. Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint arXiv:2509.15937*, 2025.

[10] A. Liang, Y. Korkmaz, J. Zhang, M. Hwang, A. Anwar, S. Kaushik, A. Shah, A. S. Huang, L. Zettlemoyer, D. Fox, et al. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.

[11] T. Lee, A. Wagenmaker, K. Pertsch, P. Liang, S. Levine, and C. Finn. Roboreward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.

[12] R. Singh, A. Allshire, A. Handa, N. Ratliff, and K. Van Wyk. Dextrah-rgb: Visuomotor policies to grasp anything with dexterous hands. *arXiv preprint arXiv:2412.01791*, 2024.

[13] B. Wen, M. Trepte, J. Aribido, J. Kautz, O. Gallo, and S. Birchfield. Foundationstereo: Zero-shot stereo matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5249–5260, 2025.

[14] A. Y. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[15] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[16] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[17] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[18] Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Biyik, D. Held, and Z. Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. *arXiv preprint arXiv:2402.03681*, 2024.

[19] Y. J. Ma, J. Hejna, C. Fu, D. Shah, J. Liang, Z. Xu, S. Kirmani, P. Xu, D. Driess, T. Xiao, et al. Vision language models are in-context value learners. In *International Conference on Learning Representations*, volume 2025, pages 33984–34009, 2025.

[20] S. Chen, C. Harrison, Y.-C. Lee, A. J. Yang, Z. Ren, L. J. Ratliff, J. Duan, D. Fox, and R. Krishna. Topreward: Token probabilities as hidden zero-shot rewards for robotics. *arXiv preprint arXiv:2602.19313*, 2026.

[21] R. Tian, Y. Wu, and A. Bacjsy. Position: Good embodied reward models need bad behavior data, 2026.

[22] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[23] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[24] T. Lin, K. Sachdev, L. Fan, J. Malik, and Y. Zhu. Sim-to-real reinforcement learning for vision-based dexterous manipulation on humanoids. *arXiv preprint arXiv:2502.20396*, 2025.

[25] T. He, Z. Wang, H. Xue, Q. Ben, Z. Luo, W. Xiao, Y. Yuan, X. Da, F. Castañeda, S. Sastry, et al. Viral: Visual sim-to-real at scale for humanoid loco-manipulation. *arXiv preprint arXiv:2511.15200*, 2025.

[26] A. Deshpande, M. Guru, R. Hendrix, S. Jauhri, A. Eftekhar, R. Tripathi, M. Argus, J. Salvador, H. Fang, M. Wallingford, et al. Molmob0t: Large-scale simulation enables zero-shot manipulation. *arXiv preprint arXiv:2603.16861*, 2026.

[27] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[28] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[29] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[30] Z.-H. Yin, C. Wang, L. Pineda, F. Hogan, K. Bodduluri, A. Sharma, P. Lancaster, I. Prasad, M. Kalakrishnan, J. Malik, et al. Dexteritygen: Foundation controller for unprecedented dexterity. *arXiv preprint arXiv:2502.04307*, 2025.

[31] P. Yin, T. Westenbroek, Z. Zhang, J. Tran, I. Dagnino, E. Shilamkar, N. Mbiziwo-Tiapo, S. Bagaria, X. Liu, G. Mullins, et al. Emergent dexterity via diverse resets and large-scale reinforcement learning. *arXiv preprint arXiv:2603.15789*, 2026.

[32] A. Wagenmaker, K. Huang, L. Ke, K. Jamieson, and A. Gupta. Overcoming the sim-to-real gap: Leveraging simulation to learn to explore for real-world rl. *Advances in Neural Information Processing Systems*, 37:78715–78765, 2024.

[33] M. Wu, L. Wu, Y. Wu, W. Huang, H. Fan, Z. Hu, H. Geng, J. Li, J. Ying, L. Yang, et al. Simlauncher: Launching sample-efficient real-world robotic reinforcement learning via simulation pre-training. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7933–7940. IEEE, 2025.

[34] J. Hu, P. Stone, and R. Martín-Martín. Slac: Simulation-pretrained latent action space for whole-body real-world rl. *arXiv preprint arXiv:2506.04147*, 2025.

[35] Y. Wang, S. Li, H. Niu, Z. Huang, W. Zhang, and H. Su. A recipe for efficient sim-to-real transfer in manipulation with online imitation-pretrained world models. *arXiv preprint arXiv:2510.02538*, 2025.

[36] P. Yin, T. Westenbroek, S. Bagaria, K. Huang, C.-a. Cheng, A. Kobolov, and A. Gupta. Rapidly adapting policies to the real world via simulation-guided fine-tuning. *arXiv preprint arXiv:2502.02705*, 2025.

[37] J. Levy, T. Westenbroek, K. Huang, F. Palafox, P. Yin, S. Omidshafiei, D.-K. Kim, A. Gupta, and D. Fridovich-Keil. Simulation distillation: Pretraining world models in simulation for rapid real-world adaptation. *arXiv preprint arXiv:2603.15759*, 2026.

[38] H. Hu, S. Mirchandani, and D. Sadigh. Imitation bootstrapped reinforcement learning. *arXiv preprint arXiv:2311.02198*, 2023.

[39] A. Wagenmaker, Y. Zhang, M. Nakamoto, S. Park, W. Yagoub, A. Nagabandi, A. Gupta, and S. Levine. Steering your diffusion policy with latent space reinforcement learning. In *Conference on Robot Learning*, pages 258–282. PMLR, 2025.

[40] Z. Zhou, A. Peng, Q. Li, S. Levine, and A. Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. In *International Conference on Learning Representations*, volume 2025, pages 32343–32368, 2025.

[41] W. Xiao, H. Lin, A. Peng, H. Xue, T. He, Y. Xie, F. Hu, J. Wu, Z. Luo, L. Fan, et al. Self-improving vision-language-action models with data generation via residual rl. *arXiv preprint arXiv:2511.00091*, 2025.

[42] Y. Guo, J. Zhang, X. Chen, X. Ji, Y.-J. Wang, Y. Hu, and J. Chen. Improving vision-language-action model with online reinforcement learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15665–15672. IEEE, 2025.

[43] W. Ye, Y. Zhang, H. Weng, X. Gu, S. Wang, T. Zhang, M. Wang, P. Abbeel, and Y. Gao. Reinforcement learning with foundation priors: Let the embodied agent efficiently learn on its own. *arXiv preprint arXiv:2310.02635*, 2023.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[45] S. Tao, A. Shukla, T.-k. Chan, and H. Su. Reverse forward curriculum learning for extreme sample and demonstration efficiency in reinforcement learning. *arXiv preprint arXiv:2405.03379*, 2024.

[46] Y. Xiong, C. Zhou, X. Xiang, L. Wu, C. Zhu, Z. Liu, S. Suri, B. Varadarajan, R. Akula, F. Iandola, et al. Efficient track anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11513–11524, 2025.

[47] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. Pmlr, 2017.

[48] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020.

[49] B. Wen, S. Dewan, and S. Birchfield. Fast-foundationstereo: Real-time zero-shot stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7513–7524, 2026.

[50] Z.-H. Yin, S. Yang, and P. Abbeel. Object-centric 3d motion field for robot learning from human videos. *Advances in Neural Information Processing Systems*, 38:55923–55943, 2026.

[51] X. Li, T. Zhao, X. Zhu, J. Wang, T. Pang, and K. Fang. Planning-guided diffusion policy learning for generalizable contact-rich bimanual manipulation. *arXiv preprint arXiv:2412.02676*, 2024.

[52] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[53] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T.-k. Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

A Implementation Details

A.1 Iterative State Expansion and Policy Optimization

Here, we introduce the *Iterative State Expansion and Policy Optimization* method details in Sec. 3.1.1 of the main paper. The pseudocode is given in Algorithm 1.

Algorithm 1 State Expansion with Policy Updates

Input: Policy π_{sim}; Initial States $\mathcal{S}_0$;
Output: Expanded State Buffer $\mathcal{B}$;
$\mathcal{B} \leftarrow \mathcal{S}_0$;
for $i = 1, \ldots, N_{\text{round}}$ **do**
  $\mathcal{B}_{\text{new}} \leftarrow \emptyset$;
  for $j = 1, \ldots, N_{\text{expand}}$ **do**
    $s_{\text{seed}} \leftarrow \text{Sample}(\mathcal{B})$;
    $n_{\text{step}} \sim \text{Unif}[1, T]$;
    $s_{\text{cand}} \leftarrow \text{NoisyRollout}(\pi_{\text{sim}}, s_{\text{seed}}, n_{\text{step}})$;
    if $d_{\text{KNN}}(s_{\text{cand}}, \mathcal{B}) > \epsilon$ **then**
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{s_{\text{cand}}\}$;
      $\mathcal{B}_{\text{new}} \leftarrow \mathcal{B}_{\text{new}} \cup \{s_{\text{cand}}\}$;
    end if
  end for
  $\mathcal{B}_{\text{old}} \leftarrow \mathcal{B} \setminus \mathcal{B}_{\text{new}}$;
  $\rho_{\mathcal{B}} \leftarrow \frac{1}{2}\text{Unif}(\mathcal{B}_{\text{new}}) + \frac{1}{2}\text{Unif}(\mathcal{B}_{\text{old}})$;
  Update π_{sim} with PPO starting from $\rho_{\mathcal{B}}$.
end for

A.1.1 K-Nearest-Neighbor Distance Computation

We use a k-nearest-neighbor (kNN) novelty criterion to decide whether a newly generated state should be added to the online initialization set. For all tasks, we set $k = 50$, unless the reference pool contains fewer than 50 states, in which case we use all available reference states. For each candidate simulator state s, we extract a low-dimensional geometric feature $\phi(s)$ and compare it against the current reference state pool $\mathcal{R}$. The kNN novelty score is defined as the average distance to the k nearest reference states:

$$\eta_k(s; \mathcal{R}) = \frac{1}{k} \sum_{s_j \in \mathcal{N}_k(s, \mathcal{R})} d(\phi(s), \phi(s_j)).$$

Here, $\mathcal{N}_k(s, \mathcal{R})$ denotes the k nearest neighbors of s in the reference pool under the task-specific metric $d(\cdot, \cdot)$. A candidate state is admitted into the initialization pool only if

$$\eta_k(s; \mathcal{R}) > \tau,$$

where τ is a novelty threshold. We set the threshold so that approximately 10% of rollout samples are added to the buffer during the first 10 rounds.

Task-specific features. For all tasks, the kNN distance is computed from compact geometric features of the gripper/TCP and task-relevant objects. Specifically, we use their positions and orientations extracted from the privileged simulator state, and normalize each component by a predefined task-specific range that covers the maximum variation we aim to collect during state expansion. For Push Cube, the feature includes the TCP, cube, and target poses. For Pack Cubes, the feature includes the TCP, container, and two cube poses, with the two cubes ordered consistently. For Flip Hamburger, the feature includes the TCP, hamburger, and wall poses. Since the hamburger is approximately symmetric around one local axis, we ignore the corresponding rotation component when computing the distance, so that physically similar rotations are not treated as novel states. The final kNN novelty score is computed as the average distance to the nearest reference states in this normalized feature space.

A.1.2 Details of State Expansion and Policy Optimization

As described in the main paper and summarized in Algorithm 1, we start from a PPO [44] policy trained in the original reset region and iteratively alternate between state expansion and policy optimization. The state buffer $\mathcal{B}$ (Algo. 1 Lines 2–3) is initialized with 1000 states sampled from the original reset region for Push Cube and Flip Hamburger. For Pack Cubes, it is initialized with 2000 states: 1000 reset states and 1000 states at the first task milestone. At each expansion round (Algo. 1 Lines 4–14), we sample states from the current buffer and run $N_{\text{expand}} = 400$ noisy rollouts (Algo. 1 Lines 7–9). Candidate states satisfying the KNN novelty criterion are added to the buffer (Lines 10–13), yielding roughly 19, 31, and 99 new states per round on average for Push Cube, Flip Hamburger, and Pack Cubes, respectively.

After each expansion round, we construct the balanced reset distribution $\rho_{\mathcal{B}}$ from newly added and older states (Algo. 1 Lines 15–16), and perform 25 PPO update iterations (Algo. 1 Line 17). Each update uses 122880 rollout samples for Push Cube and Flip Hamburger and 153600 for Pack Cubes, collected from 1024 parallel environments over 120 and 150 steps, respectively, with half of the environments initialized from $\mathcal{B}_{\text{new}}$ and half from $\mathcal{B}_{\text{old}}$. This sampling lets the policy adapt to newly discovered states while maintaining performance on previously covered regions.

We run up to $N_{\text{round}} = 131$ expansion rounds (Algo. 1 Lines 4–18), for up to 3255 PPO update iterations in total. For Pack Cubes, subsequent data collection uses the checkpoint at iteration 3001, after 121 expansion rounds. The resulting state buffer contains around 3500 feasible states for Push Cube, around 5000 for Flip Hamburger, and around 14000 for Pack Cubes.

A.2 Value Labeling and Distillation

A.2.1 Value Labeling

As described in the main paper, after completing state expansion and policy optimization, we use the final simulation policy π_{sim} to generate value labels for states in the buffer $\mathcal{B}$. For each state $s_0 \in \mathcal{B}$, we roll out π_{sim} in simulation and record point-cloud observations along the trajectory. For efficiency, we store observations and labels every four environment steps.

Each sampled state is labeled by its negative distance to success. If a rollout succeeds at timestep T_{succ}, then a sampled state at timestep t is assigned the raw label

$$\tilde{y}_t = -(T_{\text{succ}} - t).$$

Thus, states closer to success receive larger values, with the success state receiving value 0. If the rollout fails, we assign all sampled states in that trajectory the minimum raw label

$$\tilde{y}_t = -T_{\max},$$

where $T_{\max}$ is the maximum episode length of the corresponding task. In our experiments, $T_{\max} = 120$ for Push Cube and Flip Hamburger, and $T_{\max} = 150$ for Pack Cubes.

To make the subsequent value distillation training numerically easier, we rescale all labels by a constant factor of 0.04:

$$y_t = 0.04\,\tilde{y}_t.$$

This produces a supervised dataset

$$\mathcal{D} = \{(o_t, y_t)\},$$

where o_t denotes the recorded point-cloud observation at timestep t.

A.2.2 Value Distillation

After obtaining value labels from simulation, we distill them into a vision-based value function V_ψ. For each simulator state s, let o_s denote its corresponding observation. We train V_ψ to predict the value label from o_s, so that the learned value function can be used in the real world without access to simulator states. To reduce the visual sim-to-real gap, we represent o_s as a point cloud

augmented with per-point object segmentation labels. The segmentation labels are obtained from a segmentation tracker.

Distributional value prediction. Rather than directly regressing a scalar value, we use distributional value prediction. Let the rescaled value range be $[-C, 0]$, where $C = 0.04T_{\max}$ for each task. We divide this range into $K = 100$ uniformly spaced bins with centers

$$z_j = -C + \frac{j-1}{K-1}C, \qquad j = 1, \dots, K.$$

The network outputs a categorical distribution

$$p_\psi(j \mid o_s) = \mathrm{softmax}(f_\psi(o_s))_j$$

over these bins.

To train this distributional predictor, we convert each continuous value label y_s into a target categorical distribution $\Pi_K(y_s)$ over the same bins. Let $\bar{y}_s = \mathrm{clip}(y_s, -C, 0)$. We define $\Pi_K(y_s)$ by linearly interpolating $\bar{y}_s$ onto its two nearest bin centers. Specifically, let ℓ and u be the lower and upper bin indices such that

$$z_\ell \le \bar{y}_s \le z_u, \qquad u = \ell + 1.$$

For non-boundary values, the target distribution is

$$\Pi_K(y_s)_j = \begin{cases} \dfrac{z_u - \bar{y}_s}{z_u - z_\ell}, & j = \ell, \\ \dfrac{\bar{y}_s - z_\ell}{z_u - z_\ell}, & j = u, \\ 0, & \text{otherwise.} \end{cases}$$

If $\bar{y}_s$ lies on the boundary of the value range, all probability mass is assigned to the corresponding endpoint bin. We train the network with the cross-entropy loss

$$\mathcal{L}_{\mathrm{value}}(\psi) = \mathbb{E}_{(o_s, y_s)} \left[\mathrm{CE}\big(\Pi_K(y_s), p_\psi(\cdot \mid o_s)\big) \right].$$

At test time, the scalar value estimate is computed as the expectation over bin centers:

$$V_\psi(o_s) = \sum_{j=1}^{K} p_\psi(j \mid o_s) z_j.$$

Network architecture. The input point cloud is downsampled to $N = 2000$ workspace-normalized points. Each point is represented by its 3D coordinate and a one-hot object segmentation label. We use a PointTokenizer to convert the point cloud into local tokens. Specifically, we first sample 512 center points using farthest point sampling. For each center point, we group its 32 nearest neighbors using KNN. For each local group, the input feature of each neighbor is the concatenation of its absolute coordinate, coordinate relative to the group center, and one-hot segmentation label. This local feature is projected to a 384-dimensional feature, max-pooled over the 32 neighbors, and passed through another linear layer to produce a 384-dimensional token.

We add a learned position embedding to each token. The position embedding is computed from the center coordinate through a two-layer MLP. We then prepend a learnable aggregation token, resulting in a token sequence of length $1 + 512 = 513$. The sequence is processed by a Transformer encoder with hidden dimension 384, 6 layers, 12 attention heads, feed-forward dimension 1536, and dropout rate 0.2.

The output corresponding to the aggregation token is used as the global point-cloud representation. We project it to a 512-dimensional feature with a linear layer followed by layer normalization.

The value head is a four-layer MLP with hidden width 1024 and ReLU activations. It takes the 512-dimensional global point-cloud feature as input and outputs $K = 100$ logits, corresponding to the 100 value bins. After applying a softmax, these logits define the categorical value distribution $p_\psi(\cdot \mid o_s)$. The scalar value used for reward shaping is computed as the expectation over the bin centers.

Depth and Point-Cloud Processing. Accurate depth is essential for constructing the point-cloud observation o_s, yet real-world depth estimation is inevitably noisy. We mitigate this from several angles. First, rather than using the depth maps produced directly by the RealSense camera, we feed the camera's two raw IR (stereo) images into Fast-FoundationStereo [49] to recover depth, which yields more accurate and consistent estimates. Second, we find that the depth values near object boundaries remain particularly noisy, so during simulation training we apply a data augmentation that adds random noise to the depth of object edges, following [50], making the value network robust to this effect. Finally, we additionally adopt the flying point augmentation of [51], which randomly perturbs a small fraction of points in the point cloud with large noise.

Training details. We train the value network with a batch size of 400 for 2000 epochs. Training is performed on a single node with 8 NVIDIA A800 GPUs. Depending on the task and dataset size, value distillation takes approximately 12–16 hours.

A.3 Online Reinforcement Learning

We use the Noise-Aliased variant of DSRL (DSRL-NA) [39] for online reinforcement learning.

Let π_{dp} denote the pretrained diffusion policy, which is kept frozen throughout RL training. The SAC [52] actor is a latent-noise steering policy: it takes three RGB images (including two third-person camera views and one wrist-camera view), together with robot proprioception as input. The image inputs are encoded with a compact CNN, concatenated with robot proprioception, and passed to an MLP that parameterizes a tanh-squashed Gaussian distribution over latent-noise action chunks. A sampled latent-noise chunk is then decoded by π_{dp} into a robot action chunk of length 4.

Following DSRL-NA, we use two twin-Q critics: an action-space critic $Q^A(s, a)$ and a latent-noise-space critic $Q^W(s, w)$. Both use the same RGB/proprioception encoder design as the actor, followed by MLP Q-heads; Q^A additionally conditions on decoded action chunks, while Q^W conditions on latent-noise chunks. Q^A is trained with temporal-difference learning on decoded actions. Q^W is trained by distilling $Q^A(s, \pi_{\mathrm{dp}}(s, w))$, allowing the SAC actor to be optimized directly in the diffusion latent-noise space. During evaluation, we use the deterministic SAC policy, i.e., the mean action of the actor before diffusion decoding.

For VSR, we freeze the sim-to-real value function and use it to provide the dense reward signals throughout RL training. Both VSR and Robometer combine their method-specific dense reward with the sparse binary reward. Specifically, we rescale these dense rewards to $[-1, 0]$, and use $r = r_{\mathrm{dense}} + \alpha r_{\mathrm{binary}}$. We set $\alpha = 0.2$ for both methods and across all tasks.

B Task Settings

We build all simulation environments in ManiSkill3 [53], with SAPIEN [48] used as the underlying physics simulation and rendering backend.

Across all tasks, the gripper is modeled as rigid in simulation, while the real-world gripper is soft. This introduces a dynamics gap between simulation and the real world. Despite this gap, the simulation-trained value function remains effective when transferred to real-world experiments.

Push Cube. The cube has a side length of 4 cm. The target cylinder has a radius of 2.5 cm and a height of 2.5 cm.

Flip Hamburger. The shovel used in the real-world setup is 3D-printed with TPU, making it slightly compliant, whereas the simulated shovel is modeled as rigid. The wall height is 3 cm in the standard setting. For the sim-to-sim transfer setting, we increase the wall height to 4 cm. The hamburger has a bottom radius of approximately 5 cm and a height of approximately 8 cm. In simulation, the hamburger is modeled as a rigid body, and its collision shape is approximated by a convex hull. In the real world, the toy hamburger is elastic, with only the bottom part being relatively

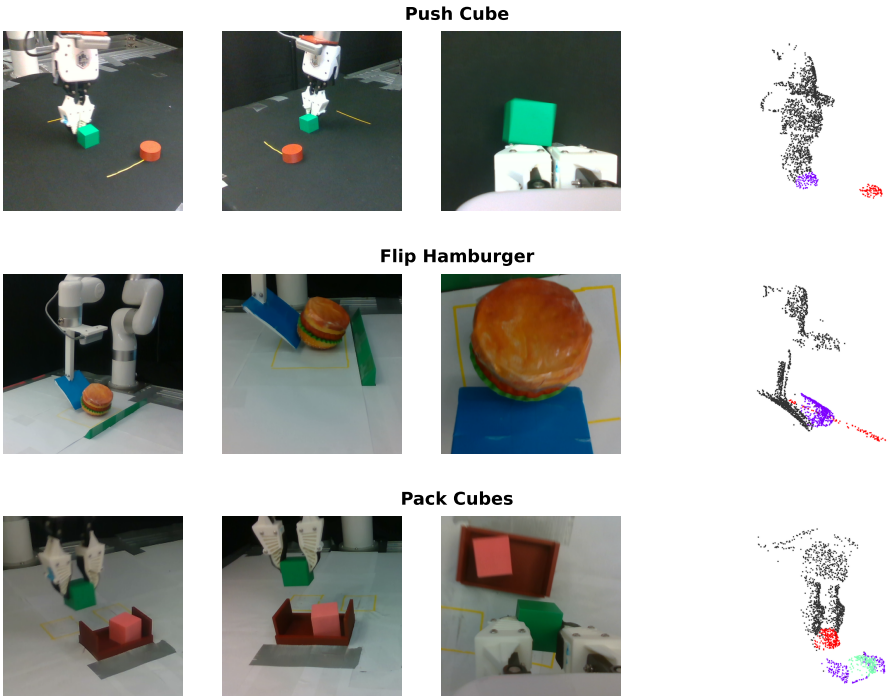


Figure 6: Example observations for the three manipulation tasks. In the point clouds, different colors denote different segmentation labels.

rigid. These differences create a noticeable dynamics gap, yet our value function still provides useful guidance after transfer.

Pack Cubes. The container has a width of $10\ \mathrm{cm}$ along the y-axis and a length of $6\ \mathrm{cm}$ along the x-axis. The base has a depth of $0.5\ \mathrm{cm}$. The front and back walls have a height of $1\ \mathrm{cm}$, while the two side walls have a height of $4\ \mathrm{cm}$, matching the cube height. The taller side walls are designed to prevent a shortcut strategy in which the robot first randomly drops the first cube into the container and then pushes it aside with the second cube to make space.

We collect a small set of expert demonstrations to train the base diffusion policy for each task. Specifically, we use 15 demonstrations for Push Cube, 10 for Flip Hamburger, and 30 for Pack Cubes.

We provide example observations for all three tasks in Figure 6. The DSRL policy takes the cropped RGB images as input, while the sim-to-real value function takes the corresponding segmentation-labeled point cloud.

C More Experiment Results

C.1 Coverage & Characteristics of Expanded States

Here, we show the visual demonstration for state expansion coverage of Push Cube task in Fig. 7, and the representative states as well as the performance of π_{sim} when initialized from them in Fig. 8.

C.2 Real World RL Training Curves

We present the real-world RL training curves for all three tasks in Figure 9. Across all three tasks, our method shows a consistent increase in success rate and a consistent decrease in completion time over the course of online RL training.

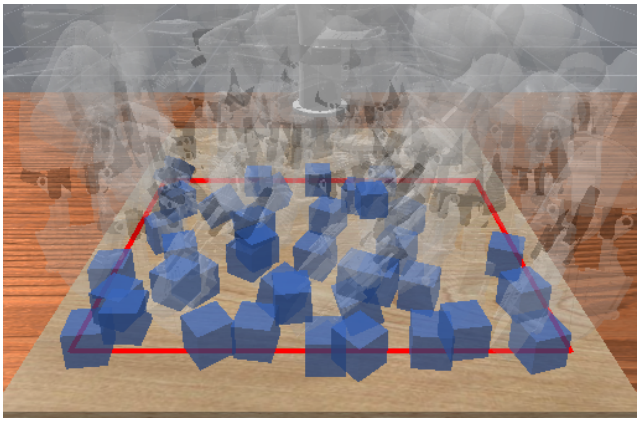


Figure 7: Spatial coverage of expanded states on Push Cube. The red rectangle marks the horizontal projection of the task workspace used during expansion. The collected states broadly cover this workspace.

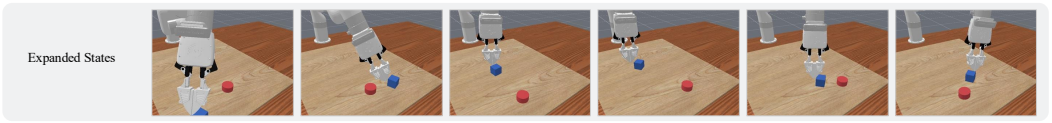


Figure 8: Representative states collected from our state expansion process. We show the success rate and average steps to success of π_{sim} from each state. The variation in average steps to success across these configurations provides fine-grained supervision for value learning.

D Limitations

While our real-world experiments show that the learned value function tolerates moderate sim-to-real dynamics discrepancies, its transfer may still degrade under a large dynamics gap. When the gap is severe, simulation can admit short-cut behaviors that are physically unrealizable in the real world, causing the value function to give erroneous estimates that mislead downstream RL. Characterizing which dynamics gaps most degrade transfer, and thereby delineating the capability boundary of value sim-to-real, is an important direction for future work.

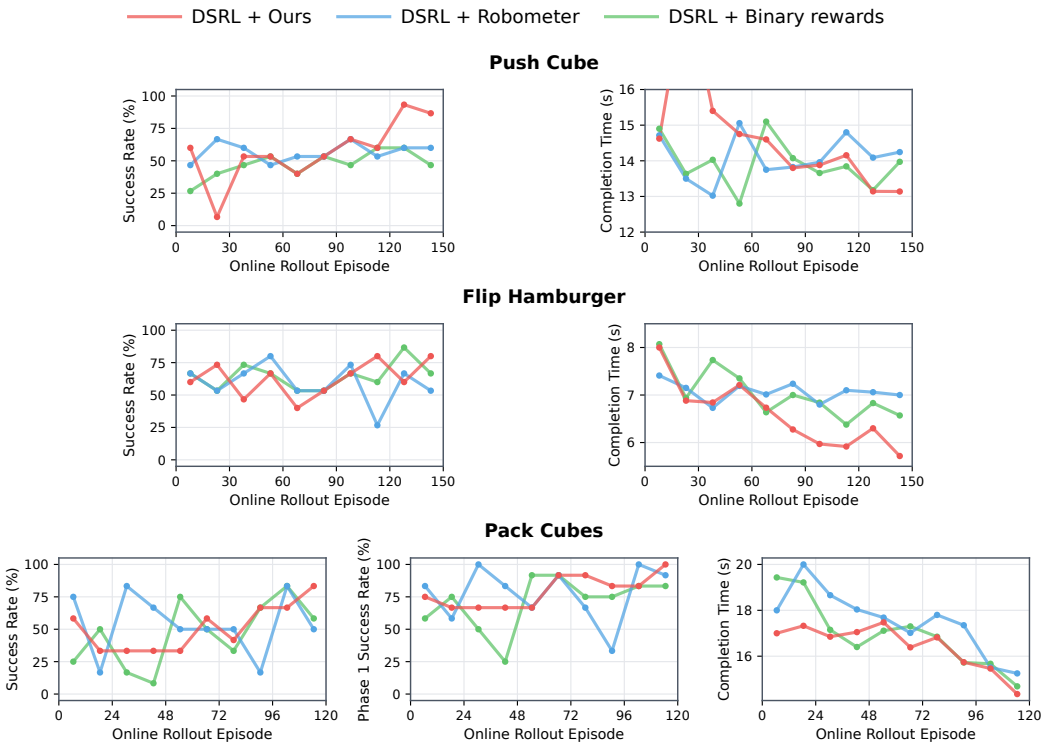


Figure 9: Real-world RL training curves across 3 real-world tasks. The online rollout budget is split into 10 windows, and we plot the metrics within each window. Note that these are not evaluation results.