SPECRL: SPECIFICATION-GUIDED DENSE REWARDS FOR VLA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning fine-tuning (RLFT) has emerged as a promising approach for adapting pretrained vision-language-action models through task-directed interaction. However, many existing methods rely on sparse binary rewards, which provide limited guidance for policy optimization and thereby hinder learning performance. While recent approaches seek denser feedback via LLM-generated reward functions or vision-language model evaluations, they are often inefficient and the resulting rewards are often unreliable, potentially introducing errors that misguide policy optimization. In this work, we propose SPECRL, a specification-guided framework designed to provide dense, interpretable online rewards for RLFT. The core idea is to formalize task requirements in Signal Temporal Logic (STL) and derive rewards from its explicit quantitative semantics. SPECRL first uses an LLM-aided offline process to convert language instructions into STL specifications constituted by task-level atomic propositions. During execution, SPECRL evaluates proposition satisfaction using either a privileged-state grounding module or a lightweight visual grounding module, supporting both simulation and real-world deployments. A runtime monitor then aggregates the quantitative values of propositions along the specification's temporal structure, translating fine-grained task progress into dense rewards. Extensive evaluations on the LIBERO benchmark show that privileged-state SPECRL improves the success rate over binary-reward RLFT by 3.3 percentage points, while visual SPECRL surpasses the strongest visual baseline by 4.8 percentage points with $1.43\times$ faster training. Real-world trajectory evaluations further demonstrate that its deterministic and interpretable design delivers more accurate reward feedback than baselines.

1 INTRODUCTION

While pretrained vision-language-action (VLA) models exhibit impressive capabilities (Kim et al., 2024; Wang et al., 2026; Zhang et al., 2026), adapting them to specific target environments requires post-training. Supervised fine-tuning (SFT) (Liu et al., 2026b) serves as an effective approach for adapting VLA models via the imitation of expert demonstrations, yet it struggles to generalize beyond static training data. To overcome this limitation, reinforcement learning fine-tuning (RLFT) has attracted growing attention. By enabling VLA policies to refine their behaviors through task-directed online interaction, RLFT significantly enhances generalization and robustness beyond what SFT can achieve (Zang et al., 2025; Dong et al., 2026; Liu et al., 2026b; Sun et al., 2026).

Despite its ability to adapt VLA models through interaction, RLFT depends heavily on the quality of the reward signal. Most existing methods rely on sparse binary rewards (Chen et al., 2025a;b; Luo et al., 2025; Zhang et al., 2025a), which indicate whether a task succeeds but provide no feedback on intermediate progress. As a result, the policy has little information about which actions move it closer to completion, making learning especially difficult for complex and long-horizon tasks.

To provide more informative feedback during RLFT, recent work has explored several ways to construct rewards. One line of work (Yu et al., 2023; Xie et al., 2024) utilizes LLMs to synthesize task-specific reward programs based on privileged simulator information. However, delegating the selection of reward components and weights to heuristic generation can lead to misaligned rewards that fail to accurately capture task progress. Another approach (Yang et al., 2024; Wang et al., 2024; Lu et al., 2025; Zhang et al., 2025b; Feng et al., 2026) employs vision-language models (VLMs) to

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

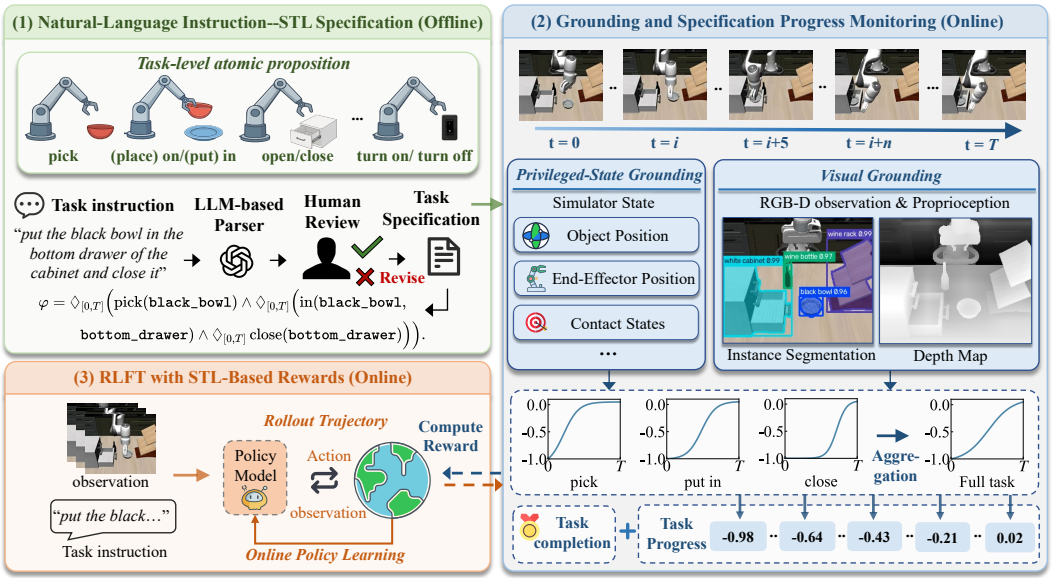


Figure 1: **Overview of the SPECRL framework.** SPECRL constructs dense rewards through two core stages: (1) offline translation of natural-language instructions into JSON task representations encoding STL specifications; and (2) online grounding of task-level atomic propositions and monitoring of specification progress. The resulting dense rewards are then used for online VLA RLFT.

estimate task progress from visual observations. While convenient, VLM-derived rewards are difficult to interpret, which can lead to unreliable and misleading feedback that reinforces unsuccessful behaviors in VLA models. Repeated VLM inference also adds substantial computational cost during RLFT. These limitations raise a central question: *how can we design explicit, deterministic, widely applicable dense rewards without relying on computationally expensive evaluators?*

To answer this question, we present SPECRL, an interpretable framework for dense reward generation that leverages Signal Temporal Logic (STL) (Maler & Nickovic, 2004), as illustrated in Figure 1. By formalizing task structures through STL, our framework utilizes logical operators to enforce the strict execution order of sub-tasks, while employing the STL quantitative semantics to deterministically measure continuous task progress. Specifically, SPECRL operates in two main phases. First, natural-language instructions are translated via an LLM-aided process into structured JSON representations that encode STL specifications. Composed of task-level atomic propositions (e.g., *pick* and *place on*) and their required temporal order, these explicit representations can be manually validated. Second, at runtime, SPECRL evaluates these propositions using either a privileged-state grounding module in simulation or a lightweight visual grounding module in physical environments. Building upon these evaluations, a runtime monitor aggregates fine-grained progress along the required task order into dense rewards to guide online VLA RLFT, allowing intermediate task progress to directly contribute to policy optimization.

Overall, the main contributions of this paper are summarized as follows:

- We propose SPECRL, an interpretable and efficient framework that translates natural-language instructions into formal STL specifications, leveraging their quantitative semantics to derive dense rewards for online model fine-tuning.
- We develop a unified pipeline that evaluates STL specifications using either privileged simulator states or visual observations without incurring heavy computational overhead, where a runtime monitor aggregates fine-grained task progress into dense rewards, thereby providing explicit feedback for intermediate task progress.
- Empirical results show that privileged-state SPECRL exceeds binary-reward RLFT by 3.3 percentage points in success rate, while the visual variant outperforms the strongest baseline by 4.8 percentage points and trains 1.43× faster. Real-world trajectory evaluations further show that its interpretable design provides more accurate reward feedback.

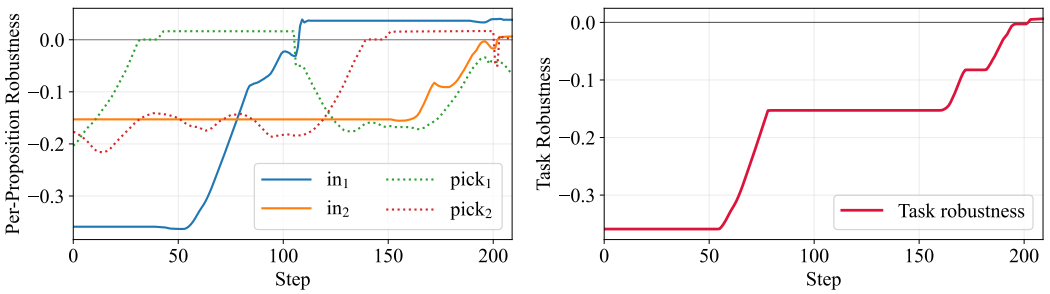


Figure 2: **Illustration of robustness computation in LIBERO.** The task comprises four atomic propositions to be satisfied in the order $\text{pick}_1 \rightarrow \text{in}_1 \rightarrow \text{pick}_2 \rightarrow \text{in}_2$, which can be encoded by the STL specification $\varphi = \Diamond_{[0,T]}\big(\text{pick}_1 \wedge \Diamond_{[0,T]}\big(\text{in}_1 \wedge \Diamond_{[0,T]}\big(\text{pick}_2 \wedge \Diamond_{[0,T]}\,\text{in}_2\big)\big)\big)$. Left: robustness trajectories of atomic propositions. Right: robustness trajectory of the task specification φ.

2 Background and Problem Formulation

Reinforcement Learning Fine-Tuning of VLAs. Let π_θ denote a pretrained VLA policy. Given an instruction $l \in \mathcal{L}$ and a history of observations $o_{\leq t}$, the policy generates $a_t^{(H)} = (a_t, \ldots, a_{t+H-1}) \sim \pi_\theta(\cdot \mid o_{\leq t}, l)$. Here, $H = 1$ corresponds to a single action, while $H > 1$ corresponds to an action chunk. RLFT updates the policy by maximizing the expected cumulative reward obtained through interaction with the environment:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}(l, x_t)\right], \tag{1}$$

where t indexes low-level environment steps, T is the episode horizon, $\gamma \in (0, 1]$ is the discount factor, and x_t denotes the task-relevant information extracted from the observation history $o_{\leq t}$ for reward computation. A common choice in robotic manipulation is a binary success reward, $\mathcal{R}(l, x_t) = \mathbb{I}[\text{Success}(l, x_t)]$, which provides no feedback on intermediate task progress.

Syntax and Quantitative Semantics of Signal Temporal Logic. The syntax of STL is given by:

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_{[t_1,t_2]}\varphi \mid \Box_{[t_1,t_2]}\varphi, \tag{2}$$

where μ is an atomic proposition evaluated by a real-valued predicate function $m(x) : \mathcal{X} \rightarrow \mathbb{R}$, such that μ holds at time t iff $m(x_t) \geq 0$. The bounded interval $[t_1, t_2]$ is measured in environment steps, and $\Diamond$ and $\Box$ are modal operators denoting eventually and always, respectively. Nested temporal operators can encode ordered task requirements. The quantitative semantics (Takayama et al., 2025) assign a robustness value $\rho(\varphi, x, t)$ to formula φ over signal x at time t. Non-negative and negative robustness indicate satisfaction and violation, respectively. The standard min-max robustness semantics are defined recursively as:

$$\begin{aligned}
\rho(\mu, x, t) &= m(x_t), & \rho(\varphi_1 \wedge \varphi_2, x, t) &= \min\{\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)\}, \\
\rho(\neg\varphi, x, t) &= -\rho(\varphi, x, t), & \rho(\varphi_1 \vee \varphi_2, x, t) &= \max\{\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)\}, \\
\rho(\Diamond_{[t_1,t_2]}\varphi, x, t) &= \max_{\tau \in [t+t_1, t+t_2]} \rho(\varphi, x, \tau), & \rho(\Box_{[t_1,t_2]}\varphi, x, t) &= \min_{\tau \in [t+t_1, t+t_2]} \rho(\varphi, x, \tau).
\end{aligned} \tag{3}$$

Figure 2 illustrates an example of STL robustness computation. As the task progresses, initially negative proposition margins become non-negative, while the robustness of φ aggregates them in the temporal order to quantify task progress. However, a potential problem is that varying proposition scales adversely affect their contributions and standard min-max semantics can mask the completion of certain sub-tasks (e.g., variations in in_1 and in_2 fail to contribute to φ).

Problem Formulation. Given a language instruction, our objective is to construct explicit, dense reward signals for VLA RLFT by translating task requirements into an STL specification and leveraging its quantitative semantics for deterministic computation, while ensuring that every sub-task contributes proportionally to the task reward.

3 SPECRL: SPECIFICATION-GUIDED DENSE REWARDS FOR RLFT

As shown in Figure 1, the SPECRL framework operates through a two-stage pipeline: (1) offline instruction parsing for STL specification and (2) online grounding of task-level atomic propositions and monitoring of specification progress for RLFT. The design details are elaborated as follows.

3.1 FROM LANGUAGE INSTRUCTIONS TO STL SPECIFICATIONS

Translating a natural language instruction into an executable task specification requires bridging high-level semantics with formal logic. To this end, we first establish a domain-specific vocabulary of atomic propositions, and subsequently leverage an LLM to parse instructions into structured, temporally ordered JSON representations. These representations are then utilized by the runtime engine to evaluate specification progress under STL semantics.

Domain-specific Atomic Propositions. To formalize task-completion events within a specific domain, we define a closed vocabulary $\mathcal{V}$ of task-level atomic propositions. In the domain of object manipulation using robot arms (such as the tasks of the LIBERO benchmark), we set: $\mathcal{V} = \{\mathrm{pick}, \mathrm{on}, \mathrm{in}, \mathrm{open}, \mathrm{close}, \mathrm{turn_on}, \mathrm{turn_off}\}$. These seven propositions cover all task-completion events in our experiments, with their Boolean satisfaction conditions summarized in Table 4.

LLM-Aided Instruction Parsing. Given an instruction l, SPECRL prompts an LLM with the vocabulary $\mathcal{V}$, atomic proposition definitions, and task-decomposition rules to generate a structured JSON representation that explicitly encodes propositions and enforces strict temporal ordering. To improve reliability and correct potential parsing errors, a one-time manual review is conducted prior to caching to validate consistency with the task instruction and environment semantics. Once cached, SPECRL directly utilizes the JSON representation for runtime monitoring, eliminating online LLM queries or human intervention during RLFT. The structured logic within the JSON corresponds to an STL formula φ, where nested eventually operators specify the execution stage order. For instance, the instruction "*put the black bowl in the bottom drawer of the cabinet and close it*" can be represented as: $\varphi = \Diamond_{[0,T]}\big(\mathrm{pick}(\texttt{black_bowl}) \wedge \Diamond_{[0,T]}\big(\mathrm{in}(\texttt{black_bowl}, \texttt{bottom_drawer}) \wedge \Diamond_{[0,T]}\,\mathrm{close}(\texttt{bottom_drawer})\big)\big)$. Since the exact completion time of each sub-task is unknown, we use the episode horizon T as the common bound for all eventually operators.

3.2 GROUNDING TASK-LEVEL ATOMIC PROPOSITIONS

Precisely evaluating task completion requires computing the satisfaction status of atomic propositions at each time step. To achieve this, we introduce a dedicated grounding mechanism that maps raw observations or states to a signed robustness margin m_i for each proposition μ_i. Moreover, SPECRL implements two observation-specific grounding modules to adapt to both simulation and real-world environments.

Privileged-state Grounding. In simulation environments where ground-truth states are accessible, the privileged module directly leverages simulator-provided object poses, contact states, and joint configurations. To ensure continuous progress feedback rather than sparse binary rewards, SPECRL designs a margin function for each proposition. For instance, the pick primitive $\mathrm{pick}(o)$ outputs a positive margin upon successful grasp contact; otherwise, its margin is defined as $\min\{r_g - \|p_e - p_o\|_2, -\epsilon\}$, where p_e and p_o denote the end-effector and object positions, r_g is an approach threshold, and $\epsilon > 0$. This formulation provides continuous shaping during the approach phase until contact is established, while remaining negative beforehand. The remaining proposition types are constructed similarly, with their full mathematical formulations detailed in Appendix C.

Visual Grounding. To support real-world deployment where privileged states are unavailable, SPECRL implements a lightweight visual grounding module that evaluates atomic propositions using solely RGB-D observations and robot proprioception. Specifically, SPECRL leverages a YOLOE model (Wang et al., 2025) to extract 2D object masks from RGB images, which are then integrated with aligned depth data to reconstruct 3D object geometries within the robot base frame. By computing spatial relationships between these reconstructed structures and the end-effector, the visual module mirrors the privileged-state grounding pipeline, outputting consistent robustness margins. Figure 3 illustrates a representative example of the evaluation for $\mathrm{pick}(\texttt{black_bowl})$ and $\mathrm{on}(\texttt{black_bowl}, \texttt{plate})$, and Appendix C provides details of the implementation.

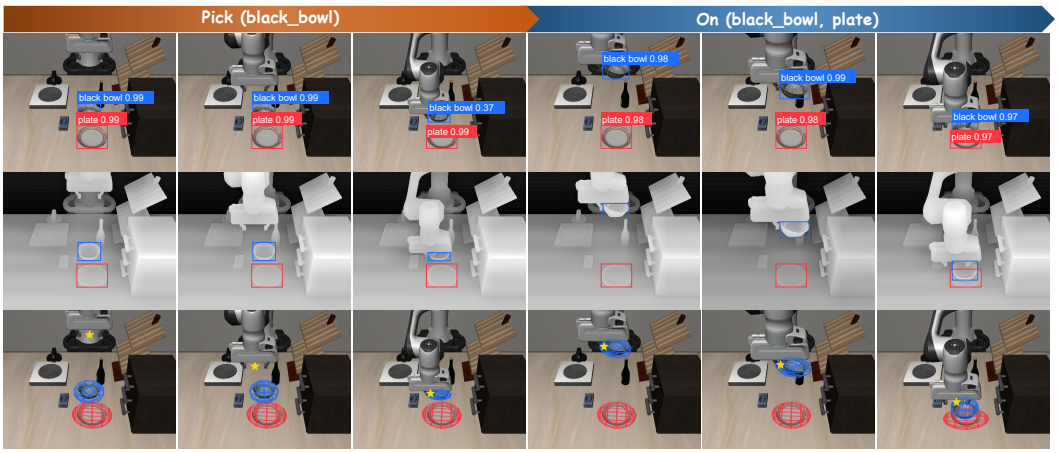


Figure 3: **Visual grounding in SpecRL.** The visual module segments RGB objects via YOLOE (top) and fuses depth maps (middle) to reconstruct 3D geometries (bottom). Distances between these geometries and the gripper center (yellow star) are used to compute the satisfaction of propositions.

3.3 Specification Monitoring and Reward Shaping

To measure progress toward satisfying the full task specification, SpecRL combines grounded condition margins into a task-level measure. As discussed above, standard min-max semantics can mask the completion of certain sub-tasks and varying scales adversely affect their contributions when aggregated directly. Drawing inspiration from prior work (Jothimurugan et al., 2019), we organize the specification into ordered stages and employ a finite-state automaton (FSA) to track stage progression. Specifically, within each stage, normalized condition margins measure local progress, while the FSA records completed stages and advances in the temporal order required by the specification.

Stage-level Progress. Each stage corresponds to a task-level atomic proposition whose satisfaction may depend on multiple constituent conditions. Let $\mathcal{I}_k$ denote the index set of condition margins for stage k. We normalize each initially violated margin, for which $m_i(x_0) < 0$, by its initial magnitude: $\bar{m}_i(t) = m_i(x_t)/|m_i(x_0)|$, $i \in \mathcal{I}_k$. A standard min aggregation enforces the requirement that all conditions hold, but changes in any margin other than the smallest do not affect the stage score. SpecRL instead uses the arithmetic-geometric mean quantitative semantics of STL (Mehdipour et al., 2019), formulating the stage score as $A_k(t) = \left(\prod_{i\in\mathcal{I}_k}(1+\bar{m}_i(t))\right)^{1/|\mathcal{I}_k|} - 1$ if all $\bar{m}_i(t) > 0$, and $A_k(t) = \frac{1}{|\mathcal{I}_k|}\sum_{i\in\mathcal{I}_k}\min\{\bar{m}_i(t), 0\}$ otherwise. The resulting score averages the negative parts of the constituent margins while any condition remains violated.

Specification-level Monitoring. For a specification with K ordered stages, let k_t denote the active stage of the FSA, initialized at $k_0 = 1$. When $A_{k_t}(t) \geq 0$, the FSA advances to the next stage and retains the completion of preceding stages. The task-level progress score combines this discrete progression with the quantitative score of the active stage: $\Phi_\varphi(t) = -\frac{K-k_t}{K} + \frac{\mathrm{clip}(A_{k_t}(t),-1,0)}{K}$ for $k_t \leq K$, and $\Phi_\varphi(t) = 0$ upon completion of the final stage. Each stage occupies a range of $1/K$. The FSA state records how far the execution has advanced through the specification, while the clipped score measures progress toward completing the current stage. For a specification with alternative execution paths, we maintain a separate FSA for each path and apply disjunctive semantics by taking the highest path-level progress score as the task-level score.

Dense reward for RLFT. The task-level progress score $\Phi_\varphi(t)$ summarizes how far execution has advanced toward satisfying the specification. To translate this measure into step-wise feedback, SpecRL defines the final reward using progress-based shaping augmented with a completion bonus:

$$\mathcal{R}_{\text{SpecRL}}(l, x_{\leq t+1}) = \underbrace{\gamma(\Phi_\varphi(t+1)+1) - (\Phi_\varphi(t)+1)}_{\text{specification-progress reward}} + \underbrace{\mathcal{R}^g_{\text{comp}}(l, x_{\leq t+1})}_{\text{task-completion reward}}, \tag{4}$$

where the progress score is shifted by 1 so that $\Phi_\varphi(t) + 1 \in [0, 1]$, thereby avoiding a positive shaping reward when no actual progress is made. $g \in \{\text{priv}, \text{vis}\}$ denotes the grounding regime.

For privileged grounding, $\mathcal{R}^{\text{priv}}_{\text{comp}}$ is the environment success reward. For visual grounding, the FSA monitor supplies the completion signal: $\mathcal{R}^{\text{vis}}_{\text{comp}}(l, x_{\leq t+1}) = \mathbb{I}[\Phi_\varphi(t) < 0 \wedge \Phi_\varphi(t+1) \geq 0]$. Because the FSA retains its completed state, this unit completion reward is issued strictly once at the moment of initial task completion, preventing duplicate issuance.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Experimental Setup. We evaluate SPECRL on the LIBERO benchmark (Liu et al., 2023) and real-world robotic manipulation tasks. All methods are initialized from the same $\pi_{0.5}$ checkpoint obtained through few-shot SFT and fine-tuned using Proximal Policy Optimization (PPO) (Schulman et al., 2017). Model training is conducted on a single server equipped with eight NVIDIA A100 (80GB) GPUs, 88 CPU cores, and 1.58 TB of system memory, while real-world evaluations are deployed on a Franka Research 3 manipulator. Given the heavy overhead of online VLA RLFT, we train models for 400 epochs across all suites, where each epoch corresponds to one PPO update iteration. Ablations and reward comparisons are conducted for 100 epochs on LIBERO-Goal, as its relatively long-horizon nature makes it suitable for evaluating different reward formulations. To further test generalization under out-of-distribution (OOD) scenarios, we evaluate the performance of the trained models using the LIBERO-Plus suite. Other implementation details, including model parameter configurations and training settings, are provided in Appendix A.

Evaluation Metric. We report the task success rate (SR) over N_{eval} evaluation episodes for each task suite, defined as the percentage of successful trial trajectories.

Baselines. We compare SPECRL against several SOTA reward baselines: (1) **Text2Reward** (Xie et al., 2024), which leverages LLMs to generate task-specific rewards using privileged states; (2) **Robo-Dopamine** (Tan et al., 2025), which estimates task progress via a step-aware VLM-based reward model with multi-view visual inputs; (3) **TOPReward** (Chen et al., 2026), which derives zero-shot progress rewards from the token probabilities of a VLM model; (4) **Robometer** (Liang et al., 2026), which evaluates step-wise visual rewards by assessing both task progress and task completion via a fine-tuned VLM. In addition, we use **SPECRL-P** and **SPECRL-V** to denote our framework instantiated with privileged-state and visual grounding, respectively.

4.2 RLFT PERFORMANCE WITH PRIVILEGED-STATE GROUNDING

Comparison with Binary-Reward RLFT. As shown in Figure 4(a) and (b), SPECRL-P outperforms binary-reward RLFT across all four LIBERO suites, achieving faster convergence, more stable policy improvement during training, and an increased overall mean SR from 89.3% to 92.6%. The largest gains occur on the more challenging LIBERO-Goal and LIBERO-Long suites, with improvements of 7.2 and 4.8 percentage points, respectively, while gains on Spatial and Object are smaller. These results suggest that dense progress feedback is beneficial for RLFT improvement on complex tasks, especially those with longer execution horizons.

Comparison with Privileged SOTA Baselines. We compare SPECRL-P with Text2Reward and binary-reward RLFT on LIBERO-Goal. From Figure 5(a), it can be observed that SPECRL-P continues to improve and achieves the highest final success rate, while Text2Reward plateaus at a lower level and gains from binary-reward RLFT diminish earlier. The test results in Figure 8 and the OOD evaluation in Table 3 further indicate that SPECRL-P exceeds Text2Reward by 3.8 and 4.98 percentage points, respectively. These results suggest that reward density alone is insufficient for effective RLFT; how rewards represent task progress also matters. LLM-generated rewards rely on heuristic terms and weights that may not faithfully reflect task requirements, leading to inconsistent feedback on task progress. In contrast, by explicitly encoding task conditions and their temporal order, SPECRL delivers more reliable rewards for policy optimization.

4.3 RLFT PERFORMANCE WITH VISUAL GROUNDING

Qualitative Analysis of Reward Predictions over SOTA Baselines. We compare visual reward predictions on demonstration trajectories for three LIBERO tasks reproduced in a real-world set-

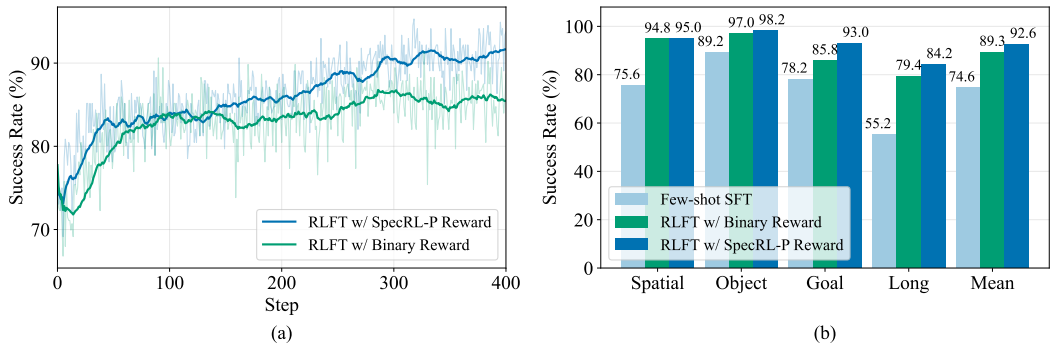


Figure 4: **RLFT performance with privileged-state grounding across the four LIBERO task suites.** (a) Learning curves across the four task suites. (b) Evaluation success rate comparison.

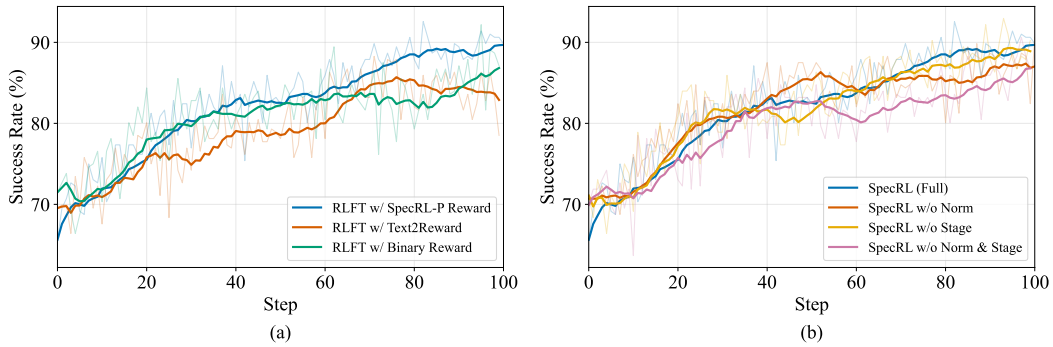


Figure 5: **Reward comparison and ablation analysis on the LIBERO-Goal suite.** (a) Learning curves of SPECRL-P, Text2Reward, and Binary rewards. (b) Ablation results, where Norm and Stage denote margin normalization and stage-wise progress monitoring, respectively.

ting, with the results illustrated in Figure 6. Notably, SPECRL-V demonstrates a strong correlation with task success and failure, whereas this is not the case for the baselines. As observed, it more accurately reflects task progress and outcomes in the evaluated real-world trajectories. Driven by explicit and deterministic reward computation, SPECRL-V yields progress scores that increase as the required sub-tasks are completed and remain low for demonstrations that do not satisfy the task instructions. In contrast, the visual baselines produce noisy progress estimates, show premature increases, or assign high scores to unsuccessful executions (picking incorrect objects, misplacements). These predictions do not faithfully reflect the observed task outcomes. Such misleading feedback may hinder policy improvement and reinforce incorrect behaviors during online RLFT.

Comparison with Visual SOTA Baselines. We evaluate SPECRL-V against visual baselines across three aspects. **(1) RLFT Performance.** SPECRL-V achieves superior performance in visual-based RLFT comparisons. Specifically, as presented in Figure 7(a), SPECRL-V surpasses the three visual baselines after epoch 30 and converges to the highest performance. In contrast, Robo-Dopamine and Robometer plateau earlier at lower success rates. Furthermore, as illustrated in Figure 8, SPECRL-V outperforms TOPReward, Robo-Dopamine, and Robometer by 4.8, 7.2, and 15.6 percentage points, respectively. Results from the OOD evaluation in Table 3 further show that SPECRL-V achieves an SR exceeding the three visual baselines by at least 1.97 percentage points. As discussed earlier, these results are consistent with the more informative and structured progress signals provided by SPECRL-V, which effectively facilitate RLFT. **(2) Training Efficiency.** From Figure 7(b), it can be seen that SPECRL-V demonstrates competitive training efficiency, with only a 34.8% increase in training time compared to SPECRL-P. In contrast, visual baselines suffer from prohibitively long training durations. This is attributed to the heavy computational overhead incurred by frequent VLM inference for reward scoring during online RL. By avoiding such inference, SPECRL-V achieves training speedups of $1.43\times$, $2.46\times$, and $2.37\times$ over TOPReward, Robo-Dopamine, and Robometer,

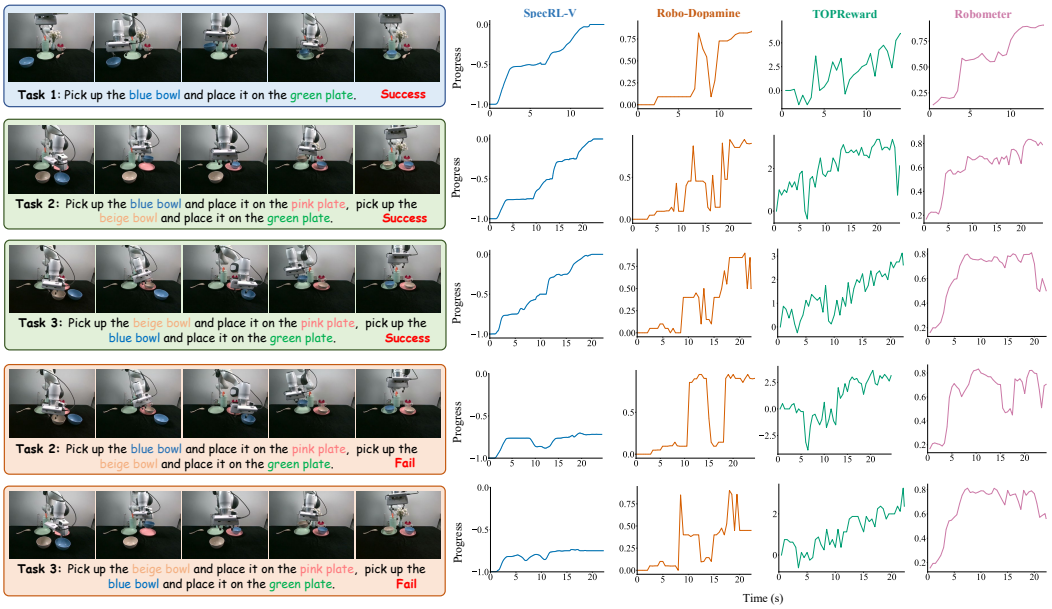


Figure 6: **Qualitative analysis of real-world LIBERO demonstration trajectories.** The first three trajectories demonstrate successful executions; the last two fail to match the task descriptions. SPECRL-V produces task progress that closely reflects task outcomes, with rewards steadily increasing during successful execution and remaining low for mismatched demonstrations. In contrast, the reward curves of Robo-Dopamine and TOPReward are unstable, exhibiting high fluctuations and erroneously assigning high reward scores to the failed demonstrations. Moreover, Robometer assigns high rewards prematurely and also fails to distinguish failure cases by giving high scores.

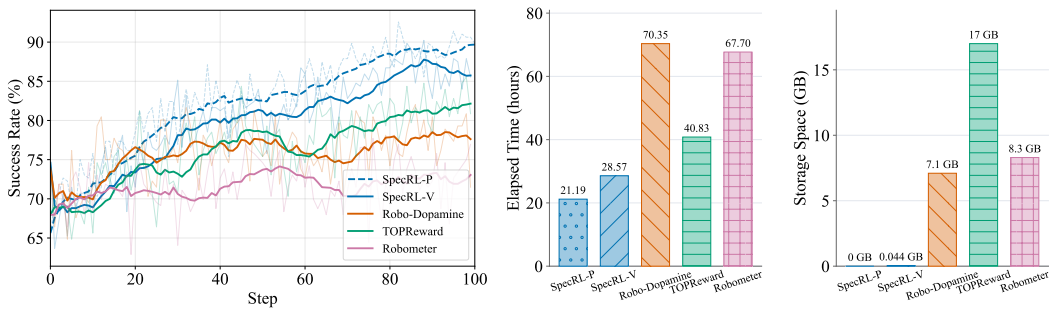


Figure 7: **Comparison of RLFT performance and resource requirements across reward methods**. (a) Success rate during RLFT. (b) RLFT training time. (c) Storage requirements.

respectively. **(3) Storage Requirements.** SPECRL-V requires only 44 MB of model storage, compared to 7.1–17 GB for visual reward baselines, as indicated by Figure 7(c). In fact, beyond disk storage, keeping VLM evaluators in GPU memory alongside VLA policies adds substantial memory overhead during RLFT. By avoiding the need for large VLM evaluators, SPECRL-V alleviates this overhead, making deployment and execution viable in resource-limited environments.

Visual versus Privileged-State Grounding. As presented in Figure 7(a), both SPECRL-V and SPECRL-P exhibit consistent performance improvements, with the visual variant slightly behind its privileged counterpart. Furthermore, in-distribution evaluation results in Figure 8 and OOD evaluation results in Table 3 indicate that SPECRL-P outperforms SPECRL-V by 2.2 and 1.31 percentage points, respectively. This performance gap arises because the visual branch suffers from state approximations and occlusions, while the privileged branch leverages exact ground-truth information.

4.4 Ablation of SpecRL

We further conduct ablation studies under privileged-state grounding to examine the contributions of margin normalization and stage-wise progress monitoring. As shown in Figure 5(b) and Figure 9, the SpecRL variant without both components performs worst among the evaluated variants. Although SpecRL w/o Norm and SpecRL w/o Stage slightly outperform the full method early in training, their gains diminish later, and both converge to lower success rates. The full method achieves a final test SR at least 1.8 percentage points higher than all three ablated variants. Margin normalization has a larger impact, as varying scales across proposition margins can distort their relative contributions to the aggregated reward and weaken the resulting progress feedback.

5 Related Work

Reinforcement Learning for VLAs. Although pretrained VLA models demonstrate remarkable generalization (Kim et al., 2024; Yang et al., 2026), adapting them to target environments necessitates post-training. While SFT adapts models to new tasks using expert demonstrations (Kim et al., 2025), its effectiveness is constrained by the quality and coverage of static offline data. RLFT mitigates this limitation via environmental interaction (Lyu et al., 2025; Liu et al., 2026b), optimizing task returns to recover from errors and learn robust behaviors beyond demonstrations (Dong et al., 2026). Driven by recent breakthroughs in VLA optimization frameworks (Li et al., 2025; Zang et al., 2025; Liu et al., 2026a; Li et al., 2026; Liu et al., 2026b), RLFT has rapidly emerged as a prominent paradigm for downstream adaptation. Nevertheless, most existing methods rely on sparse, binary rewards that offer limited intermediate feedback and struggle to guide exploration in long-horizon tasks, motivating a growing interest in dense reward construction.

Reward Construction for VLA Reinforcement Learning. Constructing informative rewards is important for effective RLFT. Inverse reinforcement learning (Jiang et al., 2025; Jain et al., 2026) infers reward functions from expert demonstrations, yet its heavy reliance on extensive data limits its applicability. Beyond demonstration-based methods, recent approaches exploit privileged simulator states. Representative approaches (Yu et al., 2023; Xie et al., 2024) leverage language models to translate high-level task descriptions into executable reward code. However, delegating reward design to LLMs makes it difficult to guarantee that the generated rewards faithfully capture task requirements and leaves the process vulnerable to hallucinations. Another line of work derives rewards directly from visual observations, using either pretrained VLMs for zero-shot reward estimation (Cui et al., 2022; Fu et al., 2024; Rocamonde et al., 2024; Chen et al., 2026), or models adapted with additional supervision to estimate task progress (Yang et al., 2024; Tan et al., 2025; Lee et al., 2026; Liang et al., 2026). Although these methods avoid privileged simulator states and can generalize to the real world, their reward semantics often remain implicit within neural evaluators, making the feedback difficult to interpret. Misrecognized physical events may produce misleading rewards that reinforce unsuccessful behaviors, while repeated VLM inference introduces substantial computational overhead during RLFT. SpecRL overcomes these limitations by grounding explicit STL task specifications into either privileged states or visual observations, yielding interpretable, dense rewards through rigorous quantitative monitoring while avoiding heavy computational overhead.

6 Conclusion

In this work, we introduce SpecRL, a specification-guided framework designed to provide dense online rewards for VLA RLFT. It first translates language instructions into STL specifications that encode task-level atomic propositions and their required temporal order. During execution, SpecRL evaluates proposition satisfaction using either privileged-state or visual grounding modules, after which a monitor combines proposition-level feedback with the required task order to generate well-calibrated rewards. Experimental results show that both the privileged-state and visual SpecRL improve task success rates over their respective baselines. Moreover, real-world trajectory evaluations confirm that SpecRL provides more accurate reward feedback than competing visual baselines. Overall, SpecRL demonstrates that translating language instructions into STL specifications provides an explicit and deterministic basis for computing interpretable rewards from both privileged states and visual observations. We hope this work will spur further research on specification-guided reward design, particularly on extending reusable task specifications to broader VLA RLFT settings.

AI USE STATEMENT

In this work, we used generative AI tools only for language polishing, readability improvements, limited sentence rephrasing, and suggestions on manuscript organization. We did not use generative AI to generate core research ideas, design experiments, conduct data analysis, derive mathematical results, or determine the final research claims. All AI-generated or polished text was carefully checked, verified, and edited by the authors to ensure accuracy, academic integrity, and correct interpretation of our results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide detailed descriptions of our proposed method, model configurations, training settings, and evaluation protocols in the main paper and appendix. Following the official guidelines for code submission, we will provide an anonymized code repository through an OpenReview comment to the reviewers and area chairs once the discussion forums are opened. The codebase will also be made publicly available upon acceptance of this paper.

REFERENCES

Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, et al. πrl: Online rl fine-tuning for flow-based vision-language-action models. *arXiv preprint arXiv:2510.25889*, 8, 2025a.

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*, 2026.

Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. Conrft: A reinforced fine-tuning method for vla models via consistency policy. *arXiv preprint arXiv:2502.05450*, 2025b.

Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for dynamics and control conference*, pp. 893–905. PMLR, 2022.

Perry Dong, Kuo-Han Hung, Tian Gao, Dorsa Sadigh, and Chelsea Finn. Expo-ft: Sample-efficient reinforcement learning finetuning for vision-language-action models. *arXiv preprint arXiv:2605.25477*, 2026.

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. *arXiv preprint arXiv:2605.08774*, 2026.

Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. FuRL: Visual-language models as fuzzy rewards for reinforcement learning. In *International Conference on Machine Learning*, pp. 14256–14274. PMLR, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. pp. 17–40, 2025.

Arnav Kumar Jain, Vibhakar Mohta, Subin Kim, Atiksh Bhardwaj, Juntao Ren, Yunhai Feng, Sanjiban Choudhury, and Gokul Swamy. A smooth sea never made a skilled sailor: Robust imitation via learning to search. *Advances in Neural Information Processing Systems*, 38:62676–62707, 2026.

Anqing Jiang, Yu Gao, Yiru Wang, Zhigang Sun, Shuo Wang, Yuwen Heng, Hao Sun, Shichen Tang, Lijuan Zhu, Jinhao Chai, et al. Irl-vla: Training an vision-language-action policy via reward world model. *arXiv preprint arXiv:2508.06571*, 2025.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. Roboreward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Yang Zhaohui, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. Simplevla-rl: Scaling vla training via reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 125846–125869, 2026.

Hengtao Li, Pengxiang Ding, Runze Suo, Yihao Wang, Zirui Ge, Dongyuan Zang, Kexian Yu, Mingyang Sun, Hongyin Zhang, Donglin Wang, et al. Vla-rft: Vision-language-action reinforcement fine-tuning with verified rewards in world simulators. *arXiv preprint arXiv:2510.00406*, 2025.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S Huang, Luke Zettlemoyer, Dieter Fox, et al. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Changyu Liu, Yiyang Liu, Taowen Wang, Qiao Zhuang, James Chenhao Liang, Wenhao Yang, Renjing Xu, Qifan Wang, Dongfang Liu, and Cheng Han. On-the-fly vla adaptation via test-time reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40107–40125, 2026a.

Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can rl bring to vla generalization? an empirical study. *Advances in Neural Information Processing Systems*, 38:97121–97151, 2026b.

Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. Vla-rl: Towards masterful and general robotic manipulation with scalable reinforcement learning. *arXiv preprint arXiv:2505.18719*, 2025.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

Mingyang Lyu, Yinqian Sun, Erliang Lin, Huangrui Li, Ruolin Chen, Feifei Zhao, and Yi Zeng. Reinforcement fine-tuning of flow-matching policies for vision-language-action models. *arXiv preprint arXiv:2510.09976*, 2025.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International symposium on formal techniques in real-time and fault-tolerant systems*, pp. 152–166. Springer, 2004.

Noushin Mehdipour, Cristian-Ioan Vasile, and Calin Belta. Arithmetic-geometric mean robustness for control from signal temporal logic specifications. In *2019 American Control Conference (ACC)*, pp. 1690–1695. IEEE, 2019.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 28446–28463, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Haoran Sun, Yongjian Guo, Zhong Guan, Shuai Di, Xiaodong Bai, Jing Long, Tianyun Zhao, Mingxi Luo, Hongke Zhao, Likang Wu, et al. Rl-vla^3: A flexible and asynchronous reinforcement learning framework for vla training. *arXiv preprint arXiv:2602.05765*, 2026.

Yoshinari Takayama, Kazumune Hashimoto, and Toshiyuki Ohtsuka. Stlccp: Efficient convex optimization-based framework for signal temporal logic specifications. *IEEE Transactions on Automatic Control*, 2025.

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, et al. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025.

Ao Wang, Lihao Liu, Hui Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yoloe: Real-time seeing anything. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 24591–24602. IEEE, 2025.

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*, pp. 51484–51501. PMLR, 2024.

Yuqi Wang, Xinghang Li, Wenxuan Wang, Junbo Zhang, Yingyan Li, Yuntao Chen, Xinlong Wang, and Zhaoxiang Zhang. Unified vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 80929–80944, 2026.

Tianbao Xie, Siheng Zhao, Chen Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 35663–35699, 2024.

Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4804–4811. IEEE, 2024.

Shuai Yang, Hao Li, Bin Wang, Yilun Chen, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. Vision-language-action instruction tuning: From understanding to manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 152896–152943, 2026.

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*, pp. 374–404. PMLR, 2023.

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Peihong Wang, Liangzhi Shi, Yuqing Xie, et al. Rlinf-vla: A unified and efficient framework for reinforcement learning of vision-language-action models. *arXiv preprint arXiv:2510.06710*, 2025.

Hongyin Zhang, Zifeng Zhuang, Han Zhao, Pengxiang Ding, Hongchao Lu, and Donglin Wang. ReinboT: Amplifying robot visual-language manipulation with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 77254–77271. PMLR, 2025a.

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 *Proceedings of The 9th Conference on Robot Learning*, pp. 460–488. PMLR, 2025b.

Wenyao Zhang, Hongsi Liu, Zekun Qi, Yunnan Wang, Xinqiang Yu, Jiazhao Zhang, Runpei Dong, Jiawei He, He Wang, Zhizheng Zhang, et al. Dreamvla: a vision-language-action model dreamed with comprehensive world knowledge. *Advances in Neural Information Processing Systems*, 38: 24195–24228, 2026.

Appendix

The supplementary material presents additional technical details and experimental results for SPECRL, organized as follows:

- **Section A**: Additional Experimental Setup and Implementation Details.
- **Section B**: Additional Experimental Results.
- **Section C**: Additional Technical Details.
- **Section D**: LLM Parsing Examples and Prompt Template.

A ADDITIONAL EXPERIMENTAL SETUP AND IMPLEMENTATION DETAILS

Experimental Setup and Infrastructure. All models are trained using the following configuration:

- **Hardware (Robot & Sensors):** A Franka Research 3 robotic manipulator, equipped with an Intel RealSense D455 global camera and a ZED Mini wrist-mounted (hand-eye) camera.
- **Hardware (Computing):** A single server featuring $8\times$ NVIDIA A100 (80GB) GPUs, 88 CPU cores, and 1.58 TB system RAM.
- **Software:** Ubuntu 22.04.4 LTS, Python 3.11.14, and PyTorch 2.6.
- **Random Seed:** Due to the heavy computational burden of VLA RLFT, we follow common practice in large-scale model learning by using a fixed random seed of 2026.

Base Model. We adopt $\pi_{0.5}$ (Intelligence et al., 2025) as our base VLA model. Specifically, $\pi_{0.5}$ is a 3.3B-parameter generalist policy pretrained on heterogeneous multi-robot demonstrations and vision-language corpora. Its architecture consists of three core components: (1) a SigLIP vision encoder ($\sim$400M parameters) processing camera observations; (2) a Gemma language backbone ($\sim$2.6B parameters) fusing visual and linguistic modalities; and (3) an action expert ($\sim$300M parameters) that outputs continuous action chunks via flow matching. As a widely adopted open-source VLA model, $\pi_{0.5}$ serves as an ideal testbed for evaluating our reward designs in VLA RLFT.

Supervised Fine-Tuning Setup. In our experiments, we use few-shot SFT checkpoints to provide a warm-start initialization for RLFT. Specifically, the complete LIBERO demonstration dataset contains 1,692 trajectories, from which we sample 120 trajectories for each of the Spatial, Object, and Goal task suites and 200 trajectories for the Long suite to construct the corresponding SFT training sets. We then fine-tune all parameters of the pretrained $\pi_{0.5}$ model for 30,000 steps on each training set, following the official SFT configuration. The resulting checkpoints serve as the warm-start initialization for subsequent RLFT. Detailed hyperparameters are provided in Table 1.

Reinforcement Learning Fine-tuning Setup. We implement SPECRL using the RLinf codebase (Zang et al., 2025; Chen et al., 2025a). Following standard practice during RLFT, we freeze the VLM parameters and fine-tune the 300M-parameter action expert. We optimize the policy with PPO, a widely used online RL algorithm, using the default hyperparameter settings from RLinf. Table 2 summarizes the training configuration, PPO parameters, and environment interaction settings for all LIBERO task suites. Given the computational cost of online VLA RLFT, we train policies for 400 epochs on each suite and conduct ablations and reward baseline comparisons for 100 epochs on LIBERO-Goal. Note that the interaction horizon is set to 320 steps for LIBERO-Goal and 400 steps when training across all suites, owing to the longer-horizon tasks in LIBERO-Long. When training exclusively on LIBERO-Spatial or LIBERO-Object, 240 interaction steps are sufficient. Furthermore, during evaluation, a lower noise level of 0.3 is adopted. Each LIBERO task suite is evaluated over 500 episodes, whereas the LIBERO-Plus Goal OOD evaluation comprises 2,591 episodes.

YOLOE Training Details. We use YOLOE-11m-seg (Wang et al., 2025) to segment task-relevant objects and obtain masks for estimating their geometric boundaries. We fine-tune separate models for the LIBERO simulation and real-world settings. Specifically, for the simulation environment, we sample 11,519 frames from demonstration trajectories, where training masks are generated offline using MuJoCo's instance segmentation renderer and instance names are mapped to 26 object classes. The original 256×256 images are resized and padded to 640×640 for training. In contrast, for the

Table 1: Hyperparameters for the SFT stage on the LIBERO task suites.

Parameters	**Value**
Train steps	30000
Save interval	5000
Global batch size	128
Micro batch size (per GPU)	8
Gradient accumulation	4
Actor lr	5e-5
Optimizer	AdamW (β_1=0.9, β_2=0.95, ϵ=1e-8)
Weight decay	1e-10
Grad clip	1.0
LR scheduler	True
Fine-tuning mode	Full-parameter
Action prediction horizon H	10
Denoise steps	5
Noise level σ (Flow-SDE)	0.5
Action dimension	7

Table 2: Hyperparameters for the RLFT stage on the LIBERO task suites.

Parameters	**Goal Suite**	**All Suites**
Train epochs	100	400
Global batch size	2048	2048
Update epochs	4	4
Actor lr	5e-6	5e-6
Critic lr	1e-4	1e-4
Scheduler	True	True
Reward discount rate γ	0.99	0.99
GAE λ	0.95	0.95
Clip ratio ϵ	0.2	0.2
Interaction steps	320	400
Parallel environments	128	128
Rollout epochs	2	2
Action prediction horizon H	10	10
Action replan horizon H'	10	10
Denoise steps	5	5
Noise level σ (Flow-SDE)	0.5	0.5

real-world setting, we sample 220 demonstration frames, map instance names to four object classes, and use an input resolution of 960×960. Both models are fine-tuned for 80 epochs with a learning rate of 1×10^{-2}, requiring less than one hour of fine-tuning in total.

Baseline Implementation Details. We compare SPECRL against several SOTA reward baselines. The implementation details for each baseline are as follows:

- **Text2Reward** (Xie et al., 2024): Leverages GPT-4o to generate task-specific reward functions using privileged states.
- **Robo-Dopamine** (Tan et al., 2025): We use the official implementation, which is based on Qwen3-VL-4B-Instruct. Following the original paper, the step-aware reward model estimates task progress from multi-view visual inputs comprising six frames at a resolution of 224×224. Task completion is detected when the estimated progress reaches 0.95.
- **TOPReward** (Chen et al., 2026): We adopt the official implementation powered by Qwen3-VL-8B-Instruct, which derives zero-shot progress rewards from the pretrained

VLM's token probabilities. To adapt it to our computational setup, we provide up to 15 frames at a reduced resolution of 128×128; running at the original 256×256 resolution would incur a prohibitive computational overhead exceeding 100 hours on our 8-GPU A100 server. For the real-world trajectory evaluation, we retain the original 256×256 input resolution used by the official implementation.

- **Robometer** (Liang et al., 2026): We run the official implementation based on Qwen3-VL-4B-Instruct. Following the original paper, the fine-tuned VLM simultaneously predicts both task progress and task completion from up to 16 frames at a resolution of 256×256, and task completion is determined when its success score exceeds 0.6.

B Additional Experimental Results

Evaluation Results of Trained Models. Figure 8 compares the SR of SpecRL against baseline methods. Among methods utilizing privileged information, SpecRL-P surpasses Text2Reward by 3.8 percentage points. Among methods relying on visual information, SpecRL-V outperforms TOPReward, Robo-Dopamine, and Robometer by 4.8, 7.2, and 15.6 percentage points, respectively. Figure 9 compares the SR of full SpecRL and three ablated variants. Removing both margin normalization and stage-wise monitoring yields the lowest SR, while the full method achieves a final test SR at least 1.8 percentage points higher than each ablated variant.

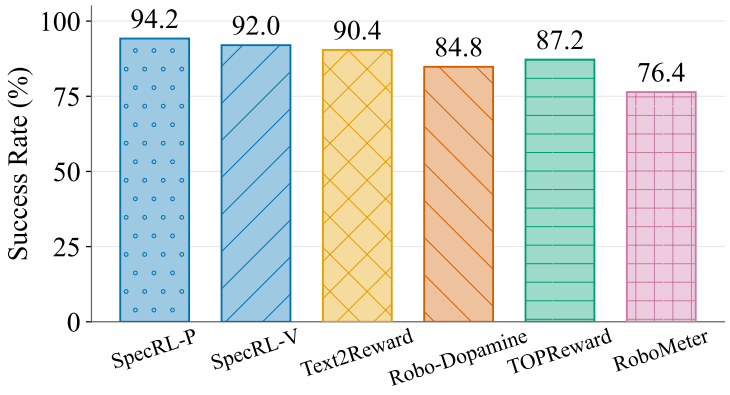


Figure 8: Evaluation success rates of SpecRL and reward baselines.

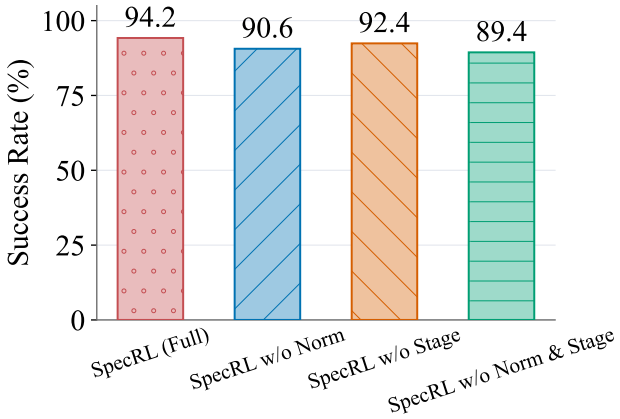


Figure 9: Evaluation success rates of SpecRL and its variants.

Out-of-Distribution Evaluation Results. In this subsection, we assess the robustness of the trained models against environmental variations using the LIBERO-Plus benchmark. As an extension of the

standard LIBERO environment, LIBERO-Plus introduces perturbations across seven physical and semantic dimensions. Our evaluation on LIBERO-Plus Goal comprises 2,591 episodes covering background textures (281), camera viewpoints (408), language instructions (410), object layouts (425), lighting conditions (279), robot initial states (409), and sensor noise (379).

Table 3 presents the evaluation results on the LIBERO-Plus Goal task suite. Among methods using privileged information for reward computation, SPECRL-P outperforms Text2Reward by 4.98 percentage points. Among visual reward methods, SPECRL-V exceeds TOPReward, Robo-Dopamine, and Robometer by 1.97, 8.34, and 14.86 percentage points, respectively. SPECRL-P also achieves an overall success rate that is 1.31 percentage points higher than SPECRL-V. Across individual perturbation dimensions, SPECRL-P achieves the highest performance on all dimensions except language instructions, while SPECRL-V obtains second-best results across the majority of perturbation dimensions (5 out of 7).

Table 3: Out-of-Distribution evaluation results under different perturbations. **Bold** and underlining indicate the best and second-best results in each row, respectively.

Perturbation Type	Text2Reward	TOPReward	Robo-Dopamine	Robometer	SPECRL-V	SPECRL-P
Background Texture	64.41	62.99	59.79	50.18	<u>71.53</u>	**71.89**
Camera View	55.15	53.68	48.04	45.10	<u>57.84</u>	**58.82**
Language instruction	39.02	**43.90**	39.76	34.63	<u>42.93</u>	<u>42.93</u>
Lighting	67.74	65.23	63.08	45.16	<u>72.76</u>	**73.12**
Object Layout	50.59	<u>52.71</u>	47.53	39.76	51.76	**54.35**
Robot Initial State	53.06	53.79	43.52	41.56	<u>55.50</u>	**56.97**
Sensor Noise	50.66	<u>58.31</u>	46.17	41.42	55.67	**58.58**
Overall	53.22	54.92	48.55	42.03	<u>56.89</u>	**58.20**

Additional Analysis of Training Efficiency. RLFT with SPECRL-P incurs no appreciable increase in training time compared with binary rewards, as both methods fall within the 21 to 22-hour training duration range. This is because SPECRL-P computes rewards directly from privileged environment observations, eliminating the need for additional model inference time. As shown in Figure 7(b), SPECRL-V takes 34.8% longer to train than SPECRL-P. Part of this overhead arises from the current implementation: conflicting dependencies prevent VLA and YOLOE inference from running in the same Python environment, so the two environments communicate over HTTP during reward computation. This communication accounts for more than 40% of the additional training time. In contrast, TOPReward, Robo-Dopamine, and Robometer run within the same Python environment as VLA inference and therefore incur no such communication overhead. Resolving the dependency conflict could therefore further reduce the training time of SPECRL-V.

Visualization of Visual Grounding and Robustness Trajectories.

Figure 10 illustrates how SPECRL-V evaluates Task 1 ("Pick up the blue bowl and place it on the green plate"). The instruction is parsed into two task-level atomic propositions. SPECRL-V segments the relevant objects in RGB images using YOLOE and combines the segmentation masks with depth maps to reconstruct their 3D geometry. Spatial relationships among the objects and the gripper center, marked by a yellow star, are then used to evaluate the propositions. Figure 11 shows the resulting proposition robustness values and aggregated task-level robustness over time. As execution progresses, the task-level robustness increases from a negative value toward zero.

Figure 12 shows the same visual grounding and reward computation process for Task 2 ("Pick up the blue bowl and place it on the pink plate, pick up the beige bowl and place it on the green plate"). This instruction is parsed into four task-level atomic propositions. YOLOE segmentation and depth-based 3D reconstruction provide the spatial information needed to evaluate each proposition, including distances to the gripper center marked by a yellow star. Figure 13 traces the proposition robustness values and the aggregated task-level robustness as the two placements are carried out.

Figure 14 presents Task 3 ("Pick up the beige bowl and place it on the pink plate, pick up the blue bowl and place it on the green plate"), which is likewise parsed into four task-level atomic propositions. Figure 15 shows how the proposition robustness values contribute to task-level robustness, which rises toward zero as the ordered task progresses.

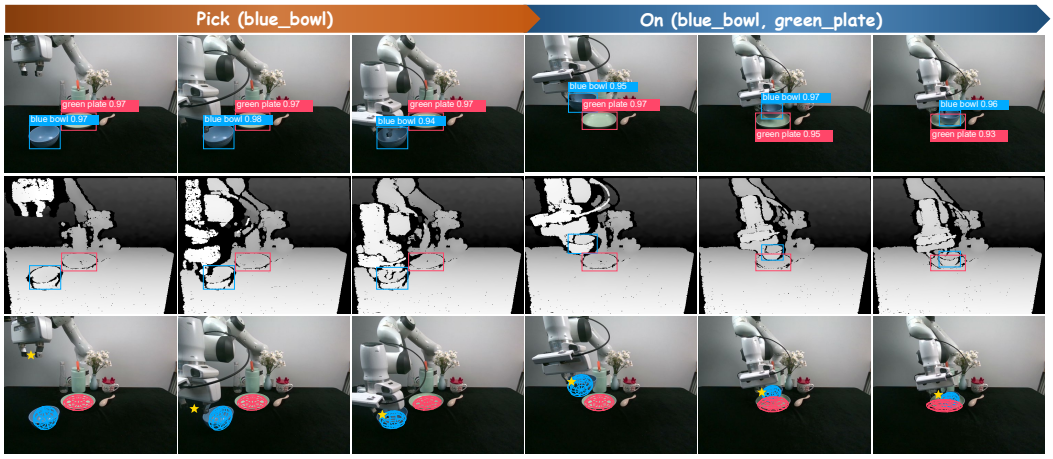


Figure 10: Visualization of the reward calculation process for Task 1 ("Pick up the blue bowl and place it on the green plate") using SPECRL-V.

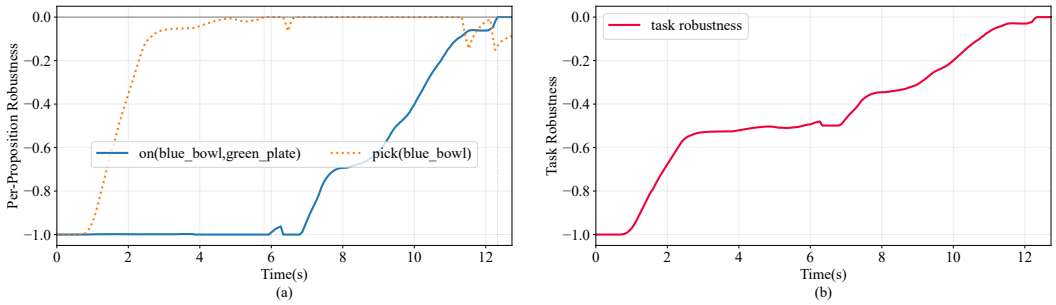


Figure 11: Robustness trajectories of the task-level atomic propositions and the aggregated task-level robustness for Task 1.

C ADDITIONAL TECHNICAL DETAILS

In this section, we first detail the implementation of privileged-state grounding and visual grounding within SPECRL. We then elaborate on the technical specifics of 3D geometry estimation from RGB-D observations, and finally describe the PPO optimization framework for the flow-noise policy.

C.1 PRIVILEGED-STATE GROUNDING

The privileged-state module grounds task-level atomic propositions using simulator-provided object and end-effector poses, contact information, and articulated-object states. Each proposition is evaluated through geometric or state-dependent conditions expressed as signed margins relative to satisfaction thresholds. The resulting proposition margin is non-negative when all conditions are satisfied and negative otherwise, providing dense feedback as the system approaches satisfaction. For brevity, we use AGM to denote the arithmetic-geometric mean aggregation.

The following definitions instantiate our design for seven types of task-level atomic propositions. Note that these implementations provide a workable choice for our benchmark, rather than a unique or necessarily optimal design. **The central principle of SPECRL is to derive dense, interpretable online rewards from explicit task specifications through deterministic grounding and monitoring.** Within this framework, users can define propositions suited to new tasks or domains. Geometric thresholds and other hyperparameters can be tuned through lightweight trials that track task completion in simulation. Although this requires some initial setup, the resulting definitions can be reused across similar post-training tasks, while the significant performance gains achieved during online RLFT help justify the upfront engineering effort.

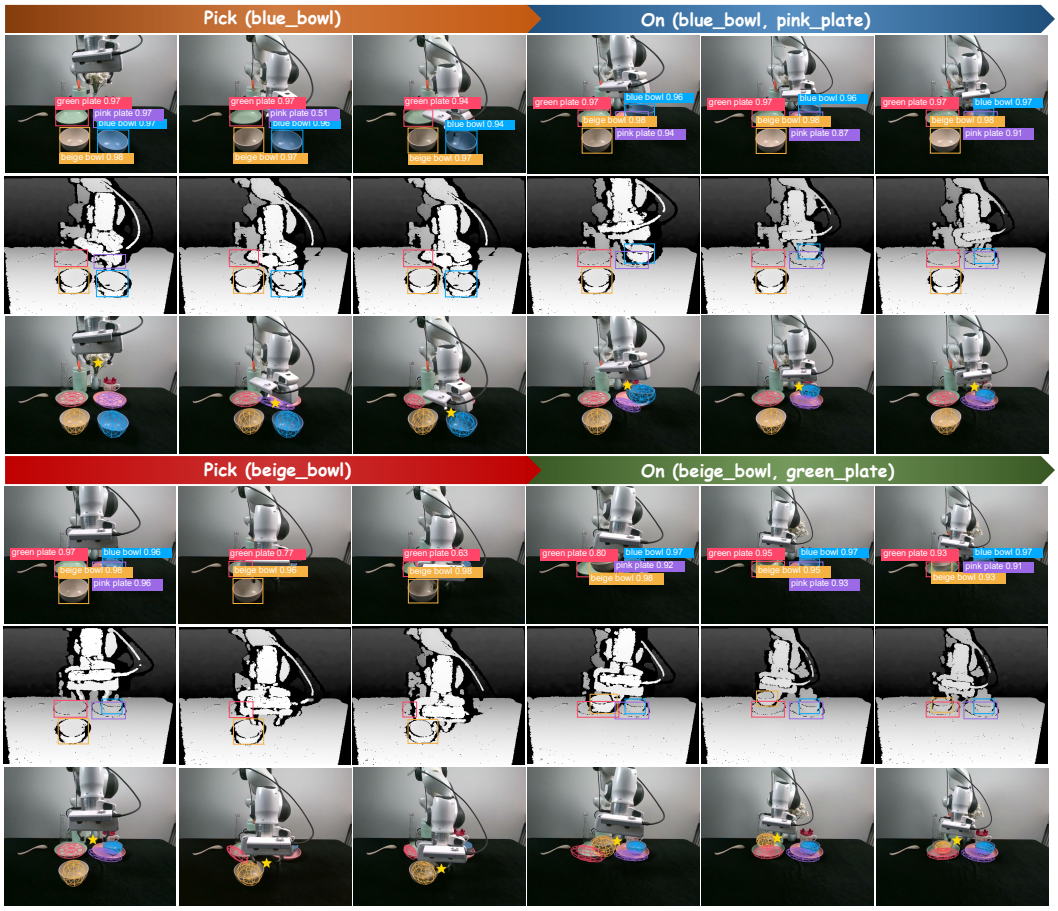


Figure 12: Visualization of the reward calculation process for Task 2 ("Pick up the blue bowl and place it on the pink plate, pick up the beige bowl and place it on the green plate") using SPECRL-V.

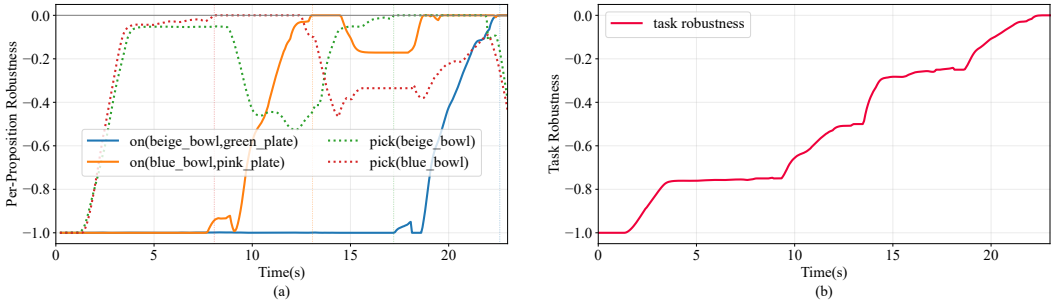


Figure 13: Robustness trajectories of the task-level atomic propositions and the aggregated task-level robustness for Task 2.

- $\mathrm{pick}(o)$: evaluates whether the robot successfully grasps object o. The success condition requires simultaneous contact between the object and both gripper fingers, denoted as $C_L(o) \wedge C_R(o)$. When this condition is satisfied, the proposition yields a constant reward margin $m^{\mathrm{priv}}_{\mathrm{pick}}(t) = 0.01$. Otherwise, to provide dense spatial guidance before contact, it is formulated as $m^{\mathrm{priv}}_{\mathrm{pick}}(t) = \min\{r_g - \|p_e - p_o\|_2, -\epsilon\}$, where p_e and p_o represent the positions of the end-effector and the object, respectively, r_g is an approach threshold, and $\epsilon > 0$ is a sufficiently small constant.

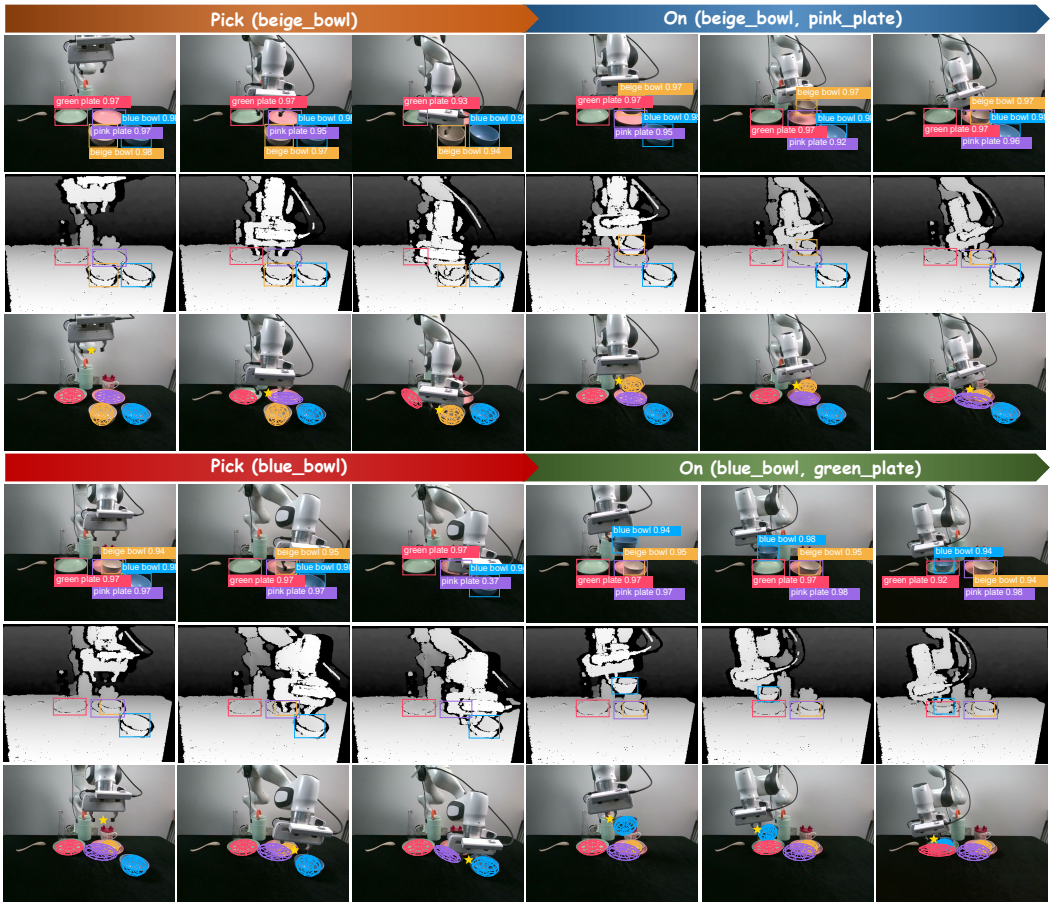


Figure 14: Visualization of the reward calculation process for Task 3 ("Pick up the beige bowl and place it on the pink plate, pick up the blue bowl and place it on the green plate") using SPECRL-V.

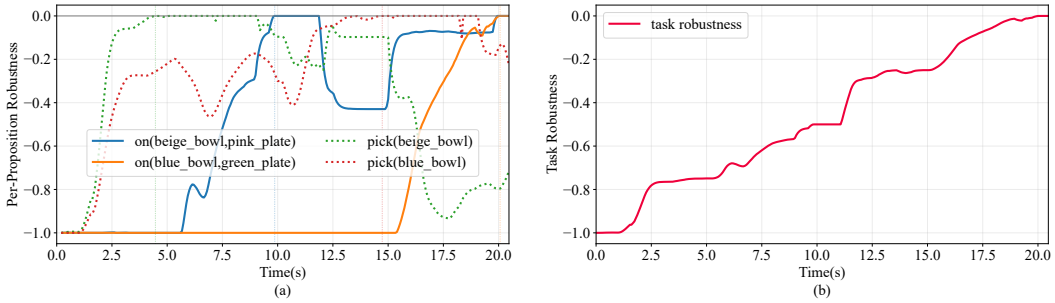


Figure 15: Robustness trajectories of the task-level atomic propositions and the aggregated task-level robustness for Task 3.

- $\mathrm{on}(o, x)$: evaluates whether object o has been placed on target x. The margin depends on whether x is a functional site or another object: (1) Functional site: The object center must lie within the site's horizontal footprint and a permitted vertical range. We define $m_{\mathrm{on}}^{\mathrm{priv}}(t) = \mathrm{AGM}\{s_x - |\delta_x|, s_y - |\delta_y|, \delta_z - (s_z - h_{\min}), (s_z + h_{\max}) - \delta_z\}$, where $\delta = (\delta_x, \delta_y, \delta_z)$ is the position of the object center and $s = (s_x, s_y, s_z)$ denotes the site's half-extents. (2) Object target: The object must be horizontally aligned with the target, positioned above it, and in valid surface contact. We define $m_{\mathrm{on}}^{\mathrm{priv}}(t) = \mathrm{AGM}\{d_{\mathrm{align}} - \|(p_o - p_x)_{xy}\|_2, (p_{o,z} - p_{x,z}) - \Delta z_{\mathrm{offset}}, m_c(o, x)\}$, where p_o

Table 4: Closed vocabulary of task-level atomic propositions.

Predicate	Satisfaction condition	Predicate	Satisfaction condition
$\mathrm{pick}(o_1)$	Object o_1 is grasped	$\mathrm{open}(o_1)$	Object o_1 is within its open range
$\mathrm{on}(o_1, o_2)$	Object o_1 is supported by o_2	$\mathrm{close}(o_1)$	Object o_1 is within its closed range
$\mathrm{in}(o_1, o_2)$	Object o_1 is inside o_2	$\mathrm{turn_on}(o_1)$	Switch o_1 is in its on state
$\mathrm{turn_off}(o_1)$	Switch o_1 is in its off state		

and p_x are the positions of the object and target, and the contact margin $m_c(o, x)$ is non-negative upon valid surface contact and negative otherwise.

- $\mathrm{in}(o, c)$: evaluates whether object o is placed inside container c. Containment is assessed by testing the object center against the container's bounding region $[L_c, U_c]$, where the margin is defined as the minimum signed distance to its six boundary planes: $m_{\mathrm{in}}^{\mathrm{priv}}(t) = \mathrm{AGM}_{j\in\{x,y,z\}}\{p_{o,j} - L_{c,j}, U_{c,j} - p_{o,j}\}$.
- $\mathrm{open}(a)$: evaluates whether articulated object a is opened. Opening is assessed using the joint position q_a relative to a target opening threshold $q^\star_{\mathrm{open}}$: $m_{\mathrm{open}}^{\mathrm{priv}}(t) = \sigma_{\mathrm{open}}(q_a - q^\star_{\mathrm{open}})$, where $\sigma_{\mathrm{open}} \in \{-1, 1\}$ is a sign factor ensuring the margin is non-negative within the open range.
- $\mathrm{close}(a)$: evaluates whether articulated object a is closed. Closing is assessed using the closed-range threshold $q^\star_{\mathrm{close}}$: $m_{\mathrm{close}}^{\mathrm{priv}}(t) = \sigma_{\mathrm{close}}(q_a - q^\star_{\mathrm{close}})$, where $\sigma_{\mathrm{close}} \in \{-1, 1\}$ orients the margin toward the closed range, configured opposite to σ_{open} for the same joint.
- $\mathrm{turn_on}(a)$: evaluates whether switch or knob a is turned on. It is assessed using its joint position q_a relative to the threshold boundary $q^\star_{\mathrm{on}}$: $m_{\mathrm{turn_on}}^{\mathrm{priv}}(t) = \sigma_{\mathrm{on}}(q_a - q^\star_{\mathrm{on}})$, where $\sigma_{\mathrm{on}} \in \{-1, 1\}$ is a sign factor ensuring a non-negative margin within the active on range.
- $\mathrm{turn_off}(a)$: evaluates whether switch or knob a is turned off. It is assessed analogously using the boundary $q^\star_{\mathrm{off}}$: $m_{\mathrm{turn_off}}^{\mathrm{priv}}(t) = \sigma_{\mathrm{off}}(q_a - q^\star_{\mathrm{off}})$, where $\sigma_{\mathrm{off}} \in \{-1, 1\}$ orients the margin toward the off range, configured opposite to σ_{on} for the same device.

C.2 VISUAL GROUNDING

SPECRL also includes a visual grounding module to support reward computation when privileged simulator states are unavailable. In a real-world setting, object poses, contact information, and articulated-object states cannot be read directly; instead, the visual module uses robot proprioception and observations from a global RGB-D camera to estimate whether task-level atomic propositions are satisfied. Specifically, YOLOE produces object segmentation masks, which are combined with aligned depth observations to reconstruct object point clouds. These reconstructions provide estimates of geometric relationships and articulated-component motion. Together with proprioceptive measurements, including the gripper state, they are used to compute signed margins for the corresponding satisfaction conditions.

The following definitions instantiate this design for task-level atomic propositions under visual perception. Similarly, **while these specific instances provide a viable vision-based scheme, they represent an operational rather than uniquely optimal design, and users can adapt or extend these formulations to new domain requirements.** Their geometric thresholds and hyperparameters can also be tuned through lightweight trials that track task completion in real-world deployments.

Although this analytical setup entails initial engineering effort, the payoff is well worth it. By providing a deterministic and rigorous formulation for reward computation and task completion, SPECRL establishes an effective paradigm for vision-based reward shaping that avoids the reliance on heuristic-heavy, black-box VLM queries. In addition to enabling stable online VLA RLFT without the computational overhead of repeated VLM inference, this analytical grounding approach can be extended to broader applications. For instance, it can provide reliable, deterministic scoring for real-world demonstration data to help bootstrap and train general-purpose VLM reward models.

- $\mathrm{pick}(o)$: evaluates whether the robot successfully grasps object o under visual perception. Grasping is assessed through proximity, closure, and motion consistency, mainly combined via

$m^{\text{vis}}_{\text{pick}} = \text{AGM}\{m_{\text{near}}, m_{\text{close}}, m_{\text{comove}}\}$. The proximity margin is $m_{\text{near}} = r_{\text{center}} - \|\hat{p}_o - p_e\|_2$, the closure margin is $m_{\text{close}} = w_{\text{close}} - w_t$, and $m_{\text{comove}} = d_{\text{comove}} - \|(\hat{p}_o - p_e)_t - (\hat{p}_o - p_e)_{t-\tau}\|_2$ evaluates relative motion consistency between the current time step t and a historical step $t - \tau$. Here, $\hat{p}_o$, p_e, and w_t denote the estimated object center, end-effector position, and current gripper opening, respectively.

- $\text{on}(o, x)$: evaluates whether object o has been placed on target x using visual estimates. Placement requires horizontal alignment, appropriate relative height, support, and gripper release: $m^{\text{vis}}_{\text{on}} = \text{AGM}\{m_{xy}, m_z, m_{\text{support}}, m_{\text{release}}\}$. Horizontal alignment is measured by $m_{xy} = \epsilon_{xy} - \|\hat{p}_{o,xy} - \hat{p}_{x,xy}\|_2$. For functional sites, horizontal placement is evaluated against the site's bounding footprint rather than its center. Here, m_z, m_{support}, and m_{release} denote the margins relative to the vertical height, surface support, and gripper release state, respectively. $\hat{p}_{o,xy}$ and $\hat{p}_{x,xy}$ represent the estimated horizontal positions of object o and target x.
- $\text{in}(o, c)$: evaluates whether object o is visually perceived inside container c. Containment requires the estimated object center to lie within the container's horizontal bounds, below its rim, and above its floor, while the gripper is released: $m^{\text{vis}}_{\text{in}} = \text{AGM}\{m_{xy}, m_{\text{below_rim}}, m_{\text{above_floor}}, m_{\text{release}}\}$. Horizontal inclusion is evaluated as $m_{xy} = \text{AGM}_{j\in\{x,y\}}\{\hat{p}_{o,j} - L^v_{c,j}, U^v_{c,j} - \hat{p}_{o,j}\}$, where $m_{\text{below_rim}}$, $m_{\text{above_floor}}$, and m_{release} denote the margins relative to the container rim, floor, and gripper release state, respectively. Here, $\hat{p}_o$ represents the estimated object center, and $[L^v_c, U^v_c]$ denotes the estimated container bounds.
- $\text{open}(a)$: evaluates whether articulated object a is opened by measuring the travel distance of the movable panel along its opening axis. The displacement Δy_{disp} is estimated from changes in the panel's point-cloud coordinates, yielding $m^{\text{vis}}_{\text{open}} = \Delta y_{\text{disp}} - d_{\text{open}}$.
- $\text{close}(a)$: evaluates whether articulated object a is closed based on its geometric retraction or structural closing progress along the articulation axis. It is formulated via either a retraction distance $m^{\text{vis}}_{\text{close}} = e_{\text{retract}} - d_{\text{close}}$, or a closing progress $m^{\text{vis}}_{\text{close}} = \rho_{\text{progress}} - \rho_{\text{thresh}}$. Here, e_{retract} and ρ_{progress} denote the estimated retraction distance and closing progress.
- $\text{turn_on}(a)$: evaluates whether stove a is turned on by detecting flame presence within its active region mask Ω. The margin $m^{\text{vis}}_{\text{turn_on}} = \frac{1}{|\Omega|}\sum_{u\in\Omega}\mathbb{I}[I^{\text{on}}_u] - \tau_{\text{flame}}$ measures the fraction of pixels satisfying the flame color condition I^{on}_u.
- $\text{turn_off}(a)$: evaluates whether stove a is turned off, which can be defined symmetrically to $\text{turn_on}(a)$ by checking the absence of flame within its active region mask.

3D Geometry Estimation from RGB-D Observations.

To evaluate spatial relations, SPECRL requires representing detected objects in the robot's 3D coordinate frame. We achieve this by combining YOLOE instance masks with aligned RGB-D observations through a two-step pipeline: metric depth conversion, and coordinate transformation.

(1) Metric Depth Conversion. For an object mask $M(u, v)$, we retain only the pixels with valid depth along the camera's optical axis, defining the valid set as $\Omega = \{(u, v) \mid M(u, v) = 1 \text{ and } Z(u, v) \text{ is valid}\}$. Using instance masks rather than bounding boxes effectively excludes irrelevant background pixels. To convert these depth values into metric units, we handle simulation and real-world sources respectively: in simulation, normalized depth values $d \in [0, 1]$ are converted using the near and far clipping planes n and f via $Z = n/[1 - d(1 - n/f)]$, whereas real-world 16-bit depth values are scaled directly and aligned using sensor extrinsics.

(2) Back-projection and Coordinate Transformation. With the metric depth Z established for each pixel in Ω, we back-project them into the camera frame. Given camera intrinsics with focal lengths (f_x, f_y) and principal point (c_x, c_y), the 3D coordinates are obtained as:

$$X = \frac{(u - c_x)Z}{f_x}, \qquad Y = \frac{(v - c_y)Z}{f_y}, \tag{5}$$

or equivalently $p_{\text{camera}} = ZK^{-1}[u, v, 1]^T$ using the camera intrinsic matrix K. Finally, to align with the robot and task spaces, we transform these points from the camera frame to the target geometric grounding frame (i.e., the world frame in simulation or the robot base frame in the real world). Using the rotation $R_{\text{ref_cam}}$ and translation $t_{\text{ref_cam}}$ from the camera to the reference frame, we compute:

$$p_{\text{ref}} = R_{\text{ref_cam}} p_{\text{camera}} + t_{\text{ref_cam}}. \tag{6}$$

The resulting object point clouds are then directly compared with robot states and other geometries to compute proposition margins.

C.3 PPO Optimization for the Flow-Noise Policy

As defined in Section 2, the policy generates an H-step action chunk conditioned on the observation history and instruction. To provide a self-contained overview, we briefly review the core idea below; for further technical details, we refer readers to Chen et al. (2025a). Starting from Gaussian noise, the Flow-Noise policy first performs stochastic denoising steps: at each step, it samples the next state from a Gaussian distribution, where the mean is determined by the predicted flow update and the variance is learned. The final step yields the desired action chunk $a_t^{(H)}$.

Following standard practices in policy optimization, the complete denoising sequence is treated as a trajectory ξ_t for PPO. Its log-likelihood is given by the sum of transition log-probabilities across all denoising steps, omitting the parameter-independent initial Gaussian term.

To compute the advantage $\widehat{\mathcal{A}}_t$ for each trajectory, a standard reward-aggregation and advantage-estimation scheme suited for action chunking is adopted: the low-level rewards defined in Equation (4) are accumulated within the H-step chunk, the chunk-level discount factor is set to γ^H, and generalized advantage estimation is applied using a value critic evaluated at chunk boundaries. The policy parameters are then optimized by maximizing the standard clipped PPO objective:

$$\mathcal{J}_{\mathrm{PPO}}(\theta,\phi) = \mathbb{E}_t\Big[\min\Big(\omega_t(\theta,\phi)\widehat{\mathcal{A}}_t,\ \mathrm{clip}(\omega_t(\theta,\phi),1-\epsilon,1+\epsilon)\,\widehat{\mathcal{A}}_t\Big)\Big], \tag{7}$$

where $\omega_t(\theta,\phi)$ represents the likelihood ratio of the sampled denoising trajectory ξ_t under the policy parameters (θ,ϕ). Both the VLA policy parameters θ and the transition-noise network parameters ϕ are jointly optimized during RLFT.

D LLM Parsing Examples and Prompt Template

SpecRL leverages an LLM-aided process to convert language instructions into STL specifications constituted by task-level atomic propositions. Although the LLM succeeds in most cases, occasional inconsistencies can occur due to limitations in the parsing process or discrepancies between instructions and the environment. Below, we present several successful parsing examples, alongside examples of such inconsistencies.

D.1 Successful Parsing Examples

Example 1: "put the black bowl in the bottom drawer of the cabinet and close it"

Analysis: The model correctly resolves the pronoun "it" to `bottom_drawer` and synthesizes a complete temporal action chain: `pick` → `in` → `close`.

```
{"paths": [{"name": "only", "stages": [
  {"atoms": [{"pred": "pick",  "obj": "black_bowl"}]},
  {"atoms": [{"pred": "in",    "obj": "black_bowl", "target": "
      bottom_drawer"}]},
  {"atoms": [{"pred": "close", "target": "bottom_drawer"}]}
]}]}
```

Example 2: "pick up the black bowl next to the ramekin and place it on the plate"

Analysis: The model correctly parses the spatial relational description (`next_to` the `ramekin`) to ground the target object and synthesizes a valid execution sequence: `pick` → `on`.

```
{"paths": [{"name": "only", "stages": [
  {"atoms": [{
    "pred": "pick",
    "obj": "black_bowl",
    "spatial": {"relation": "next_to", "refs": ["ramekin"]}
  }]},
```

```
  {"atoms": [{"pred": "on", "obj": "black_bowl", "target": "plate"}]}
]}]}
```

Example 3: "put both the cream cheese box and the butter in the basket"

Analysis: For two independent sub-goals, the LLM successfully constructs two alternative execution-order paths (`object_1_then_object_2` and `object_2_then_object_1`).

```
{"paths": [
  {"name": "object_1_then_object_2", "stages": [
    {"atoms": [{"pred": "pick", "obj": "cream_cheese_box"}]},
    {"atoms": [{"pred": "in",  "obj": "cream_cheese_box", "target": "
        basket"}]},
    {"atoms": [{"pred": "pick", "obj": "butter"}]},
    {"atoms": [{"pred": "in",  "obj": "butter", "target": "basket"}]}
  ]},
  {"name": "object_2_then_object_1", "stages": [
    {"atoms": [{"pred": "pick", "obj": "butter"}]},
    {"atoms": [{"pred": "in",  "obj": "butter", "target": "basket"}]},
    {"atoms": [{"pred": "pick", "obj": "cream_cheese_box"}]},
    {"atoms": [{"pred": "in",  "obj": "cream_cheese_box", "target": "
        basket"}]}
  ]}
]}
```

D.2 Parsing Errors and Human Refinements

Example 1: "put the yellow and white mug in the microwave and close it"

Error Analysis: There is only one mug in the scene (`yellow_and_white_mug`). The LLM incorrectly splits a single physical entity into two independent objects (`yellow_mug` and `white_mug`). Consequently, it generates an erroneous dual-path plan and fabricates an unprompted `turn_on(microwave)` precondition predicate.

Human Revision: The `turn_on` predicate is deleted, the separated entities are merged back into a single `yellow_and_white_mug` entity, and the topology is reverted to a single-path chain.

```
// Original LLM Output (Error)
{"paths": [
  {"name": "yellow_mug_then_white_mug", "stages": [
    {"atoms": [{"pred": "turn\_on", "target": "microwave"}]}, // [
        Hallucination]
    {"atoms": [{"pred": "pick", "obj": "yellow_mug"}]},     // [
        Hallucination]
    {"atoms": [{"pred": "in",   "obj": "yellow_mug", "target": "microwave
        "}]},
    {"atoms": [{"pred": "pick", "obj": "white_mug"}]},      // [
        Hallucination]
    {"atoms": [{"pred": "in",   "obj": "white_mug", "target": "microwave
        "}]},
    {"atoms": [{"pred": "close", "target": "microwave"}]}
  ]},
  // ... path 2 omitted ...
]}

// Revised Output
{"paths": [{"name": "only", "stages": [
  {"atoms": [{"pred": "pick",  "obj": "yellow_and_white_mug"}]},
  {"atoms": [{"pred": "in",    "obj": "yellow_and_white_mug", "target": "
      microwave"}]},
  {"atoms": [{"pred": "close", "target": "microwave"}]}
]}]}
```

Example 2: "open the top drawer and put the bowl inside"

Error Analysis: The LLM generates an `open(top_drawer)` predicate at the beginning of the sequence based on the literal interpretation of the instruction. However, in the physical simulation ground truth for this task, the top drawer is initially already open.

Human Revision: The `open(top_drawer)` node is removed to ensure the symbolic sequence strictly aligns with the underlying physical state of the simulation environment.

```
// Original LLM Output (Error)
{"paths": [{"name": "only", "stages": [
  {"atoms": [{"pred": "open", "target": "top_drawer"}]},    // [
      Hallucination]
  {"atoms": [{"pred": "pick", "obj": "bowl"}]},
  {"atoms": [{"pred": "in",   "obj": "bowl", "target": "top_drawer"}]}
]}]}

// Revised Output
{"paths": [{"name": "only", "stages": [
  {"atoms": [{"pred": "pick", "obj": "bowl"}]},
  {"atoms": [{"pred": "in",   "obj": "bowl", "target": "top_drawer"}]}
]}]}
```

Prompt

You convert a natural-language robot-manipulation instruction into an observation-agnostic symbolic Signal Temporal Logic (STL) specification.

OUTPUT FORMAT

Return STRICT JSON only. Do not include explanations, comments, or Markdown code fences. The output must follow this structure:

```
{
  "paths": [
    {
      "name": "<short_name>",
      "stages": [
        {
          "atoms": [
            {
              "pred": "...",
              "obj": "...",
              "target": "..."
            }
          ]
        }
      ]
    }
  ]
}
```

ATOMIC PREDICATES

The predicate vocabulary is CLOSED. Use only the following predicates:

1. `pick(obj)`
   Meaning: Grasp object `obj`.
   Arguments: `obj` is required; `target` must be omitted.
2. `on(obj, target)`
   Meaning: Place `obj` on the surface of `target`.
   Arguments: Both `obj` and `target` are required.
3. `in(obj, target)`
   Meaning: Place `obj` inside the container `target`.
   Arguments: Both `obj` and `target` are required.
4. `open(target)`

Meaning: Open an articulated fixture, such as a drawer or door.
Arguments: `target` is required; `obj` must be omitted.

5. `close(target)`
   Meaning: Close an articulated fixture.
   Arguments: `target` is required; `obj` must be omitted.
6. `turn_on(target)`
   Meaning: Turn on a fixture, such as a stove.
   Arguments: `target` is required; `obj` must be omitted.
7. `turn_off(target)`
   Meaning: Turn off a fixture.
   Arguments: `target` is required; `obj` must be omitted.

GENERATION RULES

1. **Semantic role names**
   The values of `obj` and `target` must be semantic role names in singular `snake_case` form.
   Preserve every qualifying adjective in the instruction, especially color and type descriptors, so that each name faithfully identifies the referenced entity.
   Examples:
   - "the black bowl" → `black_bowl`
   - "the yellow mug" → `yellow_mug`
   - "the white mug" → `white_mug`
   - "alphabet soup" → `alphabet_soup`

   Use the bare category only when the instruction provides no qualifier:
   - "the bowl" → `bowl`
   - "the stove" → `stove`

   Never use simulator instance names, object IDs, coordinates, or observation-dependent identifiers. The same entity must retain the same semantic role name everywhere in the output.
2. **Sequential stages**
   Each atomic predicate must form its own sequential stage. Therefore, every `stages[i].atoms` array contains exactly one atom.
   List all stages in their required execution order. An object must always be picked before applying `on` and `in`.
3. **Preconditions**
   Place required preconditions before the operations that depend on them.
   Examples:
   - Turn on the stove before placing a pot on it.
   - Pick and place an object before closing the fixture containing it.
   - Open a closed container before placing an object inside it.
4. **Independent subgoals**
   If the instruction contains two independent object-manipulation subgoals, emit two valid paths representing both execution orders.
   For example, for independent subgoals involving X and Y, emit:
   - one path executing X before Y;
   - one path executing Y before X.

   Otherwise, emit exactly one path with the name `"only"`.
5. **Inapplicable fields**
   Omit fields that do not apply to a predicate:
   - `pick` has no `target`;
   - `open`, `close`, `turn_on`, and `turn_off` have no `obj`.

   Do not include inapplicable fields with empty strings.

6. **Spatial disambiguation**
   An atom may contain an optional `spatial` object when the instruction uses a positional phrase to disambiguate a manipulated object.
   Examples of such phrases include:
   - "the black bowl between the plate and the ramekin";
   - "the black bowl next to the cookie box";
   - "the black bowl on the stove";
   - "the black bowl in the top drawer".

   Attach `spatial` only to the atom whose entity it disambiguates, usually the corresponding `pick` atom:

```
{
  "pred": "pick",
  "obj": "black_bowl",
  "spatial": {
    "relation": "between",
    "refs": ["plate", "ramekin"]
  }
}
```

   The value of `relation` must be a `snake_case` spatial relation, such as:
   - `"between"`
   - `"next_to"`
   - `"on_the"`
   - `"in_the"`
   - `"left_of"`
   - `"right_of"`
   - `"middle"`

   The entries in `refs` must be semantic role names for the reference landmarks. Do not treat the reference landmarks as manipulation targets unless the instruction explicitly requires manipulating them.
   Include `spatial` only when an explicit positional phrase is needed for entity disambiguation. Otherwise, omit it entirely.

FINAL VALIDATION
Before returning the JSON, verify that:

1. Every predicate belongs to the closed vocabulary.
2. Every stage contains exactly one atom.
3. All required arguments are present.
4. All inapplicable arguments are omitted.
5. Semantic role names are singular and use `snake_case`.
6. Entity names preserve all qualifiers from the instruction.
7. The stages follow a valid execution order.
8. The output contains no text outside the JSON object.