ROBOSTEP: ROBOT REWARDS BENEFIT FROM TRACKING TASK PROGRESS STEP BY STEP

Anonymous authors
Paper under double-blind review

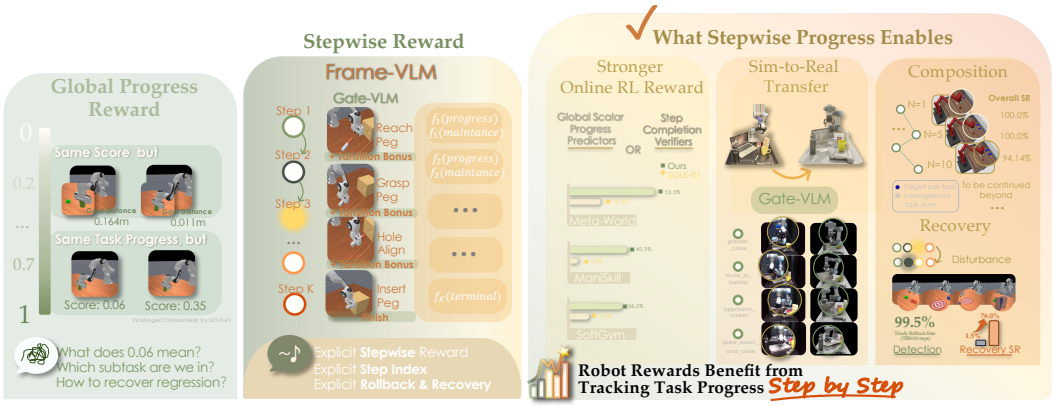


Figure 1: **RoboStep tracks task progress step by step instead of predicting a single global visual progress score.** The persistent step state determines which reward is active, while Gate-VLM verifies semantic transitions, enabling explicit reward routing, composition, and efficient rollback after regressions.

ABSTRACT

Progress reward models aim to infer task progress from visual observations, especially when privileged task states are less accessible in the real world than in simulation. Yet progress is inherently task-specific and history-dependent, making a globally normalized score difficult to calibrate. Scalable supervision therefore often relies on frame positions in successful demonstrations, implicitly coupling progress with elapsed time and providing weak supervision for stagnation, regression, and recovery. In this work, we present RoboStep, a framework for constructing online robot rewards that explicitly tracks task progress step by step, using pretrained VLM semantics for step completion verification rather than global progress calibration. We use a *Frame-VLM* to decompose tasks and generate step-local dense reward functions, while a *Gate-VLM* decides when the reward advances to the next step and switches the active reward function. Across extensive simulated manipulation experiments, we systematically compare these two roles of VLMs under matched policy-learning settings. Combined with recent pipelines that reconstruct real-world scenes in simulation, RoboStep further enables policies trained with structured rewards in simulation to transfer back to corresponding real scenes. Overall, we show that robot learning benefits from tracking task progress step by step: for online reward, VLMs are better used as step-completion verifiers than as global scalar progress predictors.

1 INTRODUCTION

Task progress provides an important signal for robot learning, yet privileged states needed to characterize semantic progress, like poses and distances, are often difficult to access in the real world. Visual reward models therefore aim to infer task progress directly from images or videos at inference (Ma et al., 2023a; Lee et al., 2026; Schroeder et al., 2026) for downstream uses such as policy

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.

learning and data annotation. However, a scalar score does not explicitly reveal which subgoals are completed, pending or invalidated, and dense human progress annotations are inevitably subjective and costly. Generally, a globally normalized task progress score is difficult to calibrate (Zhang et al., 2026a). Scalable alternatives often use normalized timestamps from successful demonstrations as supervision (Zang et al., 2025; Zhang et al., 2025a; Zhang et al.; Liang et al., 2026; Wang et al., 2026b; Lee et al., 2026), providing only weak supervision for stagnation and regression (Jeung et al., 2026; Ji et al., 2026; Liu et al., 2026).

A related lesson from language-model reasoning is that supervision on intermediate reasoning steps can be substantially more informative than final-outcome supervision alone (Lightman et al., 2023). In robotic visual reward modeling, several methods similarly introduce step awareness to improve semantic grounding by inferring the current task stage from visual observations (Chen et al., 2026a; Feng et al., 2026; Yang et al., 2026). However, these approaches often require task-specific expert demonstrations or boundary annotations to adapt to a new task. Moreover, the inferred step and within-step progress are commonly recombined again into a global task-level scalar for downstream policy or data weighting. It remains unclear how to efficiently obtain and exploit explicit step information without task-specific adaptation for online reward learning.

Unlike visual reward models that infer task progress from rollout observations, programmatic and structured reward design instead specifies reward structure directly from task and environment descriptions (Xie et al., 2024; Ma et al., 2023b). Stepwise variants range from one-shot subtask-completion bonuses to executable reward machines (Toro Icarte et al., 2018; Venuto et al., 2024; Zhang et al., 2025b; Castanyer et al., 2026). Yet it remains unclear how task steps should shape dense rewards, inform the policy, and handle regressions in progress. Moreover, in online RL, step transitions often rely on programmatic simulator states, which are difficult to access during real-world deployment. Pretrained VLMs offer an accessible source of semantic evidence for such transitions, and have been widely used for task decomposition, subgoal grounding (Fang et al., 2024; Huang et al., 2025) and task-completion signal extraction from token probabilities (Chen et al., 2026c).

This motivates our question: *Can the native semantic abilities of VLMs provide generalizable step-level signals for structuring online robot rewards?*

In this work, we present RoboStep, a framework for constructing online robot rewards through persistent, stepwise tracking of task progress. RoboStep uses a **Frame-VLM** to decompose a task and generate step-local dense reward functions from the task description and environment states. During interaction, a persistent step state determines which step-local rewards are active, while a **Gate-VLM** verifies the step completion and authorizes transitions. Upon a positive verification, RoboStep updates the active reward functions, exposes the updated step index to the policy and issues a one-time transition bonus. By representing task progression as an explicit reward state machine, RoboStep makes completion and regression auditable and provides a structured interface for rollback and recovery when previously satisfied conditions are invalidated.

Across controlled manipulation experiments, we show that robot learning benefits from tracking task progress step by step: for online rewards, VLMs are more effective as step-completion verifiers than as global scalar progress predictors. Combined with recent progress in real-to-sim scene reconstruction (Torne Villasevil et al., 2024; Han et al., 2026; Zhang et al., 2026b; Zhao et al., 2025), RoboStep further enables simulation-trained visual policies to transfer back to corresponding real scenes, with deployment-time step state information maintained through visual **Gate-VLM** verification.

To summarize, our contributions are as follows:

- We introduce RoboStep, an online reward framework that constructs robot rewards through stepwise tracking of task progress, where visual VLM verification maintains the persistent step state used to route step-local dense rewards.
- Across controlled manipulation experiments, we demonstrate that VLMs are better used as step-completion verifiers than as global scalar progress predictors for online rewards.
- With real-to-sim scene reconstruction, we enable visual policies trained in simulation to transfer back to corresponding real scenes with visually maintained step information.
- We systematically study key dimensions of stepwise rewards, including reward routing, policy step context, transition verification, and regression/recovery, and report practical observations that may inform reward design.

2 Related Work

Visual Progress Rewards for Robot Learning. Visual reward models estimate task-conditioned progress from visual observations for policy learning or data selection (Ma et al., 2023c; Sontakke et al., 2023; Rocamonde et al., 2024; Li et al., 2024; Wang et al., 2024; Fu et al., 2024; Ma et al., 2025). LIV (Ma et al., 2023a) learns a goal-conditioned value from temporally ordered videos, while ReWiND (Zhang et al., 2025a) augments temporal supervision with rewound trajectories and adds a one-time bonus from an external binary success signal. Large Reward Models (Wu et al., 2026) supports prediction of relative progress, discretized absolute progress and completion, whereas World Value Models (Wang et al., 2026b) predict distributional value chunks for advantage-weighted regression or filtered behavior cloning. RoboReward (Lee et al., 2026) and SOLE-R1 (Schroeder et al., 2026) predict episode-level and per-timestep progress respectively. At the token level, RoboMeter (Liang et al., 2026) introduces learnable progress and preference tokens for VLMs to learn, while TOPReward (Chen et al., 2026c) directly reads completion probabilities from frozen VLM output tokens. Stage-aware methods further introduce task structure into progress estimation (Chen et al., 2026a; Feng et al., 2026; Yang et al., 2026; Chen et al., 2026b; Zhou et al., 2026; Tan et al., 2025; Xu et al., 2026b; Yong et al., 2026), typically predicting the current stage and within-stage progress from visual observations before finally converting them into a task-level progress signal. Despite these differences, many scalable approaches still rely on temporal ordering from successful trajectories, while some further incorporate synthesized reversals, observed failure trajectories, or labor-intensive human labels.

Our goal is to reconsider the role of visual progress models in online reward learning. Rather than repeatedly estimating a globally calibrated progress value, RoboStep maintains the current step as persistent reward state, routes step-local dense rewards accordingly, and uses the visual reasoning capabilities of VLMs only to verify semantic transitions. This turns global progress prediction into a sequence of stepwise verification decisions.

Programmatic and Structured Reward Design. A complementary line of work directly encodes task structure into reward design (Yu et al., 2023; Ma et al., 2024). Reward Machines (Toro Icarte et al., 2018) maintain persistent task state with a persistent automaton, while Text2Reward (Xie et al., 2024) and Eureka (Ma et al., 2023b) generate executable rewards with language models. Code-as-Reward (Venuto et al., 2024) further decomposes tasks with VLMs and persistently records subtask completion through one-shot auxiliary bonuses, and PRM4RL (Zhang et al., 2025b) generates subtask structure but scalarizes local progress into a task-level potential.

ARM-FM (Castanyer et al., 2026) is particularly related with our method, using foundation models to generate explicit reward machines. Its manipulation transitions, however, rely on simulator-accessible predicates and require manual reward adjustment. RoboStep not only systematically studies how step state routes dense rewards, conditions the policy, and handles transition errors and regression, but also further leverages pretrained VLM semantics to verify step transitions directly from visual observations. A detailed method comparison and task coverage is provided in Appendix G-1.

Real-to-Sim-to-Real Robot Learning. Real-world deployment often lacks simulator-side privileged quantities and may require additional sensing or estimation to deploy simulation-trained policies (Wang et al., 2026a). Real-to-Sim-to-Real methods (Torne Villasevil et al., 2024; Jain et al., 2025; Han et al., 2026; Zhang et al., 2026b; Zhao et al., 2025; Ranawaka et al., 2026; Xu et al., 2026a; Choi et al., 2026) can be used to reconstruct target scenes in simulation for privileged-state training before transferring policies back to the real world. Building on recent real-to-sim reconstruction advances (Zhang et al., 2026b), RoboStep trains policies with stepwise rewards in simulation and transfers them back to the corresponding real scene, where **Gate-VLM** can still maintain the step state information from visual observations.

3 Method

Overview. RoboStep represents task progress as a persistent sequence of semantic steps. Instead of predicting a globally calibrated scalar progress value, RoboStep maintains an explicit step index vector that determines which reward component should be active. A **Frame-VLM** compiles the

task description and available environment interface into step-local dense reward functions and descriptions. During interaction, a **Gate-VLM** verifies whether the current step has been completed from visual observations and authorizes transitions. Upon an accepted transition, RoboStep uses the updated step index to activate the corresponding reward function, to issue a one-time transition bonus and as the policy inputs. Full inplementation of RoboStep is in Appendix A.



Figure 2: **Overview of RoboStep.** **Frame-VLM** compiles a task-conditioned stepwise reward program, while **Gate-VLM** verifies semantic step transitions from RGB observations. The persistent step state routes step-local dense rewards and provides step context for policy learning.

3.1 Framework of Stepwise Task Compilation

Reward compilation. Let ℓ denote the task description, O_b the observable-state interface available in a simulation benchmark b, and $o_t^{(b)} \in O_b$ its current observable state. The **Frame-VLM** compiles the task into K ordered non-terminal steps and generates the reward recipe and invocation rule:

$$\mathcal{R}(\ell, O_b) = \left(\{(\phi_i, M_i, b_i, \mathcal{V}_i)\}_{i=0}^{K-1}, \chi \right). \tag{1}$$

Here ϕ_i evaluates progress within step i (e.g., approaching a target pose). M_i evaluates preserved conditions required for valid execution (e.g., maintaining a grasp or stable contact). They together define the dense reward sources for the active step. b_i is a one-time transition bonus. $\mathcal{V}_i$ is the visual completion standard description provided to **Gate-VLM** (e.g., whether the intended object is grasped). The invocation rule χ produces the binary trigger $\chi_t \in \{0, 1\}$ that determines when semantic verification is requested. Instantiations of χ are detailed in Appendix A-2. **Frame-VLM** does not use benchmark-native rewards and success predicates to generate $\mathcal{R}(\ell, O_b)$. The official success metrics are reserved only for terminal success evaluation.

Persistent step state and routed reward. Let $z_t \in \{0, \ldots, K\}$ denote the persistent step state where K is the terminal state. We define the activation indicator

$$A_i(t) = \mathbf{1}[z_t = i], \qquad \sum_{i=0}^{K-1} A_i(t) = 1 \quad (z_t < K). \tag{2}$$

For each step, the local dense reward is computed as

$$D_i(o_t^{(b)}, o_{t-1}^{(b)}) = \Delta\Phi_i(o_t^{(b)}, o_{t-1}^{(b)}) + M_i(o_t^{(b)}), \tag{3}$$

$$\Delta\Phi_i(o_t^{(b)}, o_{t-1}^{(b)}) = s_i C_i \left(\gamma_i \phi_i(o_t^{(b)}) - \phi_i(o_{t-1}^{(b)}) \right) \tag{4}$$

is a bounded local progress change. Here s_i is a step-specific scale factor and C_i is a clipping operator, both are fixed reward hyperparameters specified in the benchmark or task configuration.

Let $e_t = \mathbf{1}[\text{accepted step transition at } t]$ be a binary scalar. The stepwise reward is

$$r_t^{\text{step}} = \sum_{i=0}^{K-1} A_i(t) D_i(o_t^{(b)}, o_{t-1}^{(b)}) + b_{z_t} e_t + P_t, \tag{5}$$

where P_t contains signed safety-related reward terms shared across steps. Once a transition is accepted, the previous step reward is deactivated and the next step reward becomes active. The transition bonus is paid only once at the transition boundary.

3.2 GATE-VLM STEPWISE REWARD INTERFACE

Invocation and semantic verification. The reward program provides a mechanism for deciding when the Gate-VLM is invoked for verification. With $\chi_t \in \{0, 1\}$ denoting the invocation trigger, when $\chi_t = 1$, the Gate-VLM receives the task description ℓ, current step index z_t, visual completion description $\mathcal{V}_{z_t}$, and RGB observations $\mathcal{I}_t$. Its output is encoded as

$$v_t = \mathbf{1}\left[\text{Gate-VLM}(\ell, z_t, \mathcal{V}_{z_t}, \mathcal{I}_t) = \texttt{success}\right], \tag{6}$$

with $v_t = 0$ when $\chi_t = 0$. Here $\mathcal{V}_{z_t}$ is the completion description generated by Frame-VLM, and the RGB input is

$$\mathcal{I}_t = \left[I^{\text{start}}, I_t^{\text{history}}, I_t\right], \tag{7}$$

where I^{start} is the episode-start reference frame, I_t is the current frame captured at invocation, and I_t^{history} is the optional ordered sequence of recent frames. The history depends on the benchmark and is empty in the two-frame implementation.

An accepted transition requires both invocation and successful semantic verification:

$$e_t = \chi_t v_t, \qquad z_{t+1} = z_t + e_t. \tag{8}$$

3.3 STEP-CONDITIONED POLICY

Let $\mathbf{e}(z_t)$ denote the one-hot context of the persistent step state.

$$\mathbf{e}(z_t) = \text{onehot}(z_t) \in \{0, 1\}^{K+1} \tag{9}$$

For the three simulation benchmarks, the PPO actor–critic (Schulman et al., 2017) receives the benchmark observation together with this persistent step context:

$$a_t \sim \pi_\theta\left(\cdot \mid o_t^{(b)}, \mathbf{e}(z_t)\right), \qquad V_\psi = V_\psi\left(o_t^{(b)}, \mathbf{e}(z_t)\right). \tag{10}$$

For Real-to-Sim-to-Real transfer, we reconstruct the target real scene in simulation and train a deployment-compatible visual actor. The actor receives

$$x_t^{\text{actor}} = \left[f_{\text{vis}}(I_t), x_t^{\text{robot}}, \mathbf{e}(z_t)\right], \tag{11}$$

where f_{vis} extracts visual features (Oquab et al., 2023) and x_t^{robot} contains robot proprioception.

The critic additionally receives privileged simulator information (Pinto et al., 2018):

$$x_t^{\text{critic}} = \left[s_t^{\text{sim}}, \mathbf{e}(z_t)\right]. \tag{12}$$

At deployment, the actor relies on deployment-compatible observations and the persistent step index provided through Gate-VLM visual step verification.

Ablation Design. We ablate stepwise structure, policy step context, Gate-VLM invocation, and bidirectional progress (composition and regression). Detailed formulations are in Appendix F.

4 STEPWISE REWARDS FOR ROBOT LEARNING

Within each benchmark, all reward methods use the same PPO architecture and evaluation protocol. ***Unless noted otherwise***, we evaluate policies trained with 3 independent seeds on 50 episodes each and report their mean performance across task suites, and instantiate Frame-VLM with `gpt-5.5-2026-04-23` using `xhigh` reasoning effort (OpenAI, 2026) and Gate-VLM with `Qwen3.5-9B` (Qwen Team, 2026).

4.1 Stepwise Reward Structure

Q1. Does stepwise reward improve policy learning?

Stepwise information benefits policy learning both as **reward structure** and as **policy context**.

To test the core premise of RoboStep that *robot rewards benefit from tracking task progress step by step*, we evaluate a subset of tasks from Meta-World (Yu et al., 2019), ManiSkill (Tao et al., 2025), and SoftGym (Lin et al., 2020). We independently vary two factors: whether the reward uses an explicit stepwise structure and whether the actor and critic receive the current step index.

The *stepwise* setting follows our main method in Sec. 3.1. In the *all-dense* control, all step-local dense rewards are active simultaneously with the transition bonus removed, and the step index is used as an additional controlled policy input. All variants use the same **Frame-VLM**-generated reward components and task decomposition.

Table 1: **Effect of reward-side stepwise structure and policy-side step-index input..**

	w/ Step-Index Input				w/o Step-Index Input			
Reward	Meta-World	ManiSkill	SoftGym	Avg.	Meta-World	ManiSkill	SoftGym	Avg.
Stepwise	**92.20**	**62.00**	**49.12**	**67.77**	46.38	**44.40**	**49.83**	**46.87**
All-dense	62.85	21.60	44.00	42.82	**52.98**	20.00	47.61	40.19

As shown in Table 1, both stepwise reward routing and step-index policy context improve performance, and their combination provides even better improvement. Full task subsets, formulations and results are provided in Appendix F-1.

4.2 Roles of VLMs in Online Reward

Q2. For online rewards, should VLMs verify step completion or predict global progress?

VLMs are more effective as step-completion verifiers than as global progress predictors for online rewards.

Q1 establishes the benefit of stepwise rewards, but its transitions rely on privileged Gate-Rule predicates that are unavailable in the real world or difficult to specify for deformable and fluid tasks. We therefore evaluate whether **Gate-VLM** can serve as a practical visual transition verifier.

Table 2: **Performance on VLM roles in online reward learning across 75 manipulation tasks.**

	Benchmark Average				Task-Type Average					
Reward Method	**Meta-World**	**Mani-Skill**	**SoftGym**	**Avg.**	**Rigid**	**Artic.**	**Precise**	**Tool**	**Deform.**	**Fluid**
Benchmark-native Dense reward	35.6%	31.0%	**39.6%**	34.7%	37.1%	56.7%	7.6%	18.6%	**39.5%**	39.9%
Terminal-only Sparse reward	7.5%	3.2%	37.2%	8.8%	3.7%	11.9%	0.4%	12.0%	35.2%	41.1%
Per-subtask Sparse reward	16.5%	0.8%	7.3%	11.8%	6.6%	24.9%	6.1%	1.9%	11.0%	0.0%
SOLE-R1 (Schroeder et al., 2026)	16.3%	1.6%	6.7%	11.8%	6.7%	26.6%	3.9%	2.2%	22.0%	-23.9%
RoboMeter (Liang et al., 2026)	5.5%	1.1%	6.1%	4.4%	4.5%	8.6%	0.0%	0.0%	-0.2%	18.6%
RoboReward (Lee et al., 2026)	9.1%	5.9%	7.3%	8.1%	4.5%	16.5%	0.9%	11.7%	8.0%	6.0%
TOPReward (Chen et al., 2026c)	5.1%	0.7%	11.2%	4.5%	1.2%	10.3%	0.2%	0.4%	17.2%	-0.7%
LRM-Progress (Wu et al., 2026)	6.3%	1.8%	22.8%	6.5%	1.4%	12.0%	0.6%	5.2%	13.4%	41.6%
RoboStep (Gate-Rule)	**68.3%**	**42.6%**	31.2%	**58.8%**	**72.1%**	**77.1%**	**28.3%**	**59.3%**	27.0%	39.5%
RoboStep (Gate-VLM)	53.5%	40.3%	38.3%	49.0%	56.6%	69.2%	16.5%	51.4%	31.5%	**51.9%**

We train policies for 50 Meta-World, 19 ManiSkill and 6 SoftGym tasks and compare compatible VLM-based global progress methods under matched PPO settings. Results in Table 2 show that, across benchmarks and task categories, using VLMs as step-completion verifiers generally yields higher average success rates. Although **Gate-VLM** generally trails the privileged Gate-Rule in simulation, policies trained with **Gate-VLM** performs better on deformable and fluid tasks. This suggests that VLM verification may be particularly beneficial when subtask completion is difficult to express with distance- or pose-based rules. Task types and definitions are provided in Appendix B-1 with step decompositions in Table 24. Baseline implementation details are in Appendix C.

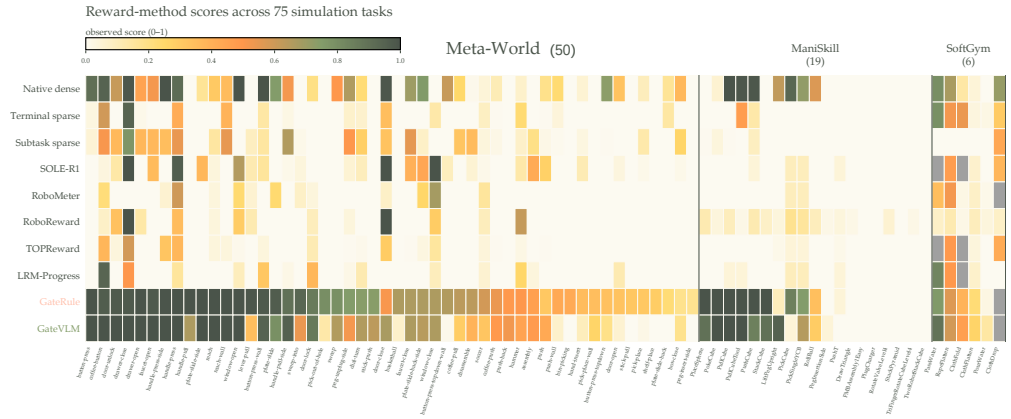


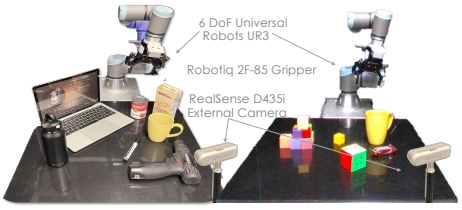
Figure 3: **Per-task performance across 75 benchmark tasks.**

4.3 Sim-to-Real Transfer

Q3. Can stepwise rewards support sim-to-real transfer?

With the persistent step index maintained by **Gate-VLM** at deployment, policies trained in reconstructed simulation can transfer back to the corresponding real scenes.

Q1 and Q2 show that the actor benefits from explicit step-index context, which can be maintained by **Gate-VLM** through visual verification. This motivates maintaining the policy step state from visual observations at deployment. We reconstruct digital twins of the target real scenes using the pipeline of prior work (Zhang et al., 2026b). During deployment, **Gate-VLM** is queried periodically to update the persistent step index supplied to the actor. The *w/o* control removes the step index only from the actor during training and deployment.



Task	Type	Task description
1.1	Articulate	*Close the computer.*
1.2	Precise	*Place the pen in the cup.*
2.1	Rigid	*Put the cube on the right into the cup.*
2.2	Rigid	*Put the rubik near the cup.*

Figure 4: **Real-world task setups for sim-to-real experiments.** Detailed task definitions and success criteria are provided in Appendix D-2

Table 3: **Sim-to-real transfer success rates (%).** *w/o* means **removing the step index only from the actor** in training and inference. Full statistics are reported in Appendix Table 14.

Task	Sim. w/	Sim. w/o	Real w/	Real w/o	Task	Sim. w/	Sim. w/o	Real w/	Real w/o
1.1	45.0	33.0	33.3	26.7	2.1	36.0	25.0	23.3	16.7
1.2	25.0	13.0	13.3	0.0	2.2	41.0	30.0	30.0	23.3

Across the four tasks, the step-conditioned actor improves average success from 63.5% to 84.0% in reconstructed simulation and from 35.8% to 61.7% in the real world. The precise insertion task remains the most challenging, and the remaining simulation-to-real gap reflects perception, calibration, and dynamics mismatch. Nevertheless, the consistent advantage of the step-conditioned actor suggests that stepwise reward design and **Gate-VLM**-maintained policy context can support direct transfer without real-robot demonstrations or real-world policy updates. See Figure 2 for the policy interfaces and Appendix D for the full protocol.

4.4 BIDIRECTIONAL TASK PROGRESSION

Q4. Can stepwise training support composition and recovery?

Explicit step-index context support **task composition** and **recovery from unseen regressions**.

A valid task progress formulation should support the representation of both *forward composition* and *backward regression*.

Forward Composition. In our experiment, Homo_n denotes n **homogeneous** macro-subtasks, each moving one object from current position to the target with three atomic steps ($K = 3n$).

Table 4: **Homogeneous forward-composition success (%).** For each Homo_n, **one end-to-end policy** executes all n **no-reset transfers**. SR completes every Gate-VLM-gated atomic step.

Composition	Homo_1	Homo_2	Homo_3	Homo_4	Homo_5	Homo_6	Homo_7	Homo_8	Homo_9	Homo_{10}
K	3	6	9	12	15	18	21	24	27	30
Success Rate	100.00	100.00	99.23	100.00	100.00	98.85	97.28	100.00	93.99	94.14

Scratch policies maintain high success through Homo_{10} with only a modest decrease after Homo_9. A clear failure boundary remains to be explored beyond Homo_{10}. **A non-stepwise control** that removes all stepwise design solves Homo_1 but fails on all substantial Homo_n. See Appendix E-1 for detailed formulation, full experiment results and an additional fixed-length **heterogeneous** study.

Backward Regression. We freeze clean-trained step-conditioned policies and inject controlled regressions only at evaluation.

Table 5: **Recovery success(%).**

	Pick	Push	Pull	YCB	**Avg.**
Forward	0	4	2	0	1.5
Gate-VLM	**64**	**92**	**92**	**56**	**76.0**
Oracle	64	94	90	58	76.5

At test time, Gate-VLM compares a valid RGB reference with the current observation to re-estimate the step index. The policy resumes execution using the corrected step-index input without any recovery-specific training. Recovery Success Rate (Recovery SR) measures task completion after disturbance. Timely Rollback Rate (TRR@10) measures whether backward update is within 10 policy steps. *Forward* uses the same actor but freezes the step index without rollback. Across our recovery tests, visual rollback achieves 76.0% Recovery SR, compared with 1.5% for forward-only, and it is also timely with an average TRR@10 of 99.5%. This indicates that with RoboStep the policy can effciently recover from unseen regressions once its persistent step context is corrected. See Appendix E-2 for the full formulation.

5 DIAGNOSTIC ABLATIONS

5.1 GATE-VLM INVOCATION CADENCE

The invocation schedule determines when semantic verification is requested. To test whether RoboStep depends on task-conditioned query timing, we replace its default invocation schedule χ_{state} with periodic Gate-VLM queries χ_{freq} and vary only the **query interval** to test performance difference.

Table 6: **Gate-VLM invocation cadence.** Values are mean success rates (%) $\pm$ standard deviation over three independently trained policies.

Benchmark	**Cadence**	**Verifier**	**Interval**	**Success Rate**
Meta-World	Frequent (fast)	Gate-VLM	16	12.73±1.46
	Frequent (mid)		64	23.60±2.16
	Frequent (slow)		128	31.40±3.34
ManiSkill	Frequent (fast)	Gate-VLM	8	31.25±3.66
	Frequent (mid)		16	43.67±3.18
	Frequent (slow)		32	38.33±1.20
SoftGym	Frequent (fast)	Gate-VLM	8	28.56±5.74
	Frequent (mid)		16	27.11±5.02
	Frequent (slow)		32	28.22±2.34

Periodic invocation supports policy learning with benchmark-dependent preferred cadences: Meta-World favors slower queries, ManiSkill favors intermediate cadences, and SoftGym is relatively insensitive. Notably, the fastest cadence is substantially worse, especially on Meta-World. This is consistent with the conclusion in Sec. 5.3: a delayed verification can be corrected later, whereas **a false acceptance** prematurely changes the step state and policy context. Thus, over-querying is more harmful than conservative querying. Full formulations are provided in Appendix F-2.

5.2 Policy Context for Stepwise Reward Learning

Q1 4.1 studies whether exposing the step index to both policy networks benefits learning. To further understand what enables effective real-world deployment, we disentangle which policy components require this context and whether privileged object and goal states can be replaced by visual observations.

When the actor is step-blind, retaining the step index in the critic improves mean success by around 10%. Replacing object and goal states with frozen visual features is substantially harder with success around 37%. Within this visual setting, removing the actor's step input has negligible effect, suggesting that the current bottleneck is visual grounding rather than symbolic step context. The latter variant further provides a deployment-compatible asymmetric actor using only visual features and robot proprioception, while privileged information is restricted to the training-time critic. Detailed formulations are provided in Appendix F-3.

Table 7: **Policy-context ablations (%).** z_t: Step index.

	Actor input	Actor z_t	Critic z_t	SR
1	State obs.	✗	✓	67.20±10.83
2	State obs.	✗	✗	58.67±2.49
3	DINOv2 + proprio	✓	✓	37.20±1.06
4	DINOv2 + proprio	✗	✓	37.07±0.96

5.3 Sensitivity to Gate Errors

A gate can fail by accepting an incomplete step (**false-acceptance rate**, FAR) or rejecting a completed step (**false-rejection rate**, FRR). For stepwise reward, We synthetically vary both error rates with the task decomposition and reward programs fixed.

The results in Figure 5 show that FAR is substantially more damaging than FRR. With FRR fixed at zero, increasing FAR from 0 to 0.30 reduces success from 94.0% to 21.8%; with FAR fixed at zero, increasing FRR to 0.30 retains 66.0% success. A false rejection preserves the current step and permits later retries, whereas a false acceptance prematurely advances the persistent step state, issues a transition bonus, and reroutes both reward and policy context. Gate operating points should therefore prioritize low on-policy FAR rather than classification accuracy alone. See Appendix F-4 for the full protocol.

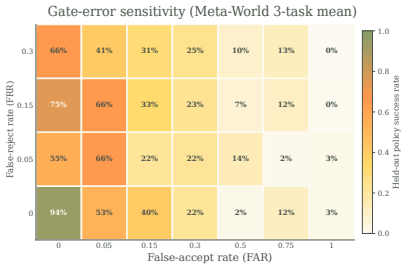


Figure 5: **Sensitivity to gate errors.** Mean held-out policy success over Meta-World tasks and three training seeds. Columns vary FAR and rows vary FRR.

6 Discussion

Our current real-world experiments do not include deformable or fluid manipulation, which would require additional simulation support. Owing to finite compute and time, the composition study currently extends only to $M = 10$, leaving empirical performance boundaries across task length and semantic diversity unresolved. Besides, future work could test whether adapting **Gate-VLM** to benchmark-specific visual domains improves step-conditioned policy learning. Broader training-seed and evaluation coverage would also better characterize optimization variability and enable false-acceptance rate and false-rejection rate to serve as metrics linking **Gate-VLM** verification quality to downstream policy performance. More broadly, we view RoboStep as one step toward understanding how robots acquire progress supervision from the physical world, how subtask understanding shapes manipulation policies, and how VLM knowledge of task structure can be transferred more effectively.

AI USE STATEMENT

We used generative AI tools to assist with English translation and grammatical editing, to organize and clarify the presentation of ideas, but not with experimental descriptions and result interpretation. All experiments and reported numerical results were produced by the authors. The authors reviewed and verified all AI-assisted content against the source materials and experimental records, made all final scientific decisions, and take full responsibility for the final content of this work.

ETHICS STATEMENT

This work does not involve human subjects or personal data. Real-robot experiments were conducted in controlled laboratory environments with task-specific safety constraints. As learned policies and visual verifiers may fail under distribution shift, we do not claim readiness for safety-critical deployment and recommend independent validation and human oversight in real-world applications.

REPRODUCIBILITY STATEMENT

We provide an anonymized implementation, including the code and reward functions used in this work at https://anonymous.4open.science/r/anonymous-code-deas. The main paper and appendices document the method formulation, policy inputs, training settings, task definitions, ablations, and evaluation protocols needed to reproduce the reported experiments.

REFERENCES

Roger Creus Castanyer, Faisal Mohamed, Pablo Samuel Castro, Cyrus Neary, and Glen Berseth. Arm-fm: Automated reward machines via foundation models for compositional reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Fred Shentu, and Philipp Wu. SARM: Stage-aware reward modeling for long horizon robot manipulation. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Qianzhong Chen, Hau Zheng, Justin Yu, Suning Huang, Jiankai Sun, Ken Goldberg, Chuan Wen, Pieter Abbeel, Yide Shentu, Philipp Wu, et al. Sarm2: Multi-task stage aware reward modeling for self improving robotic manipulation. *arXiv preprint arXiv:2606.10305*, 2026b.

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*, 2026c.

Andrew Choi, Xinjie Wang, Zhizhong Su, and Wei Xu. Scaling sim-to-real reinforcement learning for robot vlas with generative 3d worlds, 2026. URL https://arxiv.org/abs/2603.18532.

Kuan Fang, Fangchen Liu, Pieter Abbeel, and Sergey Levine. Moka: Open-world robotic manipulation through mark-based visual prompting. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.062.

Youhe Feng, Hansen Shi, Haoyang Li, Xinlei Guo, Yang Wang, Chengyang Zhang, Jinkai Zhang, Xiaohan Zhang, Jie Tang, and Jing Zhang. Procvlm: Learning procedure-grounded progress rewards for robotic manipulation, 2026. URL https://arxiv.org/abs/2605.08774.

Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. FuRL: Visual-language models as fuzzy rewards for reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14256–14274. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24j.html.

Xiaoshen Han, Junqiu Yu, Minghuan Liu, Yilun Chen, Xiaoyang Lyu, Yang Tian, Bolun Wang, Weinan Zhang, and Jiangmiao Pang. Re^{3}sim: Generating high-fidelity simulation data via 3d-photorealistic real-to-sim for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4573–4602. PMLR, 06–09 Nov 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, 2025. URL https://arxiv.org/abs/2512.16881.

Wonje Jeung, Sangyeon Yoon, Hyesoo Hong, Yoonjun Cho, Dongjae Jeon, Bumjun Kim, Jean Oh, Youngjae Yu, and Albert No. Same trajectory, contradictory rewards (ROBORMBENCH): Paraphrase fragility in vision language reward models, 2026. URL https://arxiv.org/abs/2609.05401.

Yuheng Ji, Yuyang Liu, Huajie Tan, Xuchuan Huang, Fanding Huang, et al. Prm-as-a-judge: A dense evaluation paradigm for fine-grained robotic auditing. *arXiv preprint arXiv:2603.21669*, 2026. URL https://arxiv.org/abs/2603.21669.

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

Jianxiong Li, Jinliang Zheng, Yinan Zheng, Liyuan Mao, Xiao Hu, Sijie Cheng, Haoyi Niu, Jihao Liu, Yu Liu, Jingjing Liu, Ya-Qin Zhang, and Xianyuan Zhan. DecisionNCE: Embodied multimodal representations via implicit preference learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 29461–29488. PMLR, 2024. URL https://proceedings.mlr.press/v235/li24cr.html.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023.

Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 2020.

Yuyang Liu, Yanqing Shen, Ruike Chen, Jifan Zhao, Yuxuan Tian, et al. Prm-as-a-judge 1.5: A toolkit for robot process assessment. *arXiv preprint arXiv:2608.14284*, 2026. URL https://arxiv.org/abs/2608.14284.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-image representations and rewards for robotic control. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23301–23320. PMLR, 2023a. URL https://proceedings.mlr.press/v202/ma23b.html.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv: Arxiv-2310.12931*, 2023b.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. VIP: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations*, 2023c. URL https://mlanthology.org/iclr/2023/ma2023iclr-vip/.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language model guided sim-to-real transfer. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.094.

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 *International Conference on Learning Representations*, 2025. URL https://mlanthology.org/iclr/2025/ma2025iclr-vision/.

Yiming Mao, Zixi Yu, Weixin Mao, Yinhao Li, Qirui Hu, Zihan Lan, Minzhao Zhu, and Hua Chen. Arm: Advantage reward modeling for long-horizon manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 4468–4477, June 2026.

Hexian Ni, Tao Lu, and Yinghao Cai. Core: Combined rewards with vision-language model feedback for preference-aligned reinforcement learning, 2026. URL https://arxiv.org/abs/2607.01721.

OpenAI. Introducing gpt-5.5. OpenAI, 2026. Accessed 2026.

Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, et al. Dinov2: Learning robust visual features without supervision, 2023.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.008.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, Yu Fang, Danfei Xu, Ruohan Zhang, Li Fei-Fei, Linxi Fan, Bowen Wen, Ajay Mandlekar, and Yuke Zhu. Simfoundry: Modular and automated scene generation for policy learning and evaluation. *arXiv preprint arXiv:2606.28276*, 2026.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N0I2RtD8je.

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. doi: 10.48550/arXiv.2603.28730. URL https://arxiv.org/abs/2603.28730.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Sumedh A. Sontakke, Jesse Zhang, Sébastien M. R. Arnold, Karl Pertsch, Erdem Bıyık, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. RoboCLIP: One demonstration is enough to learn robot policies. In *Advances in Neural Information Processing Systems*, volume 36, pp. 55681–55693, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/ae54ce310476218f26dd48c1626d5187-Abstract-Conference.html.

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, Shaoxuan Xie, Guocai Yao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *Robotics: Science and Systems*, 2025.

Marcel Torne Villasevil, 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 *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.015.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 2112–2121, 2018.

David Venuto, Mohammad Sami Nur Islam, Martin Klissarov, Doina Precup, Sherry Yang, and Ankit Anand. Code as reward: Empowering reinforcement learning with VLMs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 49368–49387. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/venuto24a.html.

Maggie Wang, Stephen Tian, Aiden Swann, Ola Shorinwa, Jiajun Wu, and Mac Schwager. Phys2real: Fusing vlm priors with interactive online adaptation for uncertainty-aware sim-to-real manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026a.

Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Zhihao Wang, Jianxiong Li, Yu Cui, Yuan Gao, Xianyuan Zhan, Junzhi Yu, and Xiao Ma. World value models for robotic manipulation, 2026b. URL https://arxiv.org/abs/2606.24742.

Yanru Wu, Weiduo Yuan, Ang Qi, Vitor Guizilini, Jiageng Mao, and Yue Wang. Large reward models: Generalizable online robot reward generation with vision-language models. *arXiv preprint arXiv:2603.16065*, 2026.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tUM39YTRxH.

Qinwen Xu, Jiaming Liu, Rui Zhou, Shaojun Shi, Nuowei Han, Zhuoyang Liu, Chenyang Gu, Shuo Gu, Yang Yue, Gao Huang, et al. Twinrl-vla: Digital twin-driven reinforcement learning for real-world robotic manipulation. *arXiv preprint arXiv:2602.09023*, 2026a.

Yijie Xu, Haopeng Jin, Run Zhou, Shengbang Liu, Sixiang Chen, Hongyang Cheng, Sicheng Hu, Peterson Co, Jinwen Luo, Huajie Tan, and Shanghang Zhang. Robo-Dopamine 2.0: History-conditioned and ood-aware process reward modeling for robotic manipulation, 2026b. URL https://arxiv.org/abs/2608.15680.

Yang Yang, Bingjie Chen, Zihan Wang, Yizhe Li, Guoping Pan, Yi Cheng, and Houde Liu. Stage-Transition Dense Reward Modeling for Reinforcement Learning, 2026. arXiv:2606.31377v1.

Silong Yong, Stephen Sheng, Carl Qi, Xiaojie Wang, Evan Sheehan, Anurag Shivaprasad, Yaqi Xie, Katia Sycara, and Yesh Dattatreya. Generalizable dense reward for long-horizon robotic tasks. 2026. URL https://arxiv.org/abs/2604.00055.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 374–404. PMLR, 2023. URL https://proceedings.mlr.press/v229/yu23a.html.

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Liangzhi Shi, Yuqing Xie, Zhexuan Xu, Zhihao Liu, Kang Chen, Wenhao Tang, Quanlu Zhang, Weinan Zhang, Chao Yu, and Yu Wang. Rlinf-vla: A unified and efficient framework for vla+rl training, 2025. URL https://arxiv.org/abs/2510.06710.

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. RewiND: Language-guided rewards teach robot policies without new demonstrations. In *9th Annual Conference on Robot Learning*, 2025a. URL https://openreview.net/forum?id=XjjXLxfPou.

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. doi: 10.48550/arXiv.2607.21655.

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 *Forty-third International Conference on Machine Learning*.

Shujie Zhang, Jingkun Yi, Weipeng Zhong, Zirui Zhou, Yangkun Zhu, Hanqing Wang, Xudong Xu, Weinan Zhang, and Chunhua Shen. Robosnap: One-shot real-to-sim scene generation for generalizable robot learning and evaluation, 2026b. URL https://arxiv.org/abs/2607.06699.

Xiuhui Zhang, Ning Gao, Xingyu Jiang, Yihui Chen, Yuheng Pan, Mohan Zhang, and Yue Deng. Progress Reward Model for Reinforcement Learning via Large Language Models. In *Advances in Neural Information Processing Systems*, 2025b. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/91b047c5f5bd41ef56bfaf4ad0bd19e3-Abstract-Conference.html.

Siheng Zhao, Jiageng Mao, Wei Chow, Zeyu Shangguan, Tianheng Shi, Rong Xue, Yuxi Zheng, Yijia Weng, Yang You, Daniel Seita, et al. Robot learning from any images. In *Conference on Robot Learning*, pp. 4226–4245. PMLR, 2025.

Xunlan Zhou, Xuanlin Chen, Shaowei Zhang, Xiangkun Li, ShengHua Wan, Xiaohai Hu, Yuan Lei, Le Gan, and De chuan Zhan. Marvl: Multi-stage guidance for robotic manipulation via vision-language models, 2026. URL https://arxiv.org/abs/2602.15872.

APPENDIX TABLE OF CONTENTS

A RoboStep Method Details 16

A-1 Reflective Reward Compilation 16
A-2 Detailed Formulation 17
A-3 Frame-VLM and Gate-VLM Inputs 17
A-4 Reward and Subtask Example 19

B Benchmark Experimental Protocols 20

B-1 Benchmark Configurations, Task Definitions and Success Criteria 20
B-2 Additional Experimental Results 24

C Visual Reward Model Experimental Protocols 25

C-1 Baseline Selection 25
C-2 Baseline Implementations 25
C-3 Training Cost and Reward-Query Efficiency 26

D Real-to-Sim-to-Real Experiment Details 27

D-1 Policy Inputs and Deployment Protocol 27
D-2 Scene Reconstruction and Calibration 27
D-3 Task and Success Criteria 27
D-4 Training and Evaluation Protocol 28
D-5 Results 28

E Bidirectional Task Progress 29

E-1 Forward: Composable Stepwise Rewards 29
E-2 Backward: Regression and Recovery 31

F Additional Diagnostic Ablations 34

F-1 Stepwise Reward Structure 34
F-2 **Gate-VLM** Invocation Cadence 36
F-3 Policy Context for Stepwise Reward Learning 37
F-4 Sensitivity to Gate Errors 38

G Full Related Work 39

G-1 Full Related Work Table 39
G-2 Full Related Work Task Table 39

A ROBOSTEP METHOD DETAILS

A-1 REFLECTIVE REWARD COMPILATION

For each task across the benchmarks we tested, if the development validation success remains below 50%, **Frame-VLM** receives the task specification, simulator-state interface, current reward program, and training diagnostics, and proposes one reward/stage repair while the PPO policy-learning protocol is kept fixed. **The procedure is limited to 5 compilation rounds**, after which the round-five program is frozen for the main-table evaluation.

A-2 Detailed Formulation

State-side invocation trigger. The invocation rule χ in Eq. 18 determines only when semantic verification is requested. In the reported simulation experiments, χ_t is instantiated using state/API-based transition candidates. For this simulation-specific instantiation, **Frame-VLM** additionally generates a candidate predicate $c_i : O_b \to \{0, 1\}$ and a consecutive-hold requirement H_i for each step i.

For the active step, $c_i(o_t^{(b)})$ evaluates whether the current state satisfies the physical conditions for requesting a possible transition. To suppress transient fluctuations, we maintain

$$q_i(t+1) = \begin{cases} q_i(t) + 1, & c_i(o_t^{(b)}) = 1, \\ 0, & c_i(o_t^{(b)}) = 0. \end{cases} \tag{13}$$

The corresponding invocation trigger is

$$\chi_t^{\text{state}} = \mathbf{1}[z_t < K]\, \mathbf{1}\left[q_{z_t}(t) \geq H_{z_t}\right], \tag{14}$$

and we set $\chi_t = \chi_t^{\text{state}}$ for the reported simulation experiments. When $\chi_t = 1$, the current RGB packet $\mathcal{I}_t$ is sent to **Gate-VLM** for semantic verification:

$$v_t = \mathbf{1}\left[\textbf{Gate-VLM}\left(\ell, z_t, \mathcal{V}_{z_t}, \mathcal{I}_t\right) = \texttt{success}\right], \qquad e_t = \chi_t v_t, \qquad z_{t+1} = z_t + e_t. \tag{15}$$

When no query is issued, or when **Gate-VLM** returns failure or uncertainty, $v_t = 0$. Thus, the state/API trigger determines only when visual verification is requested, while **Gate-VLM** provides the semantic transition authority.

Deployment-compatible triggers. The state-side trigger above relies on simulator-accessible quantities, but the verification interface itself is trigger-agnostic. When such state/API candidates are unavailable, χ_t can instead be instantiated as a periodic trigger:

$$\chi_t^{\text{freq}} = \mathbf{1}[z_t < K]\, \mathbf{1}[t \bmod F_b = 0], \tag{16}$$

where F_b is the query interval measured in the native policy steps of benchmark b. We evaluate this frequency-based instantiation and its benchmark-specific query cadences in Appendix F-2.

A possible fully visual alternative is to invoke verification after a sufficient change in a frozen RGB representation:

$$\chi_t^{\text{vis}} = \mathbf{1}[z_t < K]\, \mathbf{1}\left[\|f(I_t) - f(I_{t-\Delta})\|_2 > \eta\right], \tag{17}$$

where $f(\cdot)$ is a frozen visual encoder. We include this only as a deployment-compatible trigger option; it is not evaluated in the reported experiments.

A-3 Frame-VLM and Gate-VLM Inputs

We use two VLM components with distinct computational roles. **Frame-VLM** is used only during offline reward compilation and repair, where it must decompose the task, reason over the declared state/API interface, and generate executable stepwise reward logic. Since its inference cost is incurred before policy optimization and amortized over the entire training run, we prioritize reasoning and code-generation capability over latency. Unless otherwise stated, we instantiate **Frame-VLM** with `gpt-5.5-2026-04-23` using `xhigh` reasoning effort (OpenAI, 2026).

Gate-VLM, in contrast, lies on the online transition path. In the reported simulation experiments, it is queried only when the state-side candidate and its dwell requirement trigger a verification request; when simulator state is unavailable, the same interface can instead use periodic or visual-change

invocation. The trigger controls only when verification is requested and does not itself authorize a transition. Unless otherwise stated, we instantiate **Gate-VLM** with `Qwen/Qwen3.5-9B` (Qwen Team, 2026), prioritizing multimodal inference efficiency over the more expensive reasoning used for offline reward compilation.

Because the latest verified step state is held between queries, **Gate-VLM** does not need to operate at the policy or low-level controller frequency. Instead, each sparse online query must process the prescribed RGB packet and return exactly one parseable decision from {`success`, `failure`, `unknown`} before the corresponding request becomes stale. Invalid, unparsable, and `unknown` outputs fail closed and never advance the persistent step state. The verifier receives only the specified visual-language evidence, rather than simulator state, reward values, or evaluation labels.

Except for the experiments where the VLMs become part of the variant factors, all of RoboStep experiments follows the above selection.

The prompts below show the reusable templates; task-specific content is inserted into the bracketed fields.

Frame-VLM Prompt

```
You are designing a stepwise reward program
for a robot manipulation task.

Task description: {task_description}
Scene and goal conventions: {scene_and_goal_description}
Available state APIs: {available_state_apis}
Design constraints: {design_constraints}

Decompose the task into feasible,
ordered stages. For each stage, specify:
- its semantic objective and prerequisites;
- a bounded dense progress potential using only the available states;
- a persistent candidate-completion predicate and dwell requirement;
- a bounded, one-time transition bonus;
- maintenance and safety conditions that prevent shortcut solutions;
- visible evidence that would allow an RGB-only verifier to
check completion.

Do not invent unavailable observations. Distinguish progress toward
a stage from completion of that stage.

Return the stage definitions and executable
reward logic in the required format: {output_schema}.
```

Frame-VLM receives the task specification, scene conventions, available state/API interface, and task-level design constraints. Its outputs

$$\mathcal{R}(\ell, O_b) = \left(\{(\phi_i, M_i, b_i, \mathcal{V}_i)\}_{i=0}^{K-1}, \chi\right). \tag{18}$$

as defined in Sec. 3.

Gate-VLM Prompt

```
You are a visual verifier of robot manipulation stage completion.

Task: {task_description}
Current stage: {stage_id_and_name}
Visible completion condition: {visible_completion_contract}
Evidence: {episode_start_image}, {ordered_recent_frames},
          {current_frozen_image}

A fast state-side predicate has proposed a transition candidate.
This is not a completion label. Judge only whether the CURRENT stage
is visibly complete in the current image, using earlier images to
```

```
establish identity and motion.
Do not require a later stage. Reject a visibly wrong object,
wrong target, robot-only motion when object motion is required,
loss of a required relation, or other contradictory evidence.

Do not infer precise hidden distances, forces, contacts,
or benchmark scores from RGB.

Return exactly one decision from {allowed_outputs}.
```

At each candidate boundary, **Gate-VLM** receives the task description, current stage, its visible completion contract v_i, and RGB evidence consisting of the episode-start image, optional temporally ordered recent frames, and the current frozen image. In the two-frame implementation, {ordered_recent_frames} is empty.

The response is parsed as exactly one element of

$$\hat{g}_i \in \{\texttt{success}, \texttt{failure}, \texttt{unknown}\}.$$

No scalar progress value is parsed from **Gate-VLM**. A forward transition is authorized only when the persistent state-side candidate and visual verification agree:

$$c_i \text{ sustained for } H_i \quad \wedge \quad \hat{g}_i = \texttt{success} \quad \Longrightarrow \quad z_t \to z_t + 1.$$

Outputs `failure` and `unknown` do not authorize the transition. **Gate-VLM** therefore verifies stage completion without modifying the dense potential generated by **Frame-VLM**.

A-4 Reward and Subtask Example

We show one representative reward program from each benchmark. Each program is generated from the task description and the benchmark state interface, and follows

$$\mathcal{R}(\ell, O_b) = \{(\phi_i, M_i, c_i, H_i, b_i, \mathcal{V}_i)\}_{i=0}^{K-1}.$$

Meta-World

Task: `stick-push-v3`.
The generated subtasks are:

approach the stick $\to$ capture and lift the stick $\to$ contact the secondary object $\to$ push the object to the target.

For $K = 4$, the stage potentials are summarized as

$$\begin{aligned}
\phi_0 &= 0.9q_{\text{pre}} + 0.1q_{\text{open}}, \\
\phi_1 &= 0.25q_{\text{grasp}} + 0.30q_{\text{held}} + 0.45q_{\text{lift}}, \\
\phi_2 &= \sqrt{q_{\text{contact}} q_{\text{held}}}, \\
\phi_3 &= 0.7q_{\text{outcome}} + 0.2q_{\text{contact}} + 0.1q_{\text{held}}.
\end{aligned}$$

The candidates c_i test pre-grasp, held-and-lifted, tool–object contact, and object–target completion. The program uses

$$H = (2, 2, 2, 1), \qquad b = (16, 70, 230, 320),$$

with $\mathcal{V}_i$ containing TCP, stick, object, target, gripper, and relative-distance features. M_i retains the relevant grasp, contact, and outcome conditions.

ManiSkill

Task: `PegInsertionSide-v1`.
The generated subtasks are:

reach the peg $\to$ grasp the peg $\to$ align the peg with the hole $\to$ insert the peg.

For $K = 4$, the state-based program uses

$$\begin{aligned}\phi_0 &= q_{\text{reach}},\\ \phi_1 &= 0.65g + 0.35q_{\text{reach}},\\ \phi_2 &= 0.75q_{\text{align}} + 0.25g,\\ \phi_3 &= 0.80q_{\text{insert}} + 0.15q_{\text{align}} + 0.05\,\text{success}.\end{aligned}$$

The candidates c_i test TCP–peg proximity, grasping, hole alignment, and official success. The program uses

$$H = (2, 2, 2, 2), \qquad b = (0.20, 0.47, 0.73, 1.00),$$

with $\mathcal{V}_i$ containing TCP, peg and peg-head poses, hole and goal poses, grasp state, alignment error, and success. M_i maintains grasp, alignment, and insertion quality.

SoftGym

Task: `PassWater`.
The generated subtasks are:

transport the cup while retaining water → deliver water near the target.

For $K = 2$, let m denote normalized cup transport, q normalized distance to the target, and ρ retained-water fraction. The generated potentials are

$$\phi_0 = [0.80m\rho + 0.20\rho]_{[0,1]}, \qquad \phi_1 = [0.70q\rho + 0.30\rho]_{[0,1]}.$$

The candidates are

$$c_0 : \phi_0 \geq 0.35, \qquad c_1 : \phi_1 \geq 0.75,$$

with

$$H = (2, 2), \qquad b = (1.5, 1.5).$$

The state set $\mathcal{V}_i$ contains cup and target positions, water particle positions, glass geometry, and containment indicators. The retention term is included directly in the potentials, so $M_i = 0$ in the clean recipe. The later spill-aware version preserves this structure while adding stricter rejection of empty cups and major spills.

B Benchmark Experimental Protocols

Q1 in Sec. 4.1 shows that an explicit stepwise reward structure and step-index context substantially improve policy learning over conventional dense rewards. However, the step transitions in Q1 are provided by privileged Gate-Rule predicates. Such predicates can be difficult to specify outside simulation, particularly for tasks whose intermediate completion cannot be reliably characterized by simple state, distance, or pose thresholds, such as deformable-object manipulation. We therefore replace rule-based transition detection with visual-language verification, **Gate-VLM**, and investigate whether VLMs are better used as global visual progress predictors or as semantic step-completion verifiers for online reward.

B-1 Benchmark Configurations, Task Definitions and Success Criteria

Across the 75 tasks, the number of active steps ranges from 2 to 7. Meta-World tasks use 3–5 steps, while all six SoftGym tasks use two. The largest decomposition is `StackPyramid-v1` in ManiSkill, with seven active steps and therefore an eight-dimensional step encoding including the terminal state. Full task definitions and reward decompositions are provided in Appendix B.

B-1.1 Benchmark Comparison Configurations

For all benchmark comparisons, we match the policy-learning setup within each task and vary only the reward and transition-verification mechanism under study. Specifically, the environment version, reset distribution, action and controller interface, actor–critic architecture, PPO optimization protocol, rollout budget, random seeds, checkpoint-selection rule, and evaluation protocol are held fixed

across reward methods. The benchmark-specific network architectures and PPO hyperparameters used in these matched comparisons are summarized in Table 8.

Table 8: **Matched PPO policy-learning configurations.** Configurations are fixed across reward methods within each benchmark. Per task-specific training budgets and according reward functions are provided in Appendix B-1.2.

Benchmark	Actor/Critic	Rollout	Epochs / Mini.	LR / Schedule	γ	λ	Clip
Meta-World	256–256–128, ELU	32×64	$8/8$	Adaptive KL	.99	.95	.20
ManiSkill	3×256, Tanh	16×50	$4/32$	3×10^{-4}	.80	.90	.20
SoftGym	256–256–128, ELU	256 steps/update	$4/4$	3×10^{-4}	.99	.95	.20

These policy backbones are benchmark-specific controls rather than components optimized for our reward method. For ManiSkill, we follow the three-layer, 256-unit Tanh actor–critic and core PPO hyperparameters of its official PPO example. For Meta-World and SoftGym, we use a 256–256–128 ELU actor–critic within each benchmark. We hold policy capacity fixed within a task to isolate the effect of reward design and transition verification.

As detailed in Table 9, the main comparisons use each benchmark's native state observation together with the active-step code for both actor and critic, while RGB observations are provided only to visual reward or transition-verification modules and are never concatenated with the policy input. For a fixed task, the policy observation layout and step-code width are therefore shared across compared reward methods; variants that remove or hide the step code, or replace the state-based actor with visual features, are evaluated separately as input ablations.

Table 9: **Policy-input protocol used in the main reward comparison.** For each task, all reward methods share the same policy observation and PPO configuration.

Bench	Benchmark-native state x_t	Policy observation (actor / critic)	Visual reward / gate observation
Meta-World	Fully observable Sawyer state: current hand/TCP position, gripper opening, two object-pose slots, previous state, and task goal. *Dim.:* $39 = 18_{\text{cur}}+18_{\text{prev}}+3_{\text{goal}}$.	$\tilde{o}_t = [x_t; e(s_t)] \in \mathbb{R}^{39+K+1}$. Across reported tasks, the policy input has **41–45 dimensions**.	Fixed-camera RGB snapshots or temporal packets are used only by the visual reward model or **Gate-VLM** transition verifier and are never provided to the policy.
ManiSkill	Flattened `obs_mode=state`: robot proprioception and end-effector state, together with task-specific object, goal, and interaction states. *Dim.:* $d_\tau = 31$–76 across the 19 tasks.	$\tilde{o}_t = [x_t; e(s_t)] \in \mathbb{R}^{d_\tau+K+1}$. Across reported tasks, the policy input has **36–82 dimensions**.	Rendered RGB observations are queried only by the visual reward or step-verification module; the PPO actor and critic remain state-based.
SoftGym	Task-native `key_point` state: cloth: four corners + two picker positions (**18-D**); rope: ten keypoints + two picker positions (**36-D**); PassWater: cup/retained-water state (**7-D**); PourWater: two-cup/water state (**13-D**).	All six tasks use $K = 2$: $\tilde{o}_t = [x_t; e(s_t)] \in \mathbb{R}^{d_\tau+3}$. Input dimensions are **21-D** for cloth, **39-D** for rope, **10-D** for PassWater, and **16-D** for PourWater.	320×320 RGB frames and their temporal buffer are consumed only by the visual reward / gate module and are never provided to the PPO networks.

$e(s_t) \in \{0,1\}^{K+1}$ denotes the one-hot active-step code, including an additional absorbing/completed state. For a fixed task, all main-table methods use the same benchmark-native state, action space, PPO architecture, actor/critic observation tensor, step-code width, and controller interface. Reward-model images are never concatenated with the policy input. Removing the stage code, hiding it from the actor, or replacing the state actor with an RGB encoder is evaluated separately as an input ablation.

B-1.2 Observations and Stepwise Reward of Each Task

We separate the information available to four components: the policy, critic, **Frame-VLM** reward program, and the live VLM verifier. The policy and critic receive the benchmark training observation together with the persistent active stage, whereas the reward program may additionally use task-relevant simulator state to compute the dense potential and candidate transition predicates. The live VLM verifier is kept on the visual side of this boundary: simulator state can schedule a candidate verification, but the verifier itself judges visual evidence rather than receiving the privileged geometric quantities used by the reward program. Table 24 gives the task-level state/API inputs used for dense-reward construction, together with the stage decomposition and dense-potential provenance.

MetaWorld. The actor receives the standard 39-dimensional MetaWorld state concatenated with a $(K+1)$-way one-hot code for the Qwen-authorized persistent step; the critic uses the same observation. The selected PPO runs do not use the native task reward. Instead, the generated reward programs use task-dependent geometric quantities such as TCP, object/control, target, and initial poses, gripper opening, and, where required, object/tool orientation. A state rule only proposes a transition. The VLM verifier receives the episode-start reference image, a time-ordered ring of recent RGB frames, the frozen current camera frame, and the semantic instruction for the active step; metric distances and the native success flag are not included in the verifier packet.

ManiSkill. The selected policies and critics use the simulator state concatenated with the active-stage one-hot code; the selected runs set the RGB actor input to false. The reward backend is the generated stepwise rule program. Its task-specific state fields are listed in Table 24 and include, for example, poses, geometric errors, grasp/contact indicators, velocities, and, where configured for final-step logic, simulator success signals. The VLM step observer is evaluated from rendered RGB evidence using step-specific visible-completion descriptions, rather than by directly exposing these reward-program state fields to the verifier.

SoftGym. All six tasks use the benchmark's key-point observation mode. The step wrapper concatenates the persistent step one-hot code to the base policy observation and records the step code as available to both actor and critic. Dense rewards are computed from task-language-compatible simulator APIs, including PyFlex particle positions, cloth/rope metadata, cup poses, water particles, target geometry, and geometric containment tests. SoftGym's official `normalized_performance` is excluded from reward and transition decisions. For Qwen verification, the wrapper renders 320×320 RGB frames and supplies a reset reference, recent frame ring, frozen current frame, and the active-step instruction.

Across the full suite, we obtain the step statistics summarized in Table 10. The absorbing post-completion state is excluded from K, so each task contains exactly $K-1$ ordered transitions. `StackPyramid-v1` is the only seven-step task. The absorbing post-completion state is not counted in K, and therefore each task contains exactly $K-1$ ordered transitions.

Benchmark	Tasks	Steps	Internal Trans.	Mean Steps	Step Range
Meta-World	50	172	122	3.440	1–5
ManiSkill	19	81	62	4.263	3–7
SoftGym	6	12	6	2.000	2
Total	**75**	**265**	**190**	**3.533**	**1–7**

Table 10: **Step-decomposition statistics across the 75 benchmark tasks.** Internal transitions count boundaries between consecutive non-terminal steps and exclude transition into the terminal state.

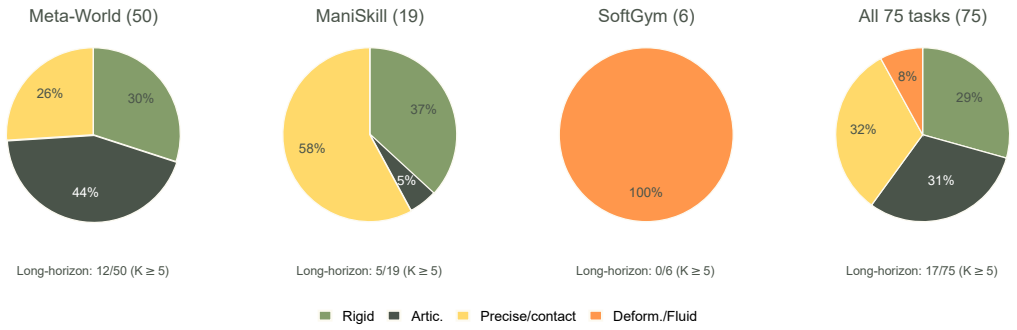


Figure 6: **Task-type distribution across the evaluated benchmarks.** Each pie shows the mutually exclusive semantic categories *Rigid*, *Artic.*, *Precise/contact*, and *Deform./Fluid*. The accompanying *Long-horizon* annotation reports tasks with at least five steps ($K \geq 5$) and may overlap with the four semantic categories. See more about task information in Table 22 and 23 and see task completion visualization in Figure 7.

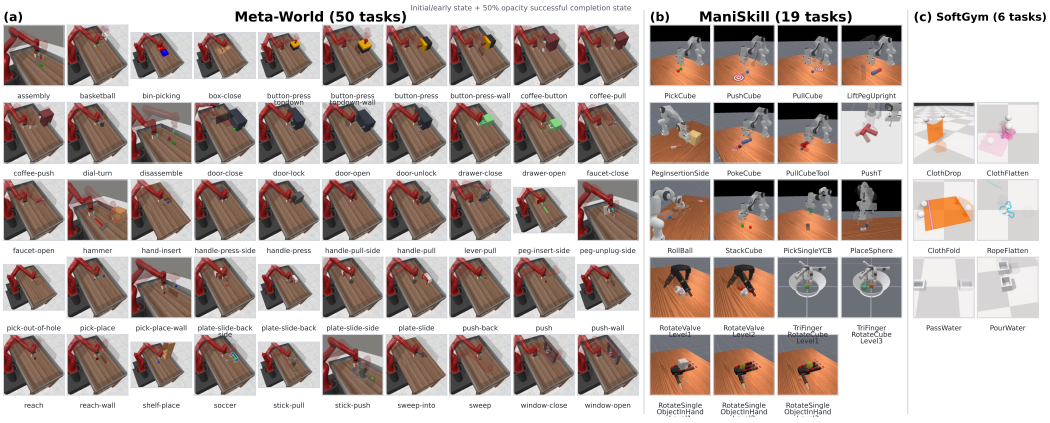


Figure 7: **Overview of benchmark tasks.** Task scene demonstrations from MetaWorld, ManiSkill, and SoftGym. Each overlaid visualization shows both the initial and final task states.

B-2 Additional Experimental Results

B-2.1 Evaluation Protocols

For the completed Meta-World RoboStep (Gate-Rule and Gate-VLM) runs, checkpoints were ranked on independent reset seed 2042 and the selected checkpoint was evaluated on held-out reset seed 1042. All 50 tasks have three independent **training seeds** (42, 43, and 44).

For each completed ManiSkill held-out retest or new reward-control run, we select by `success_once` on 20 seed-2042 validation episodes, and test the selected checkpoint on 50 seed-1042 episodes. In SoftGym, the selected Gate-Rule runs use disjoint validation and test configurations: all six tasks use normalized performance.

B-2.2 Per-Task Learning Curves

To complement the aggregate results, we visualize representative reward traces over complete task rollouts in Figure 8. These examples illustrate how the learned reward evolves within and across steps under both successful and failed executions.

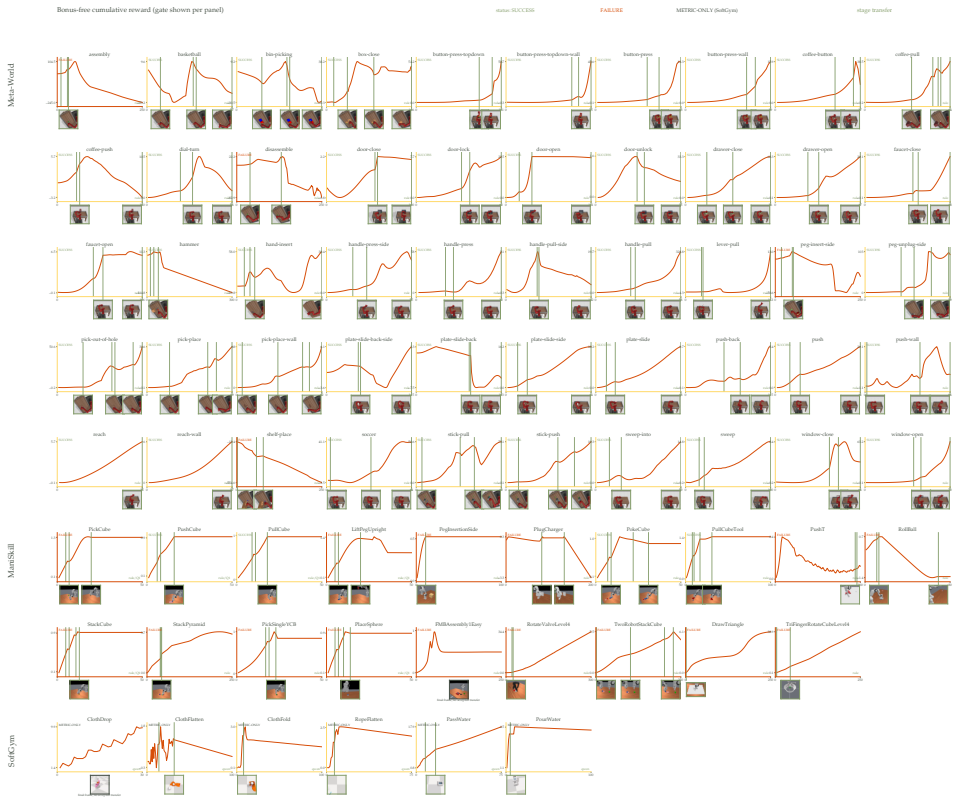


Figure 8: **Illustrative reward traces and step transitions.** We show one saved rollout per task for visualization. The examples include both successful and failed trajectories. Green and orange annotations indicate success and failure when the benchmark exposes a binary success signal, while SoftGym is marked metric-only because it has no shared binary success API. Vertical lines denote accepted step transitions.

C Visual Reward Model Experimental Protocols

C-1 Baseline Selection

We select baselines that can provide an online visual reward signal under the same downstream policy-learning interface. Not all related methods are directly compatible with this setting. For example, RL-VLM-F (Wang et al., 2024) is tightly coupled to an off-policy PEBBLE/SAC loop and task-specific SoftGym initialization; ReWiND (Zhang et al., 2025a) relies on demonstration-based pretraining and an external success signal; while STDR (Yang et al., 2026) and MARVL (Zhou et al., 2026) require task-specific reward construction from expert demonstrations or per-stage target images, respectively.

For reproduced visual-reward baselines, we regard the released VLM checkpoint, prompt, visual input construction, score normalization, and temporal reward adapter as part of the baseline itself. We therefore follow the backbone and reward interface released or reported by each method rather than replacing all baselines with a common VLM.

Table 11: **Original task coverage of reference-only baselines.** We list the tasks reported in the original papers and categorize them using the same primary interaction taxonomy as Table 2.

Method	# Tasks	Original Tasks	Type Mix
RL-VLM-F (Wang et al., 2024)	7	**Meta-World:** Open Drawer, Soccer, Sweep Into. **SoftGym:** Fold Cloth, Straighten Rope, Pass Water. **OpenAI Gym:** CartPole.	Rigid 2; Artic. 1; Deform. 2; Fluid 1; non-manip. 1.
ReWiND (Zhang et al., 2025a)	13	**Meta-World:** Window Close, Reach Wall, Faucet Close, Coffee Button, Button Press Wall, Door Lock, Handle Press Side, Sweep Into. **Real:** Separate blue/orange cups, Fold blue towel, Open red trash bin, Put orange cup on red plate, Put fruit-colored object in box.	Rigid 5; Artic. 5; Precise 2; Deform. 1.
STDR (Yang et al., 2026)	17	**Meta-World:** Button Press, Window Open, Reach Wall, Plate Slide, Push, Coffee Push, Stick Push, Pick Place. **ManiSkill:** PullCube, PushCube, PickCube. **Franka Kitchen:** Slide Cabinet, Kettle, Microwave. **Real:** PutPepperInBucket, PutToyOnPlate, ReachAndPullDrawer.	Rigid 11; Artic. 4; Precise 1; Tool 1.

Table 12: **Original evaluation coverage of reference-only baselines.** We categorize the policy-learning tasks reported in the original papers using the same primary interaction taxonomy as Table 2. Counts exclude real-world reward-only evaluation and non-manipulation tasks.

Method	Meta-World	ManiSkill	SoftGym	Rigid	Artic.	Precise	Tool	Deform./Fluid	Original reporting
RL-VLM-F (Wang et al., 2024)	3	0	3	2	1	0	0	2 / 1	MW: success rate; others: episode return
ReWiND (Zhang et al., 2025a)	8	0	0	2	4	2	0	0 / 0	79% IQM SR at 100k steps
STDR (Yang et al., 2026)	8	3	0	9	3	1	1	0 / 0	Per-task learning curves; MS table reports SR

C-2 Baseline Implementations

Evaluation Protocol. Within each benchmark and task, all reward methods use the same environment version, reset distribution, policy observation, action and controller interface, actor–critic architecture, PPO configuration, rollout budget, episode horizon, training seeds, and checkpoint cadence. Only the reward source and the visual inputs required by that reward method are changed.

Checkpoints are selected using an independent validation reset set and evaluated on a held-out reset set using the benchmark-native evaluation metric. For Meta-World and ManiSkill, we use 20 validation episodes with reset seed 2042 and 50 held-out episodes with reset seed 1042. SoftGym uses disjoint validation and test configurations and reports its native normalized-performance metric. Unless otherwise stated, results are aggregated over three independent training seeds. Missing or incomplete evaluations are reported as unavailable rather than being converted to zero.

Matched Comparison with Visual Reward Baselines. Within each task, we keep the environment version, reset distribution, action and controller interface, actor–critic architecture, PPO opti-

mization protocol, rollout budget, random seeds, checkpoint-selection rule, and evaluation protocol fixed across methods. The intentional differences are restricted to the reward formulation and any additional input required by a particular method. Apart from these method-specific inputs, all policy observations follow the standard observation setting of the corresponding benchmark.

For global-progress baselines, we follow their released or reported reward protocols as closely as possible, including their prescribed VLM backbone and additional model inputs. We treat the VLM choice as part of each reward method rather than forcing all methods to share a single backbone. In RoboStep, the reward components are constructed offline by **Frame-VLM** following the shared core formulation, while **Gate-VLM** is queried online only for semantic step-transition verification, rather than for direct prediction of a global scalar progress reward.

Baseline Reward Interfaces. The benchmark-native dense control directly uses the benchmark reward, while the terminal-only and per-subtask sparse controls retain only terminal or subtask-completion events, respectively.

For the visual baselines, we preserve their method-specific temporal reward interfaces. SOLE-R1 uses its released absolute-progress predictor on sparse visual observations; in the main comparison we use the same fixed $q = 64$, reward scale 0.05, and no progress clipping across tasks, and linearly interpolate sparse predictions over the rollout before PPO updates. RoboMeter uses its reward head on a bounded video history queried every 10 environment steps and holds the latest absolute progress estimate between queries. RoboReward similarly uses its discrete 1–5 trajectory score, mapped to $[0, 1]$ by $(s - 1)/4$, with a $q = 10$ video buffer and reward holding. TOPReward obtains a scalar completion score from VLM token probabilities on cumulative visual evidence and interpolates sparse $q = 10$ predictions over the rollout. LRM-Progress uses the released absolute-progress endpoint with $q = 10$ and holds the most recent prediction between queries. Visual reward inputs are never concatenated with the policy observation.

Inference-Side Calibration. Visual progress methods expose additional inference-side choices such as query interval, video history, score scaling or clipping, and interpolation or holding between sparse predictions. We use released settings whenever available and otherwise restrict selection to these reward-interface parameters using validation episodes only; policy architecture and PPO optimization remain unchanged.

SOLE-R1 is particularly sensitive to this temporal adapter. We therefore also evaluate task-specific query/scale/clip candidates as a separate diagnostic. These configurations are selected only from validation performance and are reported separately; the main Meta-World comparison uses the single frozen $q = 64$ configuration above. Thus, task-specific reward-interface tuning is not used to construct the main aggregate result.

C-3 Training Cost and Reward-Query Efficiency

Frame-VLM is invoked offline during task compilation and therefore introduces no reward-model inference in the policy rollout. During online learning, **Gate-VLM** is queried only when the transition trigger requests semantic verification; the verified step state is then retained between queries.

In contrast, global-progress baselines repeatedly rescore visual observations or video prefixes throughout the rollout. For a fixed-cadence method with episode horizon H and query interval q, the number of reward-model query events scales approximately as $\lceil H/q \rceil$ per environment episode before batching. In our main implementations, SOLE-R1 uses $q = 64$, while LRM-Progress, RoboMeter, RoboReward, and TOPReward use $q = 10$ according to their corresponding reward adapters. We preserve these method-specific cadences rather than artificially matching query counts, since changing the temporal adapter changes the reward definition itself. All supported visual queries are batched, and offline **Frame-VLM** compilation is accounted for separately from online reward-query cost.

D REAL-TO-SIM-TO-REAL EXPERIMENT DETAILS

D-1 POLICY INPUTS AND DEPLOYMENT PROTOCOL

The generic actor–critic interface is defined in Section 3 and Figure 2. Here we specify its Real-to-Sim-to-Real instantiation. The actor receives deployment-compatible camera-derived perception features, robot proprioception and controller state, and the persistent step one-hot $e(z_t)$. The critic additionally receives simulator-only task, contact, and safety states during PPO training and is discarded at deployment.

For the actor-blind control, we remove only $e(z_t)$ from the actor input:

$$x_{t,\mathrm{w/o}}^{\mathrm{actor}} = \left[f_{\mathrm{vis}}(I_t); x_t^{\mathrm{robot}} \right]. \tag{19}$$

The reward program, transition mechanism, critic input, PPO architecture, training budget, and checkpoint-selection protocol remain unchanged. At deployment, **Gate-VLM** is queried periodically to update z_t, and the resulting step context is supplied only to the step-conditioned actor.

D-2 SCENE RECONSTRUCTION AND CALIBRATION

We reconstruct task-specific digital twins by aligning the robot base, objects, articulated assets, external-camera calibration, and initial robot configuration with the corresponding real scenes. Policy training and checkpoint selection use only the reconstructed simulation. **No real-robot demonstrations, pretrained policy or real-world policy updates are used**.

The simulator we include is NVIDIA Isaac Sim with PhysX. Physics runs at 120 Hz, while the policy runs at 10 Hz with 12 physics substeps per action. The robot is a UR3 with a Robotiq gripper, and the four-dimensional action consists of normalized Cartesian end-effector displacement and an absolute gripper command. End-effector orientation is fixed by the controller rather than directly predicted by the actor. The external camera, robot base, and laptop articulation are calibrated in the reconstructed scene.

The actor uses deployment-compatible robot and controller state together with a camera-derived screen-handle displacement estimate. During simulation training, this estimate is perturbed with 2 mm visual position noise. Simulator-only hinge, contact, safety, and success quantities are restricted to the critic, reward computation, and evaluation.



Figure 9: **Objects used in our sim-to-real transfer experiments.**

D-3 TASK AND SUCCESS CRITERIA

We evaluate sim-to-real transfer across four real-world manipulation tasks spanning articulated, precise, and rigid-object interactions. Table 13 summarizes the task definitions and corresponding success criteria used for evaluation.

Table 13: **Real-world task setups for sim-to-real experiments.** Detailed task definitions and success criteria are provided in Appendix D-2.

Task	Type	Task description	Success criterion
1.1	Articulate	Close the computer.	The laptop hinge angle is below 10°.
1.2	Precise	Place the pen in the cup.	The longitudinal axis of the pen lies inside the cup.
2.1	Rigid	Put the cube on the right into the cup.	The cube is inside the cup.
2.2	Rigid	Put the rubik near the cup.	The center of the Rubik's Cube bottom face is within $5\,\mathrm{cm}$ of the center of the cup bottom face.

D-4 Training and Evaluation Protocol

For each task, the step-conditioned and actor-blind variants use matched actor–critic architectures, reward programs, training budgets, and checkpoint-selection rules. Policies are trained only in reconstructed simulation. No demonstration data, real-world reward, or real-world gradient update is used.

The selected checkpoint for each policy is first evaluated on 100 simulation episodes. We report its empirical success percentage directly:

$$\widehat{\mathrm{SR}}_{\mathrm{sim}} = \frac{N_{\mathrm{success}}}{100} \times 100\%. \tag{20}$$

Real-world evaluation contains three blocks of ten trials per task and policy setting. Let $s_j \in \{0, \ldots, 10\}$ be the successful trials in block $j \in \{1, 2, 3\}$ and let $r_j = 10 s_j$ be the corresponding percentage. We report

$$\overline{r} = \frac{1}{3}\sum_{j=1}^{3} r_j, \qquad \mathrm{StdErr} = \frac{1}{\sqrt{3}}\sqrt{\frac{\sum_{j=1}^{3}(r_j - \overline{r})^2}{2}}. \tag{21}$$

The reported StdErr describes variation across the three real-world evaluation blocks.

D-5 Results

Explicit actor step context improves average success by 11.5 percentage points in reconstructed simulation and by 8.3 points in the real world, with consistent gains across all four tasks. Task 1.2 remains the most challenging, and the actor without step context fails in all 30 real-world trials. Both variants exhibit a clear sim-to-real drop, indicating that explicit task progress improves transfer but does not eliminate errors from perception, calibration, and physical dynamics. All policies are trained only in reconstructed simulation. Simulation results are empirical success rates over 100 episodes for a single selected checkpoint.

Table 14: **Simulation and real-world success rates (%) for direct sim-to-real transfer.** All policies are trained only in reconstructed simulation. Real-world values are reported as mean $\pm$ StdErr over three 10-trial runs.

Task	Reconstructed Simulation		Real-World Deployment	
	w/ Actor Step Index	w/o Actor Step Index	w/ Actor Step Index	w/o Actor Step Index
1.1	**45.0**	33.0	**33.3 $\pm$ 3.3**	26.7 $\pm$ 3.3
1.2	**25.0**	13.0	**13.3 $\pm$ 3.3**	0.0 $\pm$ 0.0
2.1	**36.0**	25.0	**23.3 $\pm$ 3.3**	16.7 $\pm$ 3.3
2.2	**41.0**	30.0	**30.0 $\pm$ 5.8**	23.3 $\pm$ 3.3
Average	**36.8**	25.3	**25.0 $\pm$ 3.8**	16.7 $\pm$ 2.2

E BIDIRECTIONAL TASK PROGRESS

We evaluate whether the persistent step state learned through clean stepwise training supports two complementary capabilities: forward composition across increasingly long task chains, and backward correction after a previously completed prerequisite becomes invalid. The former trains a separate policy for each composition length; the latter freezes a clean-trained policy and changes only its step-index input after an unseen regression.

E-1 FORWARD: COMPOSABLE STEPWISE REWARDS

All composition experiments use the privileged Gate-Rule for atomic-step and macro-boundary transitions, isolating compositional policy learning factors from visual-verification errors.

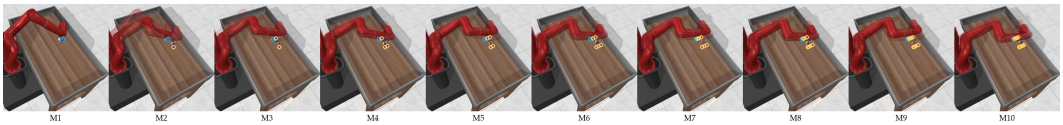


Figure 10: **Homogeneous composition protocol.** Homo_n moves one object through n distinct targets without resetting the robot, object, or scene. Each macro-subtask contains three atomic RoboStep steps. Panels show $n = 1, \ldots, 10$.

E-1.1 EXPERIMENT SETTING FORMULATION

Homogeneous composition. We use n to denote the number of macro-subtasks and K the number of atomic RoboStep steps. One macro-subtask transfers the object to one target through

$$P_j = [\text{pregrasp} \rightarrow \text{capture and lift} \rightarrow \text{place and release}]. \tag{22}$$

A homogeneous composition is

$$\mathrm{Homo}_n = P_1 \circ P_2 \circ \cdots \circ P_n, \qquad K = 3n. \tag{23}$$

Every P_j uses the same semantic primitive, reward template, and reward coefficients, but moves the object to a distinct target g_j. The complete reward program is formed by ordered concatenation:

$$\mathcal{R}(\mathrm{Homo}_n) = \mathcal{R}(P_1) \oplus \cdots \oplus \mathcal{R}(P_n), \tag{24}$$

where $\oplus$ denotes sequential composition rather than simultaneous reward summation. Appending P_{n+1} does not modify the reward functions or coefficients of any existing prefix macro.

For a non-terminal global step index $z_t \in \{0, \ldots, 3n-1\}$, the active macro and its local atomic step are

$$m_t = 1 + \left\lfloor \frac{z_t}{3} \right\rfloor, \qquad k_t = z_t \bmod 3. \tag{25}$$

Thus, z_t simultaneously selects the active target g_{m_t}, the local three-step reward component indexed by k_t, and the one-hot context supplied to the policy.

Target sequence and physical continuity. The experiment uses one Meta-World `pick-place-v3` scene, one robot, and one object. In the distinct-target protocol, the target sequence is

$$\begin{aligned}(g_1, \ldots, g_{10}) = \big[&(-0.08, 0.80),\ (0.08, 0.80),\ (-0.08, 0.84),\ (0.08, 0.84),\\ &(-0.08, 0.88),\ (0.08, 0.88),\ (-0.08, 0.82),\ (0.08, 0.82),\\ &(-0.08, 0.86),\ (0.08, 0.86)\big],\end{aligned} \tag{26}$$

where the coordinates specify the target position in the tabletop plane and the height follows the benchmark configuration. **No target repeats within the reported chain.**

The robot, object, and scene are not reset between macros. After P_j finishes, only the active target and persistent step state are updated:

$$\xi_{\text{start}}^{P_{j+1}} = \xi_{\text{terminal}}^{P_j}, \tag{27}$$

where ξ denotes the full physical simulator configuration. Consequently, arm pose, placement error, and accumulated control error are preserved across macro boundaries. A full reset occurs only at the beginning of a new episode.

Fixed-length heterogeneous composition. We additionally vary semantic composition while fixing $n = 3$ and $K = 9$. **All modules are instantiated within a single Meta-World `pick-place-v3` MuJoCo scene**. The same-scene macro modules are

$$\mathsf{A} = \text{pick-and-place}, \qquad \mathsf{B} = \text{object push}, \qquad \mathsf{C} = 3 \text{ step end-effector waypoint reaching}. \tag{28}$$

Module C requires the TCP to reach three successive Cartesian waypoints, one per atomic step, without object interaction. We compare

$$\text{Homo}_3 = \mathsf{AAA}, \qquad \text{Hetero}_3(\mathsf{AAB}), \qquad \text{Hetero}_3(\mathsf{ABC}). \tag{29}$$

All three settings share the same Meta-World scene, four-dimensional action interface, observation space, policy architecture, step-index width, 2,100 PPO iterations, and 750-step episode horizon. The physical state remains continuous across macro boundaries.

E-1.2 Training and Evaluation

Policy input and architecture. The compositional wrapper preserves the native Meta-World observation $o_t^{\text{MW}} \in \mathbb{R}^{39}$ and four-dimensional continuous action interface. The stepwise actor and critic receive

$$o_t^{\text{actor}} = o_t^{\text{critic}} = \left[o_t^{\text{MW}}; \mathbf{e}_{K_{\max}}(z_t)\right], \qquad a_t = [\Delta x_t, \Delta y_t, \Delta z_t, g_t] \in [-1, 1]^4. \tag{30}$$

The action is the standard four-dimensional Meta-World control vector, consisting of normalized end-effector Cartesian displacement and gripper control. **A single actor–critic policy controls all macro-subtasks throughout an episode with no macro-specific policy or skill instantiated**.

The actor and critic are MLPs with ELU activations and hidden widths $(256, 256, 128)$. The continuous state features are running-normalized, while the global step one-hot remains unnormalized. Its width covers all atomic steps and the terminal state. Within each matched length comparison, the shorter and longer policies use the same one-hot width.

Training protocol. The primary protocol trains a separate policy for each composition length. **The nominal training budget and episode horizon scale linearly with** n:

$$B(n) = 700n \quad \text{PPO iterations}, \qquad H_{\text{ep}}(n) = 250n \quad \text{environment steps}. \tag{31}$$

Training uses 128 parallel environments and 64 rollout steps per environment and PPO iteration. PPO hyperparameters and actor–critic hidden architectures are otherwise fixed.

The global step one-hot has enough coordinates for every atomic step and the terminal state. Within the length comparison, the shorter and longer policies use the same input width.

Strict evaluation criterion. Policies are evaluated deterministically on held-out environment seeds. **An episode succeeds only when all** $3n$ **gated atomic steps are completed**:

$$\text{SR}(\text{Homo}_n) = \Pr\left(z_t = 3n \text{ for some } t \le H_{\text{ep}}(n)\right). \tag{32}$$

Merely placing the object near the final target without completing the prescribed gated sequence does not count as success. Benchmark-native target success is retained only as an auxiliary diagnostic.

For each trained policy, numerators and denominators are pooled across the predeclared held-out environment seeds before computing SR. The table reports the actual number of completed episodes.

Non-stepwise controlled baseline. To test the importance of the complete stepwise interface, we jointly remove active-step reward routing, the policy step-index input, and transition bonuses. For the currently active macro, the control sums all three local dense terms,

$$\widetilde{r}_t = \sum_{k=0}^{2} D_k\left(o_t, o_{t-1}; g_{m_t}\right), \qquad \widetilde{b}_t = 0, \tag{33}$$

while the actor and critic receive only the native benchmark observation. The internal rule tracker is retained solely to **switch the sequential target and apply the same strict evaluation criterion**.

E-1.3 Experiment Results and Conclusions

Table 15: **Forward-composition result ledger.**

Composition	Homo_1	Homo_2	Homo_3	Homo_4	Homo_5	Homo_6	Homo_7	Homo_8	Homo_9	Homo_{10}
K	3	6	9	12	15	18	21	24	27	30
Eval. Episodes	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024
SR	100.00%	100.00%	99.23%	100.00%	100.00%	98.85%	97.28%	100.00%	93.99%	94.14%

The scratch policies remain near ceiling through Homo_8, with no reproducible monotonic degradation in the completed scratch results. Continuation evaluations show that policies can also complete Homo_{10} with $K = 30$ atomic steps.

Completed evaluations currently extend to Homo_{10} because of the available training-time budget and reporting cutoff, **rather than an observed failure at the next length**. Future experiments will continue to search for a reproducible composition breakpoint under matched scratch training, input width, target geometry, and training seeds.

The non-stepwise control in Equation 33 reaches 256/256 strict success on Homo_1, but obtains 0/256 on Homo_2 and the following compositional length. Thus, the tested configuration without reward routing, step-index context, or transition bonuses **fails on every evaluated composition beyond one macro**.

For the fixed-length semantic heterogenous compositional experiment as reported in Equaton 28, mean SR over three independently trained policies is

$$\begin{aligned} \mathrm{Homo}_3(\mathsf{AAA}) &: 99.23 \pm 7.51, \\ \mathrm{Hetero}_3(\mathsf{AAB}) &: 85.83 \pm 5.02, \\ \mathrm{Hetero}_3(\mathsf{ABC}) &: 73.33 \pm 5.74. \end{aligned} \tag{34}$$

The results indicate that semantic diversity in task composition increases difficulty: replacing repeated instances of the same subtask with heterogeneous combinations reduces success from $\mathrm{Homo}_3(\mathsf{AAA})$ to $\mathrm{Hetero}_3(\mathsf{AAB})$ and further to $\mathrm{Hetero}_3(\mathsf{ABC})$. Nevertheless, the heterogeneous compositions still achieve substantial success rates, suggesting that the explicit stepwise reward structure remains effective **even when successive steps correspond to different manipulation semantics**.

E-2 Backward: Regression and Recovery

E-2.1 Experiment Setting Formulation

Task source. We evaluate four tasks from ManiSkill 3: `PickCube-v1`, `PushCube-v1`, `PullCube-v1`, and `PickSingleYCB-v1`. For each task τ, we use an independently trained PPO actor–critic policy learned only from clean forward execution. Following the main policy interface, the actor and critic receive the native ManiSkill state observation together with the persistent step one-hot:

$$a_t \sim \pi_{\theta_\tau}\left(\cdot \mid o_t^{(\mathrm{MS})}, e_\tau(z_t)\right), \qquad V_{\psi_\tau} = V_{\psi_\tau}\left(o_t^{(\mathrm{MS})}, e_\tau(z_t)\right), \tag{35}$$

where

$$e_\tau(z_t) \in \{0,1\}^{K_\tau+1} \tag{36}$$

encodes the persistent step state, including the terminal state. Both actor and critic use three hidden layers of width 256 with Tanh activations. The actor outputs the native continuous action consumed by the ManiSkill `pd_joint_delta_pos` controller, while the critic outputs a scalar value.

During recovery evaluation,

$$\theta_{\tau,t} = \theta_{\tau,\text{clean}} \qquad \forall t, \tag{37}$$

so the actor receives no gradient update, online finetuning, disturbance curriculum, recovery demonstration, or recovery-specific reward. Within each task, the same checkpoint is used before and after the disturbance and for all compared rollback mechanisms.

Controlled regressions and environment randomization. Each episode begins from the native ManiSkill reset distribution, which randomizes the initial scene configuration, object pose, and task goal. For `PickSingleYCB-v1`, the manipulated YCB object and its geometry also vary across reconfigurations.

The disturbance itself is controlled rather than randomly sampled. For each task, its type, magnitude, and trigger phase are fixed. The frozen actor first reaches a designated downstream step through normal execution. Two policy steps later, at time t_d, we apply a **task-specific disturbance operator** Γ_τ:

$$s^{\text{sim}}_{t_d^+} = \Gamma_\tau\left(s^{\text{sim}}_{t_d^-}\right), \tag{38}$$

where s^{sim}_t denotes simulator state used only to construct and evaluate the controlled regression. The actor and **Gate-VLM** are not told that the intervention occurred.

Table 16: **Controlled regression settings.** The disturbance and evaluator-only recovery step are not provided to **Gate-VLM**.

Task	Family	Controlled regression	RGB reference
`PickCube-v1`	Pick-and-place	The grasped cube is returned to the table near the gripper, approximately 0.05 m from its pre-disturbance position, invalidating grasp-and-carry.	Step entry
`PushCube-v1`	Nonprehensile push	The visible target is displaced by 0.16 m, invalidating the previously satisfied object–target relation.	Step entry
`PullCube-v1`	Nonprehensile pull	The visible target is displaced by 0.16 m, invalidating the previously satisfied object–target relation.	Step entry
`PickSingleYCB-v1`	Object generalization	The grasped YCB object is returned to the table near the gripper, approximately 0.04 m from its pre-disturbance position.	Refreshed

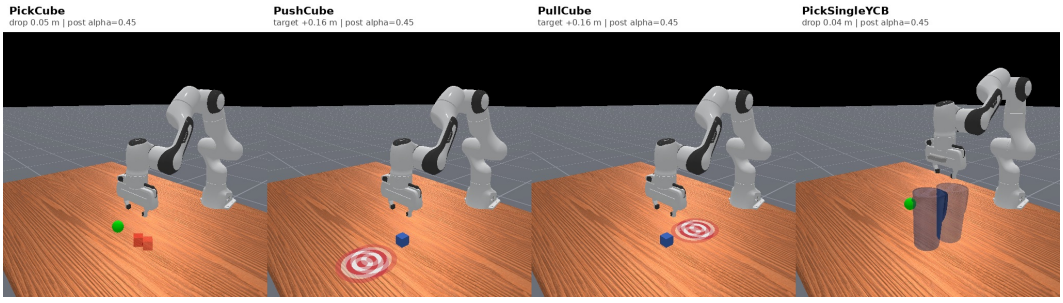


Figure 11: **Controlled regressions used for recovery evaluation.** Each panel overlays the valid pre-regression state and the corresponding disturbed state. The overlays are shown only for visualization; **Gate-VLM** receives the original RGB reference and current frame without overlays or information about the intervention metadata.

The first three tasks use the RGB frame saved when the downstream step was entered. In all cases, the reference is captured before the disturbance, while the relevant prerequisite is still satisfied.

Gate-VLM backward-step observer. Normal forward progress continues to use the original forward step tracker from the main method. The regression experiment augments this interface with backward-only visual authority.

Let

$$\mathcal{I}_t^{\text{back}} = \left[I^{\text{ref}}, I_t\right] \tag{39}$$

denote the RGB packet containing the valid reference image and the current observation. At a fixed query cadence of every

$$q = 3 \tag{40}$$

policy steps, Qwen3.5-9B (Qwen Team, 2026) predicts the step currently required by the task:

$$\hat{z}_t = G_\psi^{\text{back}}\left(\ell_\tau, z_t^-, \{\mathcal{V}_{\tau,i}\}_{i=0}^{z_t^-}, \mathcal{I}_t^{\text{back}}\right), \tag{41}$$

where ℓ_τ is the task description, z_t^- is the maintained step before the visual decision, and $\mathcal{V}_{\tau,i}$ is the **Frame-VLM**-generated visual completion description for step i.

The query schedule is fixed and is not triggered by the intervention. The observer receives only the task description, ordered step descriptions, last maintained step, and the two RGB images. It does not receive simulator states.

Backward-only step update. The visual observer may decrease, but not increase the persistent step state:

$$z_t^+ = \begin{cases} \hat{z}_t, & \hat{z}_t < z_t^-, \\ z_t^-, & \text{otherwise.} \end{cases} \tag{42}$$

Invalid, unchanged, or later-step predictions therefore leave the persistent state unchanged. After an accepted rollback, the same frozen actor resumes execution with the corrected step input:

$$a_t \sim \pi_{\theta_{\tau,\text{clean}}}\left(\cdot \mid o_t^{(\text{MS})}, e_\tau(z_t^+)\right). \tag{43}$$

No earlier-step policy is loaded, and no policy network is switched. Subsequent forward transitions are again handled by the original forward step tracker.

Compared step trackers. All variants use the same task-specific frozen actor, matched environment reset keys, disturbance operator, action space, recovery horizon, and official task-success predicate.

1. **Forward-only:** the pre-regression step state is retained and is not allowed to move backward, even after its prerequisite becomes invalid;
2. **Gate-VLM rollback:** the visual observer applies the backward-only update in Eq. 42;
3. **Oracle rollback:** privileged simulator state supplies the evaluator-derived recovery step $z^\star$, providing an upper-bound control.

E-2.2 Evaluation Protocol

An episode is counted as effective only when the clean actor naturally reaches the designated downstream step and receives the controlled disturbance. Each task and tracker is evaluated on

$$N = 50 \tag{44}$$

effective episodes using deterministic actor inference and matched reset keys across Forward-only, **Gate-VLM**, and Oracle.

The first three tasks use evaluation seed 12442 and the natural step-entry RGB reference. `PickSingleYCB-v1` uses evaluation seed 5042 and the refreshed-reference protocol. All four rows use a post-disturbance recovery horizon of

$$H = 120 \tag{45}$$

policy steps and a timely-rollback window of

$$W = 10. \tag{46}$$

Recovery Success Rate. Recovery Success Rate measures whether the original task is completed after the disturbance and within the controlled recovery horizon:

$$\text{Recovery SR} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\left[\exists t \in \left(t_d^{(n)}, t_d^{(n)} + H\right] : \text{Success}_\tau\left(s_t^{\text{sim},(n)}\right) = 1\right]. \tag{47}$$

The official task-success predicate is used only for outcome evaluation and is never provided to the actor or Gate-VLM.

Timely Rollback Rate. For episode n, let

$$t_b^{(n)} = \inf\left\{t \geq t_d^{(n)} : z_t < z_{t_d^-}^{(n)}\right\} \tag{48}$$

denote the first accepted backward update. Timely Rollback Rate measures whether any backward correction is applied within W policy steps:

$$\text{TRR@}W = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\left[0 \leq t_b^{(n)} - t_d^{(n)} \leq W\right]. \tag{49}$$

Stage Accuracy. Stage Accuracy additionally requires the first correction within the same window to exactly match the evaluator-derived recovery step:

$$\text{SA@}W = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\left[0 \leq t_b^{(n)} - t_d^{(n)} \leq W \ \wedge \ z_{t_b^{(n)}} = z^{\star(n)}\right]. \tag{50}$$

Thus, SA penalizes a missing rollback, an insufficient rollback, and an over-correction to a step earlier than the evaluator-derived recovery step.

E-2.3 Experiment Results and Conclusions

Table 17: **Controlled regression and zero-shot recovery.** Each row contains 50 effective disturbed episodes.

Task	Regression	Forward SR	Gate-VLM SR	Oracle SR	TRR@10	SA@10
`PickCube-v1`	Object drop, 0.05 m	0/50	**32/50**	32/50	50/50	49/50
`PushCube-v1`	Target shift, 0.16 m	2/50	**46/50**	47/50	50/50	49/50
`PullCube-v1`	Target shift, 0.16 m	1/50	**46/50**	45/50	50/50	32/50
`PickSingleYCB-v1`	Object drop, 0.04 m	0/50	**28/50**	29/50	49/50	35/50
Coverage total	*two RGB-reference protocols*	**3/200**	**152/200**	**153/200**	**199/200**	**165/200**

Across the four reported tasks, visual rollback achieves $152/200$ (76.0%) Recovery SR, compared with $3/200$ (1.5%) for Forward-only execution and $153/200$ (76.5%) for privileged Oracle rollback. Gate-VLM applies a backward correction within ten policy steps in $199/200$ episodes, corresponding to 99.5% TRR@10, and exactly matches the evaluator-derived recovery step in $165/200$ episodes, corresponding to 82.5% SA@10.

These results show that recovery behavior can already be latent in an actor **trained only on clean stepwise rewards**. Correcting only its persistent step context can elicit recovery from unseen regressions without updating the actor parameters or introducing a recovery-specific policy.

F Additional Diagnostic Ablations

F-1 Stepwise Reward Structure

For the controlled reward-structure study in Sec. 4.1, we evaluate 10 Meta-World tasks, five relatively challenging ManiSkill tasks, and three SoftGym tasks. Together, they cover articulated manipulation, planar transport, grasping and insertion, contact-rich interaction, fluid transfer, and deformable-object manipulation. The complete task list is given in Table 18.

The same task subset, step decomposition, and Frame-VLM-generated dense reward components are used across all four combinations of reward structure (stepwise or all-dense) and policy step-index input (with or without the index). Within each benchmark, policy architecture, optimization budget, and evaluation protocol are also fixed. Stepwise variants route only the active dense component and include the one-time transition bonus, whereas all-dense variants activate all dense components simultaneously with no transition bonus. The privileged Gate-Rule tracker is shared across variants only for matched task progression and evaluation.

Table 18: **Task subsets used in the Q1 reward-structure study.** The per-task visualization and configurations can be checked in Figure 7 and Table 24.

Benchmark	# Tasks	Tasks
Meta-World	10	`handle-press-v3`, `handle-pull-v3`, `plate-slide-side-v3`, `handle-press-side-v3`, `lever-pull-v3`, `plate-slide-v3`, `sweep-into-v3`, `handle-pull-side-v3`, `pick-out-of-hole-v3`, `sweep-v3`
ManiSkill	5	`PegInsertionSide-v1`, `StackCube-v1`, `PickSingleYCB-v1`, `RollBall-v1`, `PokeCube-v1`
SoftGym	3	`PassWater`, `PourWater`, `RopeFlatten`

F-1.1 Experiment Setting Formulation

For a task decomposed into K non-terminal steps, let $z_t \in \{0, \dots, K\}$ denote the persistent step state, where K is the terminal state. We define

$$A_i(t) = \mathbf{1}[z_t = i], \qquad \sum_{i=0}^{K-1} A_i(t) = 1$$

for every non-terminal timestep. Let $D_i(x_t, x_{t-1})$ denote the Frame-VLM-generated dense reward component for step i, and let P_t collect reward terms shared across all variants.

In the *stepwise* formulation, only the dense component associated with the active step contributes:

$$r_t^{\text{step}} = \sum_{i=0}^{K-1} A_i(t)\, D_i(x_t, x_{t-1}) + b_{z_t}\, \mathbf{1}[\text{accepted transition}] + P_t,$$

where b_{z_t} is a one-time bonus for an accepted step transition. After the transition, reward is routed to the new active step and completed-step dense components are no longer active.

The *all-dense* control removes all reward-side dependence on the current step:

$$r_t^{\text{all}} = \sum_{i=0}^{K-1} D_i(x_t, x_{t-1}) + P_t.$$

Thus, all Frame-VLM-generated dense components remain active simultaneously, with no step-dependent routing or transition bonus. The same privileged Gate-Rule tracker is retained only for matched task progression, termination, and evaluation, and never contributes to the all-dense reward.

Independently, we vary whether the actor and critic receive the current step index. This yields the 2×2 comparison between reward-side stepwise structure and policy-side step context.

F-1.2 Experiment Results and Conclusions

We independently vary whether the reward maintains an explicit stepwise state and whether the policy receives the current step index. All variants use the same Frame-VLM-generated reward components and matched policy-learning settings. Values report mean success rate with standard deviation across independent training runs as in Table 19.

Table 19: **Full results for stepwise reward structure and step-index input.**

	w/ Step-Index Input				w/o Step-Index Input			
Reward	Meta-World	ManiSkill	SoftGym	Avg.	Meta-World	ManiSkill	SoftGym	Avg.
Stepwise	92.20±3.15	62.00±2.67	49.12±1.51	67.77	46.38±0.87	44.40±4.87	49.83±1.66	46.87
All-dense	62.85±3.18	21.60±2.35	44.00±5.12	42.82	52.98±5.21	20.00±5.33	47.61±3.55	40.19

F-2 GATE-VLM INVOCATION CADENCE

F-2.1 EXPERIMENT SETTING FORMULATION

Controlled interface. We use the same 18-task subset as the Q1 reward-structure study, listed in Table 18. For every task, the step decomposition, Frame-VLM-generated reward program, actor–critic architecture, policy observations, PPO budget, and evaluation protocol are fixed across invocation variants.

For benchmark b, the actor and critic receive the benchmark observation together with the persistent step context:

$$a_t \sim \pi_\theta\left(\cdot \mid o_t^{(b)}, \mathbf{e}(z_t)\right), \qquad V_\psi = V_\psi\left(o_t^{(b)}, \mathbf{e}(z_t)\right). \tag{51}$$

The RGB observations used by Gate-VLM are not provided to these state-based policies. At an invoked timestep, the verifier receives the same inputs as in the main formulation:

$$v_t = \mathbf{1}\left[\text{Gate-VLM}\left(\ell, z_t, \mathcal{V}_{z_t}, \mathcal{I}_t\right) = \texttt{success}\right], \tag{52}$$

where

$$\mathcal{I}_t = \left[I^{\text{start}}, I_t^{\text{history}}, I_t\right] \tag{53}$$

is the benchmark-specific RGB packet. Outputs indicating `failure`, `unknown`, or an unparsable response yield $v_t = 0$. All variants use Qwen3.5-9B as Gate-VLM.

The accepted transition remains

$$e_t = \chi_t v_t, \qquad z_{t+1} = z_t + e_t. \tag{54}$$

Thus, this ablation changes only the invocation trigger χ_t; the visual decision v_t, persistent state update, routed dense rewards, transition bonus, and policy interface remain unchanged.

Periodic invocation. To isolate the effect of query timing, we replace the default task-conditioned invocation schedule with a task-state-independent periodic trigger. For benchmark-specific query interval F_b,

$$\chi_t^{\text{freq}}(F_b) = \mathbf{1}[z_t < K]\,\mathbf{1}[t \bmod F_b = 0]. \tag{55}$$

Benchmark-specific cadence ranges. A policy step does not represent the same temporal granularity across the three benchmarks, which use different native control interfaces, episode timescales, simulation environment and PPO rollout protocols. We therefore define **fast, intermediate, and slow** invocation regimes separately within each benchmark, rather than applying one absolute query interval to all environments. Specifically, we use intervals $\{16, 64, 128\}$ for Meta-World and $\{8, 16, 32\}$ for ManiSkill and SoftGym. These settings are intended to span a practical range of relative query cadences within each benchmark and absolute interval values are not compared across benchmarks.

We evaluate three query cadences per benchmark:

$$\begin{aligned} \mathcal{F}_{\text{MW}} &= \{16, 64, 128\}, \\ \mathcal{F}_{\text{MS}} &= \{8, 16, 32\}, \\ \mathcal{F}_{\text{SG}} &= \{8, 16, 32\}, \end{aligned} \tag{56}$$

corresponding to the fast, intermediate, and slow settings. Under periodic invocation,

$$e_t^{(F_b)} = \chi_t^{\text{freq}}(F_b) v_t, \qquad z_{t+1} = z_t + e_t^{(F_b)}. \tag{57}$$

Once a transition is accepted, the next step-local dense reward becomes active, the one-time transition bonus is issued, and the updated step one-hot is supplied to the actor and critic. Therefore, F_b changes only when verification is requested.

Verification-error asymmetry. The two types of verification error have different effects on the persistent reward state. A false rejection leaves z_t unchanged, so the current dense reward and policy context remain active and completion can be reconsidered at a later query. A false acceptance instead advances z_t, activates the next dense reward, issues the transition bonus, and changes the policy context.

More frequent invocation reduces the delay before a valid transition can be recognized, but also creates more opportunities for premature acceptance. Periodic cadence therefore trades off delayed progression against erroneous advancement.

Training and evaluation. Each task is trained independently. All cadence variants use the same reset distribution, reward functions, transition bonuses, policy architecture, PPO hyperparameters, training budget, checkpoint rule, and evaluation protocol. We report benchmark-average success rates with standard deviations over three independently trained policies. Official benchmark success predicates are used only for evaluation.

F-2.2 Experiment Results and Conclusions

Periodic invocation supports policy learning across all three benchmarks, but no single cadence performs best universally. On Meta-World, average success increases from $12.73\pm1.46\%$ at interval 16 to $23.60\pm2.16\%$ at interval 64 and $31.40\pm3.34\%$ at interval 128. On ManiSkill, the intermediate interval of 16 performs best at $43.67 \pm 3.18\%$, compared with $31.25 \pm 3.66\%$ at interval 8 and $38.33 \pm 1.20\%$ at interval 32. SoftGym is comparatively insensitive, with average success ranging from 27.11% to 28.56% across the three intervals.

These results show that RoboStep does not require task-conditioned query timing to remain operational, although the appropriate cadence depends on the benchmark. Excessively sparse queries may delay valid transitions, whereas excessively frequent queries repeatedly expose the persistent step state to verification errors. The weak fast-query results on Meta-World and ManiSkill are consistent with premature advancement being more damaging than delayed advancement. Because this ablation does not directly label individual verification errors, we treat this explanation as a mechanistic interpretation rather than a measured error decomposition.

F-3 Policy Context for Stepwise Reward Learning

F-3.1 Experiment Setting Formulation

Controlled comparison. We use the same five ManiSkill tasks: `StackCube-v1`, `PushCube-v1`, `PullCube-v1`, `PlaceSphere-v1`, and `PickCube-v1` across all variants. we fix the generated dense rewards, persistent stage machine, transition authority, PPO implementation, training budget, checkpoint-selection rule, and held-out evaluation protocol. The only experimental variable is the information exposed to the actor and critic. The stage tracker continues to route rewards even when neither policy network receives the stage index.

Let o_t^π and o_t^V denote the benchmark state observations available to the actor and critic, respectively, and let $e(z_t)$ be the active-stage one-hot. The full state-based policy used in Sec. 4.1 has

$$x_t^\pi = [o_t^\pi; e(z_t)], \qquad x_t^V = [o_t^V; e(z_t)]. \tag{58}$$

We evaluate four additional input variants:

$$\text{Actor-blind stage:} \qquad x_t^\pi = o_t^\pi, \qquad x_t^V = [o_t^V; e(z_t)], \tag{59}$$

$$\text{No policy-stage input:} \qquad x_t^\pi = o_t^\pi, \qquad x_t^V = o_t^V. \tag{60}$$

Visual actor variants. For the visual policies, the current RGB image is resized to 224×224 and encoded using a frozen DINOv2 ViT-S/14:

$$h_t = f_{\text{DINO}}(I_t) \in \mathbb{R}^{384}. \tag{61}$$

No gradient is propagated into the encoder. Let p_t denote robot-only proprioception. The two asymmetric visual variants use

$$\text{Visual actor with stage:} \qquad x_t^\pi = [h_t; p_t; e(z_t)], \qquad x_t^V = \left[o_t^V; e(z_t)\right], \tag{62}$$

$$\text{Visual actor without stage:} \qquad x_t^\pi = [h_t; p_t], \qquad x_t^V = \left[o_t^V; e(z_t)\right]. \tag{63}$$

Thus, the visual actor receives neither object poses, goal coordinates, nor the benchmark privileged state. The second variant additionally removes the stage index from the actor, while retaining a privileged stage-aware critic only during training. We refer to these as visual policies rather than RGB-only policies because the actor also receives robot proprioception. The actor images are policy observations and are distinct from the RGB packets supplied to **Gate-VLM** for transition verification.

Training and evaluation. Each variant is trained independently with three seeds on all five tasks. Checkpoints are selected using the same fixed validation protocol and evaluated on held-out test episodes. We report the mean test success rate over the resulting 15 task–seed cells.

F-3.2 Experiment Results and Conclusions

The stage-blind actor with a stage-aware critic achieves $67.20 \pm 10.83\%$ mean success, compared with $58.67 \pm 2.49\%$ when the step index is removed from both networks. Thus, even when the actor does not directly observe task progress, exposing the persistent step state to the critic provides a measurable optimization benefit.

The visual actor with and without explicit step context obtains $37.20 \pm 1.06\%$ and $37.07 \pm 0.96\%$, respectively. Their nearly identical performance indicates that, under the current frozen-DINO representation, **visual object and goal grounding** is the dominant limitation and adding the symbolic step index alone does not resolve it. Because all reward and optimization settings remain fixed, this decrease reflects the stricter visual policy input rather than a change in the reward method.

The stage-blind visual variant nevertheless demonstrates a deployment- compatible asymmetric interface: the deployed actor requires only frozen visual features and robot proprioception, while privileged state and stage information are confined to the training-time critic.

F-4 Sensitivity to Gate Errors

Task group and controlled variables. We use three staged Meta-World tasks: `handle-pull-side-v3`, `lever-pull-v3`, and `sweep-into-v3`. Across all operating points, we fix the task decomposition, step-local dense rewards, transition bonuses, persistent step machine, actor–critic architecture, PPO hyperparameters, training budget, checkpoint-selection rule, and evaluation protocol. Only the gate error probabilities vary.

Synthetic gate errors. To isolate the causal effects of verification errors from model-specific VLM behavior, we replace **Gate-VLM** with a synthetic gate. At a scheduled query, let $g_t^\star \in \{0, 1\}$ denote the privileged completion label of the active step and let $v_t \in \{0, 1\}$ denote gate acceptance. We define

$$\Pr(v_t = 1 \mid g_t^\star = 0) = \alpha, \qquad \Pr(v_t = 0 \mid g_t^\star = 1) = \beta, \tag{64}$$

where α is the false-acceptance rate (FAR) and β is the false-rejection rate (FRR). The transition rule remains unchanged:

$$e_t = \chi_t v_t, \qquad z_{t+1} = z_t + e_t. \tag{65}$$

A false acceptance therefore advances the persistent step state, issues the normal transition bonus, and activates the next step reward. A false rejection leaves the current step active, allowing a later retry.

Query schedule and operating-point grid. All settings use the same Hybrid-K64 schedule: native candidate events invoke the gate, while an additional shadow query is issued every 64 environment steps during an active step; coincident events are counted once. FAR and FRR are therefore conditional rates over gate opportunities rather than per-environment-step probabilities.

We evaluate

$$\alpha \in \{0, 0.05, 0.15, 0.30, 0.50, 0.75, 1.00\}, \qquad \beta \in \{0, 0.05, 0.15, 0.30\}, \tag{66}$$

forming the 4×7 grid in Figure 5. Each operating point contains three independently trained policies per task, giving $28 \times 3 \times 3 = 252$ task–seed runs. Checkpoints are selected using the fixed validation protocol and evaluated on 50 held-out episodes.

Results. At $(\mathrm{FAR}, \mathrm{FRR}) = (0, 0)$, mean success is 94.0%. Increasing FAR to 0.30 while fixing FRR at zero reduces success to 21.8%, whereas increasing FRR to 0.30 while fixing FAR at zero retains 66.0% success. The surface is not strictly monotonic at every individual cell because of task and training-seed variability, but the overall degradation is substantially steeper along the FAR axis. These results show that premature step advancement is considerably more damaging than delayed advancement under the tested protocol.

Real-VLM operating points. We additionally evaluate five Qwen-family checkpoints under the same Hybrid-K64 query schedule, prompt, camera, and parser used for policy training. For each task and model, FAR and FRR are measured on a fixed packet containing 200 operational positive and 200 operational negative opportunities. The matched panel uses `sweep-into-v3` and `lever-pull-v3`; `handle-pull-side-v3` is retained only as an auxiliary diagnostic because an equally complete large-model policy panel is not available for that task.

Table 20: **Real Gate-VLM operating points and downstream policy success.** All models use the same query and parsing protocol. SR reports the seed-42 held-out policy result; additional completed seeds are discussed in text.

	`sweep-into-v3`			`lever-pull-v3`		
Model	**FAR**	**FRR**	**SR**	**FAR**	**FRR**	**SR**
Qwen3-VL-4B	0.695	0.270	0.00	0.815	0.165	0.04
Qwen3-VL-8B	0.540	0.380	0.44	0.555	0.465	0.14
Qwen3.5-9B	0.795	0.195	0.26	0.830	0.080	0.00
Qwen3.6-27B	0.895	0.245	0.58	0.385	0.095	0.00
Qwen3-VL-32B	0.740	0.280	0.00	0.615	0.240	0.00

G Full Related Work

G-1 Full Related Work Table

G-2 Full Related Work Task Table

Table 22: **Original task coverage of related work.** Task names are copied from the original papers whenever they are explicitly listed. Simulation and real-world evaluations are marked separately; a benchmark count is not treated as a downstream policy task unless the paper uses it that way.

Method	**# Tasks**	**Original Tasks (simulation / real)**
LIV (Ma et al., 2023a)	20	**Sim (11):** MetaWorld—close drawer, open drawer, turn faucet right, turn faucet left, move black mug right, move the white mug left; FrankaKitchen—open microwave, slide cabinet, open left door, turn on stove, switch on light. **Real (9):** RealRobot fruit–container assignments: {apple, pineapple, pear}×{tray, black pot, green pot}.

Continued on next page

Method	# Tasks	Original Tasks (simulation / real)
Code-as-Reward (Venuto et al., 2024)	8	**Sim (8):** MiniGrid—DoorKey, Unlock, DoorKey8x8; PandasGym—PandasPush, PandasSlide; CLIPort—Separating-Piles, Put-Blocks-in-Bowls, Packing-Shapes. **Real:** –.
RL–VLM–F (Wang et al., 2024)	7	**Sim (7):** OpenAI Gym—Cart Pole; MetaWorld—Open Drawer, Sweep Into, Soccer; SoftGym—Fold Cloth, Straighten Rope, Pass Water. **Real:** –.
EUREKA (Ma et al., 2023b)	29	**Sim (29):** IsaacGym—Cartpole, Quadcopter, FrankaCabinet, Anymal, BallBalance, Ant, AllegroHand, Humanoid, ShadowHand; Dexterity—Over, DoorCloseInward, DoorCloseOutward, DoorOpenInward, DoorOpenOutward, Scissors, SwingCup, Switch, Kettle, LiftUnderarm, Pen, BottleCap, CatchAbreast, CatchOver2Underarm, CatchUnderarm, ReOrientation, GraspAndPlace, BlockStack, PushBlock, TwoCatchUnderarm. **Real:** –.
FuRL (Fu et al., 2024)	10	**Sim (10):** MetaWorld MT10—button-press-topdown-v2, door-open-v2, drawer-close-v2, drawer-open-v2, peg-insert-side-v2, pick-place-v2, push-v2, reach-v2, window-close-v2, window-open-v2. **Real:** –.
VLAC (Zhang et al.)	4	**Sim:** –. **Real (4):** Desktop Sweep Disposal, Pick and Place Bowl, Unfold Mat, Rice Scooping and Transfer.
GVL (Ma et al., 2025)	6 sim + 7 real	**Sim (6):** single_insertion, double_insertion, mug_on_plate, bowl_on_rack, plate_on_rack, glass_on_rack. **Real (7 downstream):** bowl-in-rack, banana-handover, close-laptop, open-drawer, remove-gears, pen-handover, fold-dress. Additional benchmark scale: ALOHA-250 and more than 300 real-world tasks across OXE datasets.
PRM4RL (Zhang et al., 2025b)	15	**Sim (15):** MetaWorld—button-press, door-close, drawer-close, faucet-open, window-open, sweep-into, pick-place, pick-out-of-hole, sweep, push; ManiSkill—LiftCube, PullCube, PushCube, PickCube, LiftPegUpright. **Real:** –.
ReWiND (Zhang et al., 2025a)	13	**Sim (8):** MetaWorld—Window Close, Reach Wall, Faucet Close, Coffee Button, Button Press Wall, Door Lock, Handle Press Side, Sweep Into. **Real (5):** Separate blue/orange cups, Fold blue towel, Open red trash bin, Put orange cup on red plate, Put fruit-colored object in box.
SARM (Chen et al., 2026a)	3	**Sim (1):** MuJoCo cube-picking. **Real (2):** T-shirt folding, unloading dishes from a rack.
RoboReward (Lee et al., 2026)	5 downstream	**Sim (3):** Robomimic—Lift, Can, Square. **Real (2):** Pick-and-place monkey, Open drawer. RoboRewardBench/OXE is retained as benchmark data, not expanded into a finite task list.
TOPReward (Chen et al., 2026c)	130 bench. + 6 downstream	**Sim:** –. **Real (6 downstream):** Place toy car in box, Stack red cube on green cube, Put pen into cup, Place doll in box, Pick up cube, Put cube in cup. ManiRewardBench contains 130 unique real-world manipulation tasks.

Continued on next page

Method	# Tasks	Original Tasks (simulation / real)
Large Reward Models (Wu et al., 2026)	1 named + sim NR	**Sim:** ManiSkill3 benchmark; task names are not enumerated in the paper text. **Real (1):** Pick-and-place toy giraffe into a target bowl.
ARM–FM (Castanyer et al., 2026)	10 named + 1000 generated	**Sim:** MiniGrid—DoorKey, UnlockToUnlock, BlockedUnlockPickup, KeyCorridorS3R3; Craftium—mine a diamond; MetaWorld—Assembly, Bin-Picking, Pick-Place, Shelf-Place, Stick-Push; XLand–MiniGrid—1,000 generated tasks. **Real:** –.
RoboMeter (Liang et al., 2026)	8 named downstream	**Sim (2):** LIBERO-90 Task 28 (close the top drawer), Task 33 (close the microwave). **Real (6):** Put the bowl on the table; put the corn in the pot and then put the lid on the pot; Put the bread in the oven; Put the red bowl on the blue plate; Put the marker in the pen cup; Put the red cup on the purple coaster. The full RBM-1M/RBM-EVAL data span 21 embodiments and are not a finite named-task list.
ARM (Mao et al., 2026)	1	**Sim:** –. **Real (1):** bimanual towel-folding.
SOLE–R1 (Schroeder et al., 2026)	39 named†	**Sim (33):** MetaWorld—faucet-open, faucet-close, window-open, window-close, plate-slide (puck in net), drawer-close, drawer-open, door-close, door-open, button-press, handle-pull, soccer, lever-pull, coffee-button, handle-press, coffee-push; RoboSuite—Lift, Wipe, Door, PickPlaceCan; LIBERO—Close microwave, Open microwave, Turn on stove, Turn off stove, Close top drawer, Open bottom drawer; ManiSkill—PickPanda, PushCube, PullCube, Mobile-OpenCabDrawer, PickSO100, PickWidowXAI, PickSingleYCB. **Real (6):** Pick can, Touch strawberry in clutter, Push can to cube, Close drawer, Open drawer, Insert pipe.
ProcVLM (Feng et al., 2026)	10 sim + 1 real	**Sim (10):** LIBERO-10. **Real (1):** Stack Bowls.
MARVL (Zhou et al., 2026)	8	**Sim (8):** MetaWorld—Button press (topdown), Door open, Drawer close, Drawer open, Push, Reach, Window close, Window open. **Real:** –. Panda-Gym is used for domain-shift/cross-embodiment evaluation, but the paper does not enumerate a separate task list.
SARM2 (Chen et al., 2026b)	10 bench. + 2 downstream	**Sim:** –. **Real benchmark (10):** Pick and place plates into bin, Pick and place plates into dish rack, Fold the t-shirt, Fold the shorts, Pull plugs off the socket, Clean whiteboard with whiteboard eraser, Set dinner table, Sweep paper scraps with broom, Coil and wrap headphones, Put away an umbrella. **Real downstream (2):** Folding Shorts, Cleaning Whiteboard.
STDR (Yang et al., 2026)	17	**Sim (14):** MetaWorld—Button Press, Window Open, Reach Wall, Plate Slide, Push, Coffee Push, Stick Push, Pick Place; ManiSkill—PullCube, PushCube, PickCube; Franka Kitchen—Slide Cabinet, Kettle, Microwave. **Real (3):** PutPepperInBucket, PutToyOnPlate, ReachAndPullDrawer.

Continued on next page

Method	# Tasks	Original Tasks (simulation / real)
World Value Models (Wang et al., 2026b)	15 bench. + 6 downstream	**Sim benchmark (6):** RoboSuite—lift, pick & place bread, pick & place can, pick & place cereal, pick & place milk, stack. **Real benchmark (9):** AgileX—carrot off plate, carrot on plate, mickey box, sausage pot; ARX—flip bottles, open box, split cups, stack bowls, stack cups. **Downstream:** Sim—Stack cubes, Put cereal in tray, Put milk in tray; Real—Put mouse in box, Put shrimp in pot, Put marker in holder.
RoboStep (ours)	75 sim + 4 real	**Sim:** Demonstrated in Table 24. **Real:** Demonstrated in Table 4.

† The SOLE–R1 paper lists 39 named tasks in its task table, while one figure caption states 40.

Reward-model-only evaluations on fixed trajectories (e.g., progress/ranking accuracy or failure detection) are excluded. Each task is assigned to one primary physical-interaction category. Long-horizon is an orthogonal attribute and counts explicitly identified long-horizon task families; difficulty variants and atomic sub-stages are not counted separately. '0' denotes that no such task is reported, 'NR' denotes insufficient task-level information in the original paper for a defensible count, and 'N/A' denotes a reported robotic task suite that cannot be meaningfully partitioned using this manipulation taxonomy. Aggregate benchmark names are not expanded into task-level totals unless the benchmark provides an explicit task list. For SOLE–R1, the main text and Fig. 3 report 40 tasks, whereas Appendix Table 3 lists 39; we follow Fig. 3, which additionally includes `handle-press-side`.

Table 23: **Task-type coverage of reported robot-learning tasks.** Counts include only downstream robot policy/control experiments in which the proposed method is used for policy training, policy refinement, planning, or closed-loop execution.

Method	Sim	Real	Rigid	Artic.	Precise/ contact	Deform./ Fluid	Long-horizon
LIV (Ma et al., 2023a)	11	9	11	9	0	0	3
Code-as-Reward (Venuto et al., 2024)	2	0	2	0	0	0	0
RL–VLM–F (Wang et al., 2024)	6	0	2	1	0	3	0
EUREKA (Ma et al., 2023b)	29	0	N/A	N/A	N/A	N/A	0
FuRL (Fu et al., 2024)	10	0	3	5	2	0	0
VLAC (Zhang et al.)	0	4	1	0	1	2	0
GVL (Ma et al., 2025)	6	7	7	2	3	1	0
PRM4RL (Zhang et al., 2025b)	15	0	8	5	2	0	2
ReWiND (Zhang et al., 2025a)	8	5	5	5	2	1	0
SARM (Chen et al., 2026a)	1	1	1	0	0	1	1

Continued on next page

Method	Sim	Real	Rigid	Artic.	Precise/ contact	Deform./ Fluid	Long- horizon
RoboReward (Lee et al., 2026)	3	2	3	1	1	0	0
TOPReward (Chen et al., 2026c)	0	6	3	0	3	0	0
Large Reward Models (Wu et al., 2026)	NR	1	NR	NR	NR	NR	NR
ARM–FM (Castanyer et al., 2026)	5	0	3	0	2	0	0
Phys2Real (Wang et al., 2026a)	2 families	2 families	2	0	2	0	0
RoboMeter (Liang et al., 2026)	2	7	5	3	1	0	1
ARM (Mao et al., 2026)	0	1	0	0	0	1	1
SOLE–R1 (Schroeder et al., 2026)	34	6	13	22	5	0	0
ProcVLM (Feng et al., 2026)	10	1	7	3	1	0	10
MARVL (Zhou et al., 2026)	10	0	3	5	2	0	0
SARM2 (Chen et al., 2026b)	0	2	0	0	1	1	2
STDR (Yang et al., 2026)	14	0	6	3	5	0	NR
World Value Models (Wang et al., 2026b)	3	3	4	0	2	0	0
CoRe (Ni et al., 2026)	10	5	2	2	6	5	0
RoboStep (ours)	75	4	22	23	24	6	17

Table 24: **Per-task stepwise reward specification.** Frame-VLM state/API inputs, ordered stages, and dense reward components. ***LH*** is short for long-horizon.

Task	Frame-VLM state/API	K	Ordered stages	Dense components
MetaWorld				
`assembly-v3` *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** `approach_object`; **2** `capture_object`; **3** `fixture_clearance`; **4** `transport_or_extract`; **5** `verify_task_outcome`	**1** `tcp_to_pregrasp`, `gripper_opening`; **2** `tcp_to_grasp`, `held`, `capture_progress`; **3** `tcp_to_vertical`, `clearance_progress`, `tcp_to_grasp`, `held`; **4** `tcp_to_transport`, `object_transport_to_final`, `clearance_progress`, `tcp_to_grasp`, `held`, `object_to_final`; **5** `object_to_final`, `fine_object_to_final`, `tcp_to_grasp`, `held`

Continued on next page

Table 24 – continued

Task	Frame-VLM state/API	K	Ordered stages	Dense components
basketball-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
bin-picking-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
box-close-v3 *Artic.*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
button-press-topdown-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
button-press-topdown-wall-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
button-press-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
button-press-wall-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
coffee-button-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
coffee-pull-v3 *Rigid*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
coffee-push-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening

Continued on next page

Table 24 – continued

Task	Frame-VLM state/API	K	Ordered stages	Dense components
dial-turn-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
disassemble-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
door-close-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
door-lock-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
door-open-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
door-unlock-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_lock; **2** contact_lock; **3** unlock_articulation	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
drawer-close-v3 *Artic.*	TCP, handle/target poses	3	**1** approach_handle; **2** contact_handle; **3** close_drawer	**1** align_xy_distance, tcp_above_handle; **2** hook_distance, tcp_above_handle; **3** open_distance, pull_waypoint_distance
drawer-open-v3 *Artic.*	TCP, handle/target poses	3	**1** align_above_handle; **2** hook_handle; **3** open_drawer	**1** align_xy_distance, tcp_above_handle; **2** hook_distance, tcp_above_handle; **3** open_distance, pull_waypoint_distance
faucet-close-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_handle; **2** contact_handle; **3** close_faucet	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
faucet-open-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_handle; **2** contact_handle; **3** open_faucet	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
hammer-v3 *Tool*	TCP, object/target/initial poses, gripper	3	**1** approach_object; **2** capture_object; **3** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_obj, held, capture_progress; **3** obj_to_target, tcp_to_goal, tcp_to_obj, gripper_closed, held
hand-insert-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
handle-press-side-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_handle; **2** contact_handle; **3** press_handle	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance

Continued on next page

Table 24 – continued

Task	Frame-VLM state/API	K	Ordered stages	Dense components
handle-press-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
handle-pull-side-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
handle-pull-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_handle; **2** contact_handle; **3** pull_handle	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
lever-pull-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
peg-insert-side-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
peg-unplug-side-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
pick-out-of-hole-v3 *Precise*	TCP, object/target/initial poses, gripper	3	**1** approach_object; **2** capture_object; **3** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_obj, held, capture_progress; **3** obj_to_target, tcp_to_goal, tcp_to_obj, gripper_closed, held
pick-place-v3 *Rigid*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
pick-place-wall-v3 *Precise*	TCP, object/target/initial poses, gripper	3	**1** approach_object; **2** capture_object; **3** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_obj, held, capture_progress; **3** obj_to_target, tcp_to_goal, tcp_to_obj, gripper_closed, held
plate-slide-back-side-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
plate-slide-back-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening

Continued on next page

Table 24 – continued

Task	Frame-VLM state/API	K	Ordered stages	Dense components
plate-slide-side-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
plate-slide-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
push-back-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
push-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
push-wall-v3 *Rigid*^{LH}	TCP, object/target poses, gripper	5	**1** approach_object; **2** reach_left_clearance; **3** reacquire_front_push_contact; **4** clear_wall_front; **5** verify_task_outcome	**1** prepush_distance; **2** waypoint_distance, tcp_obj_distance; **3** path_contact_distance, tcp_obj_distance, path_progress; **4** waypoint_progress, path_contact_distance, gripper_opening; **5** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
reach-v3 *Rigid*	TCP–target distance	1	**1** approach_target	**1** tcp_to_target
reach-wall-v3 *Rigid*	TCP–target distance	1	**1** approach_target	**1** tcp_to_target
shelf-place-v3 *Precise*^{LH}	TCP, object/target/initial poses, gripper, optional orientation	5	**1** approach_object; **2** capture_object; **3** fixture_clearance; **4** transport_or_extract; **5** verify_task_outcome	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, capture_progress; **3** tcp_to_vertical, clearance_progress, tcp_to_grasp, held; **4** tcp_to_transport, object_transport_to_final, clearance_progress, tcp_to_grasp, held, object_to_final; **5** object_to_final, fine_object_to_final, tcp_to_grasp, held
soccer-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
stick-pull-v3 *Tool*	TCP, tool/secondary/target poses, gripper, optional orientation	4	**1** approach_tool; **2** capture_and_lift_tool; **3** tool_endpoint_at_secondary_object; **4** secondary_object_at_goal	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, tool_lift; **3** effect_to_secondary, alignment_distance, gripper_closed, held, tcp_to_grasp; **4** secondary_to_target, effect_to_secondary, held
stick-push-v3 *Tool*	TCP, tool/secondary/target poses, gripper, optional orientation	4	**1** approach_tool; **2** capture_and_lift_tool; **3** tool_endpoint_at_secondary_object; **4** secondary_object_at_goal	**1** tcp_to_pregrasp, gripper_opening; **2** tcp_to_grasp, held, tool_lift; **3** effect_to_secondary, alignment_distance, gripper_closed, held, tcp_to_grasp; **4** secondary_to_target, effect_to_secondary, held
sweep-into-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening

Continued on next page

Table 24 – continued

Task	Frame-VLM state/API	K	Ordered stages	Dense components
sweep-v3 *Rigid*	TCP, object/target poses, gripper	3	**1** approach_object; **2** contact_object; **3** verify_task_outcome	**1** prepush_distance; **2** contact_distance, tcp_obj_distance, gripper_opening; **3** obj_target_distance, target_contact_distance, tcp_obj_distance, gripper_opening
window-close-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
window-open-v3 *Artic.*	TCP, control/handle, target pose	3	**1** approach_control; **2** contact_control; **3** verify_task_outcome	**1** prepress_distance; **2** tcp_to_contact_point; **3** contact_distance, press_or_target_distance, tcp_press_axis_distance
ManiSkill				
PickCube-v1 *Rigid*	TCP, cube/goal poses, grasp	3	**1** reach_object; **2** establish_grasp; **3** transport_and_settle	**1** tcp_obj; **2** grasped; **3** obj_goal
PushCube-v1 *Rigid*	TCP, cube/goal poses	3	**1** reach_pre_push_pose; **2** establish_contact; **3** push_to_goal	**1** tcp_push; **2** tcp_push; **3** obj_goal
PullCube-v1 *Rigid*	TCP, cube/goal poses	3	**1** reach_pull_pose; **2** establish_contact; **3** pull_to_goal	**1** tcp_pull; **2** tcp_pull; **3** obj_goal
LiftPegUpright-v1 *Precise*	TCP, peg pose, grasp, upright/height	4	**1** reach_peg; **2** grasp_peg; **3** reorient_upright; **4** settle	**1** tcp_peg; **2** grasped; **3** upright; **4** upright, z_error
PegInsertionSide-v1 *Precise*	TCP, peg/hole poses, grasp	4	**1** reach_peg; **2** grasp_peg; **3** align_with_hole; **4** insert	**1** tcp_peg; **2** grasped; **3** align_error; **4** hole_error
PlugCharger-v1 *Precise*^{LH}	TCP, charger/goal poses, grasp, orientation	5	**1** reach_charger; **2** grasp_charger; **3** orient; **4** align_with_receptacle; **5** insert	**1** tcp_obj; **2** grasped; **3** orientation_error; **4** obj_goal, orientation_error; **5** obj_goal
PokeCube-v1 *Tool*	TCP, peg/cube/goal poses, grasp, orientation	4	**1** reach_tool; **2** grasp_tool; **3** align_tool_with_cube; **4** poke_to_goal	**1** tcp_peg; **2** grasped; **3** peg_angle, head_cube_distance; **4** cube_goal
PullCubeTool-v1 *Tool*	TCP, tool/cube poses, grasp, hook geometry	4	**1** reach_tool; **2** grasp_tool; **3** hook_cube; **4** pull_to_goal	**1** tcp_tool; **2** grasped; **3** hook_error; **4** cube_base_distance
PushT-v1 *Precise*	TCP, T/goal poses, orientation	4	**1** reach_contact_pose; **2** translate_object; **3** rotate_and_align; **4** settle	**1** tcp_obj; **2** xy_error; **3** angle_error; **4** xy_error, angle_error
RollBall-v1 *Rigid*	TCP, ball/goal pose, ball velocity	4	**1** reach_ball; **2** establish_contact; **3** roll_to_goal; **4** settle	**1** tcp_hit; **2** tcp_ball; **3** ball_goal; **4** ball_speed
StackCube-v1 *Rigid*^{LH}	TCP, cube poses, grasp, stack geometry, success	5	**1** reach_cube; **2** grasp_cube; **3** align_above_base; **4** place; **5** release	**1** tcp_obj; **2** grasped; **3** goal_error; **4** goal_error; **5** success
StackPyramid-v1 *Precise*^{LH}	TCP, three cube poses, grasp/static flags, success	7	**1** grasp_red; **2** place_red_beside_green; **3** release_red; **4** grasp_blue; **5** align_blue; **6** stack_blue; **7** release_blue	**1** tcp_red, grasp_red; **2** red_green_distance; **3** beside, release_red; **4** tcp_blue, grasp_blue; **5** blue_goal; **6** stacked; **7** success
PickSingleYCB-v1 *Rigid*	TCP, object/goal poses, grasp, placed/static	4	**1** reach_object; **2** establish_grasp; **3** transport_to_goal; **4** settle	**1** tcp_obj; **2** grasped; **3** goal; **4** placed, static
PlaceSphere-v1 *Rigid*	TCP, sphere/bin poses, grasp	4	**1** reach_sphere; **2** grasp_sphere; **3** align_with_bin; **4** place_and_release	**1** tcp_obj; **2** grasped; **3** goal; **4** on_bin, released
FMBAssembly1Easy-v1 *Precise*^{LH}	TCP, bridge/fixture/goal poses, grasp, orientation	6	**1** reach_bridge; **2** grasp_bridge; **3** reorient_with_fixture; **4** regrasp_bridge; **5** align_with_board; **6** place_bridge	**1** tcp_bridge; **2** grasped; **3** fixture_error, goal_angle; **4** grasped; **5** goal_pos, goal_angle; **6** goal_pos

Continued on next page

Table 24 – continued

Task	Frame-VLM state/API	K	Ordered stages	Dense components
`RotateValve Level4-v1` *Artic.*	finger tips, valve pose/qpos/qvel, turn direction, success	4	**1** `reach_valve_spokes`; **2** `establish_turning_motion`; **3** `rotate_full_turn`; **4** `settle_success`	**1** `contact_error`; **2** `contact_error`, `positive_motion`; **3** `rotation_progress`; **4** `success`, `valve_speed`
`TwoRobotStack Cube-v1` *Precise*^{LH}	robot TCPs, cube/goal poses, grasp, success	5	**1** `reach_assigned_cubes`; **2** `grasp_top_cube`; **3** `place_bottom_cube`; **4** `stack_top_cube`; **5** `release_both`	**1** `reach`; **2** `cubeA_grasped`; **3** `cubeB_goal`, `cubeA_grasped`; **4** `cubeA_top_error`; **5** `success`
`DrawTriangle-v1` *Tool*	TCP, canvas/vertices, path coverage, success	4	**1** `reach_canvas_start`; **2** `trace_first_side`; **3** `trace_remaining_sides`; **4** `complete_triangle`	**1** `reach`; **2** `path_coverage`; **3** `path_coverage`; **4** `covered`
`TriFingerRotate CubeLevel4-v1` *Precise*	finger tips, cube/goal pose, position/orientation error, success	4	**1** `reach_cube_with_fingers`; **2** `lift_and_translate_cube`; **3** `match_goal_orientation`; **4** `settle_at_goal`	**1** `tip_to_cube`; **2** `position_progress`; **3** `pose_progress`; **4** `success`
SoftGym				
`ClothDrop` *Deform.*	particles, cloth size/radius, target layout	2	**1** `approach_release`; **2** `flat_completion`	**1** `target_progress`, `target_alignment`, `low_height`, `spread`; **2** `target_progress`, `target_alignment`, `flatness`, `spread`
`ClothFlatten` *Deform.*	particles, cloth size/radius	2	**1** `spread_progress`; **2** `flat_completion`	**1** `spread`, `flatness`; **2** `spread`, `flatness`
`ClothFold` *Deform.*	particles, fold groups, initial group distance	2	**1** `fold_progress`; **2** `fold_completion`	**1** `coverage_A_to_B`, `coverage_B_to_A`; **2** `coverage_A_to_B`, `coverage_B_to_A`
`RopeFlatten` *Deform.*	ordered particles, rope length	2	**1** `stretch_progress`; **2** `straight_completion`	**1** `extension`, `straightness`; **2** `straightness`, `extension`, `on_plane`
`PassWater` *Fluid*	cup/target x, water particles, in-glass test, glass geometry	2	**1** `transport_progress`; **2** `water_delivery`	**1** `transport_progress`, `retained_water`; **2** `target_closeness`, `spill_penalty`
`PourWater` *Fluid*	source/target cup geometry, water particles, in-glass test	2	**1** `pour_progress`; **2** `target_fill_completion`	**1** `cup_alignment`, `tilt`, `early_transfer`, `retained_water`; **2** `target_fill`, `retained_water`, `source_emptying`

Table 21: **Comparison of robot reward-learning and structured reward-design approaches.** We compare how prior methods represent task progress, obtain reward supervision, verify progress transitions, and use the resulting rewards for policy learning.

Method	Reward Structure	Reward Supervision	Transition Verification	Learning
Visual progress rewards for robot learning				
LIV (Ma et al., 2023a)	global value / ΔV	temporal order + goal/outcome	–	planning / LCBC
ReWiND (Zhang et al., 2025a)	global scalar progress	target demos + temporal/rewind labels	external success signal	online RL
SARM (Chen et al., 2026a)	stage-aware scalar progress	target demos + stage boundaries	stage re-estimation	offline FT
RoboReward (Lee et al., 2026)	global ordinal progress	human scores + temporal labels	terminal scoring	online RL
TOPReward (Chen et al., 2026c)	global completion value	frozen VLM token likelihood	–	offline FT
Large Reward Models (Wu et al., 2026)	global multi-head progress	multi-source video + weak temporal labels	completion head	PPO / filtered BC
RoboMeter (Liang et al., 2026)	persistent stage + global progress	temporal + preference labels	success classifier	online RL
SOLE-R1 (Schroeder et al., 2026)	global scalar progress	sim progress + temporal/reversal labels	–	online RL
ProcVLM (Feng et al., 2026)	stage-aware scalar progress	successful videos + procedure labels	stage re-estimation	offline FT
STDR (Yang et al., 2026)	stage-aware scalar progress	expert videos + stage labels	learned stage classifier	online RL
WVM (Wang et al., 2026b)	global distributional value	temporal + rewind + human labels	–	offline FT
Programmatic and structured reward design				
Reward Machines (Toro Icarte et al., 2018)	persistent automaton reward	manual automaton + event labels	event predicate	online RL
Text2Reward (Xie et al., 2024)	flat executable reward	task text + environment state/API	–	online RL
Eureka (Ma et al., 2023b)	flat executable reward	task text + environment source + fitness	reward search	online RL
Code-as-Reward (Venuto et al., 2024)	persistent subtask reward	VLM decomposition + trajectory verification	programmatic completion predicate	online RL
PRM4RL (Zhang et al., 2025b)	stage-aware global potential	task text + simulator state/API	privileged-state stage inference	online RL
ARM-FM (Castanyer et al., 2026)	persistent automaton reward	language + environment API + FM reasoning	event-labeling function	compositional RL
RoboStep (ours)	**persistent step-local reward**	**task/API + pretrained VLM; no reward demos**	**semantic gate + reversible rollback**	**online RL**