# STRUCTRL: ONLINE STRUCTURED REINFORCEMENT LEARNING FOR LONG-HORIZON VISION-LANGUAGE-ACTION TASKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Vision-language-action (VLA) models perform well on shorter-horizon manipulation tasks but still struggle with long-horizon tasks that require multiple dependent manipulations from a single command. Online reinforcement learning (RL) can improve these policies through environment interaction, yet many existing methods provide reward only after the complete task succeeds. However, such terminal supervision is sparse and does not distinguish early failures from rollouts that make substantial partial progress. We propose STRUCTRL, an online RL framework that constructs structured intermediate supervision from verifiable subtask completions. STRUCTRL decomposes each task into verifiable subtasks, grants intermediate rewards only after the prerequisite subtasks have been completed, and scales each reward according to completion pace. Across RoboCasa365 and LIBERO-Long with GR00T-N1.5 and $\pi_{0.5}$, STRUCTRL consistently outperforms evaluated online RL baselines. These results show that verifiable, structured intermediate rewards improve long-horizon VLA post-training. Code is included in the supplementary material.

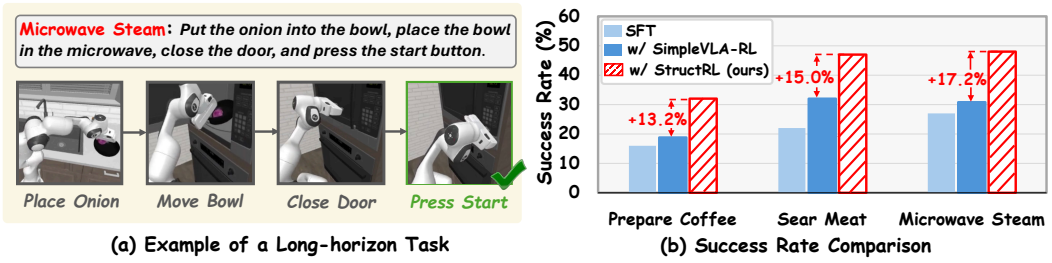



Figure 1: (a) An example long-horizon VLA task from RoboCasa365 (Nasiriany et al., 2026). (b) Success rates (%) of GR00T-N1.5 on three representative composite tasks. STRUCTRL consistently outperforms both the SFT policy and the RL baseline SimpleVLA-RL (Li et al., 2026).

## 1 INTRODUCTION

Vision-language-action (VLA) models map visual observations and language instructions to low-level robot actions (Black et al., 2025b;a; Bjorck et al., 2025; Kim et al., 2026). Recent models have shown strong capabilities in household manipulation (Black et al., 2025a), humanoid control (Bjorck et al., 2025), and dexterous manipulation (Gemini Robotics Team, 2025). Despite this progress, VLA evaluation still focuses primarily on tasks such as picking up an object and placing it at a target location (Liu et al., 2023; Li et al., 2024). These instructions typically specify a single self-contained skill rather than an extended sequence of dependent actions (Han et al., 2025; Mees et al., 2022).

A more capable VLA should execute an extended task from a single instruction, without requiring a new command after each manipulation. We refer to these problems as *long-horizon tasks*. As illustrated in Figure 1, they require the policy to compose multiple dependent skills, such as grasping several objects, navigating to a target location, and operating articulated fixtures in the correct order. Long-horizon VLA policies are commonly trained through supervised fine-tuning (SFT) on human



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.

demonstrations (Nasiriany et al., 2026), yet their task success rates remain limited. Because SFT exposes the policy primarily to demonstrated states, small errors during a long rollout can move it into poorly covered states from which recovery is difficult.

Online reinforcement learning (RL) offers a natural way to address this distribution shift by allowing the policy to train on states encountered during its own interactions (Zang et al., 2026; Nasiriany et al., 2026). Most online VLA RL methods, however, rely on a terminal reward that is issued only when the entire task succeeds (Nasiriany et al., 2026; Wang et al., 2026; Li et al., 2026). This signal becomes increasingly sparse as the task horizon grows and successful rollouts become rare. More importantly, it does not represent partial progress: a rollout that completes every step except the last receives the same return as one that fails near the beginning. Long-horizon VLA training therefore requires a denser signal that can identify meaningful progress before final success.

Learned reward models provide denser supervision by estimating intermediate task progress (Shu et al., 2025; Zhang et al., 2026; Tan et al., 2025a; Liang et al., 2026). A scalar progress estimate, however, does not by itself specify which prerequisite events make a detected completion valid. In long-horizon manipulation, an event may appear locally useful but fail to advance the task because a required prerequisite has not yet been completed. Therefore, dense supervision should encode both whether an event occurred and whether it constitutes valid progress toward the final goal given the events completed so far.

To provide dense supervision while respecting the dependency structure of long-horizon tasks, we propose StructRL, an online RL framework that constructs intermediate rewards. StructRL applies an automatic pipeline to decompose each instruction into a set of subtasks whose completion can be directly verified from the environment state, then organizes them into ordered dependency groups. This structure matters because satisfying a local completion does not always constitute valid task progress. For example, closing a box before the required objects have been placed inside should not receive intermediate credit.

StructRL rewards each verified subtask completion and shapes that reward with two mechanisms. *Structure-aware reward gating* determines whether a detected completion is eligible for intermediate credit: a subtask is rewarded only after all of its prerequisites have been satisfied. *Dynamic reward pacing* determines the magnitude of that credit, assigning larger rewards to subtasks completed more quickly relative to the demonstrations. The resulting chunk-level rewards are used to optimize the VLA with Proximal Policy Optimization (PPO) (Schulman et al., 2017). Together, these mechanisms provide dense supervision while keeping reward assignment verifiable and consistent with task dependencies.

We evaluate StructRL on RoboCasa365 and LIBERO-Long using both GR00T-N1.5 and $\pi_{0.5}$. Across both benchmarks and backbones, StructRL consistently outperforms the evaluated online RL baselines. On RoboCasa365 with GR00T-N1.5, for example, StructRL reaches 49.1% success rate (SR), compared with 38.6% for SFT and 41.5% for the strongest evaluated online RL baseline. The component ablation shows that verified subtask rewards provide most of this gain, with gating and pacing adding further improvements. Additional results show that the structured reward is compatible with GRPO and remains effective on shorter-horizon LIBERO suites. Collectively, these results support structured intermediate supervision as an effective approach to long-horizon VLA post-training.

## 2 Preliminaries

In this section, we formalize action-chunk VLA interaction and PPO training, establishing the notation used by the structured reward formulation in Section 3.

**VLA Rollouts.** We model VLA interaction with the environment at the action-chunk level. At chunk index $k$, the policy $\pi_\theta$ receives an observation $o_k$ containing multi-view images, the language instruction, and proprioceptive state, and uses its action expert to generate an action chunk $a_k \in \mathbb{R}^{L\times d}$. Here, $L$ is the number of consecutive low-level actions in the chunk, and $d$ is the number of controllable robot degrees of freedom.[1] The environment executes the complete chunk before

[1]We use flow-matching VLAs (*e.g.*, GR00T-N1.5 and $\pi_{0.5}$) as the running examples; the same action-chunk formulation applies to autoregressive VLAs, whose token-level likelihoods are directly available.

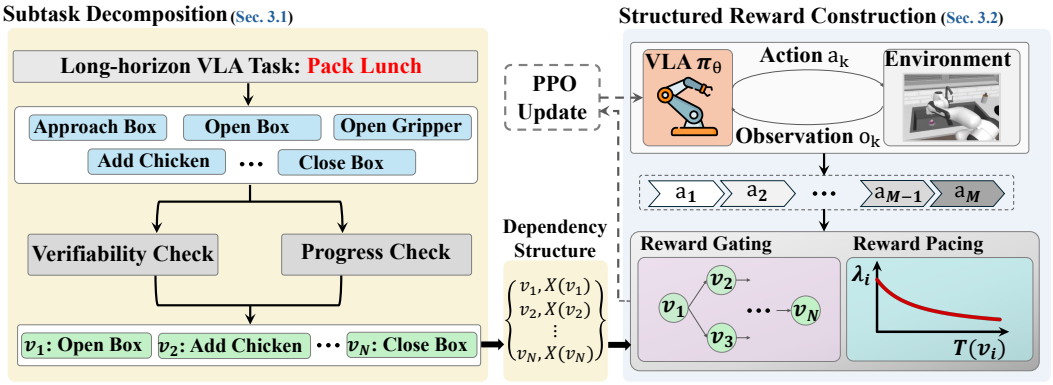



Figure 2: Overview of StructRL. **Left:** An LLM decomposes the task command into candidate subtasks and assigns a dependency structure to the retained subtasks, represented by prerequisite sets $X(v_i)$. The verifiability check removes candidates without a reliable binary completion criterion (*e.g.*, *Approach Box*), while the progress check removes candidates that do not by themselves indicate progress toward task completion (*e.g.*, *Open Gripper*). **Right:** Structure-aware reward gating uses this dependency structure to determine whether a detected completion is eligible for reward, while dynamic reward pacing determines the magnitude of that reward. The resulting chunk-level rewards are used to optimize the VLA with PPO. The training pseudocode is shown in Appendix A.

returning $o_{k+1}$. The resulting rollout is $\tau = (o_1, a_1, \ldots, o_M, a_M)$, containing $M$ action chunks, which is used for RL training.

**PPO Training.** We next explain how the collected rollouts $\tau$ are used to optimize $\pi_\theta$, taking PPO (Schulman et al., 2017) as an example. For VLA training, PPO operates at the chunk level: for each chunk $a_k$ in $\tau$, it maximizes the clipped surrogate objective

$$\mathcal{L}(\theta) = \mathbb{E}_k\Big[\min\left(\rho_k A_k,\ \mathrm{clip}(\rho_k, 1-\epsilon, 1+\epsilon)\, A_k\right)\Big], \tag{1}$$

where $\epsilon$ is the standard PPO clipping threshold, $\rho_k = \pi_\theta(a_k \mid o_k)/\pi_{\theta_{\text{old}}}(a_k \mid o_k)$ is the importance sampling ratio, and $A_k$ is the advantage estimate for each chunk $a_k$, computed from the chunk-level reward sequence using a learned critic. Under the standard terminal-only reward setting, intermediate chunks receive zero reward, while the final chunk receives a positive reward only upon successful task completion.

For flow-matching VLAs, computing $\rho_k$ is nontrivial because deterministic ODE sampling does not directly provide the chunk likelihood $\pi_\theta(a_k \mid o_k)$. Following prior work (Zang et al., 2026), we inject Gaussian noise into each denoising step to convert the ODE sampling process into an SDE, making the chunk likelihood tractable. This enables the likelihood ratio required by PPO to be evaluated. We next introduce StructRL, which replaces the terminal-only reward with structured intermediate rewards while retaining the PPO objective in Eq. (1).

## 3 Methodology

Let $c$ denote a long-horizon task command and $\pi_\theta$ the VLA policy to be fine-tuned through online RL. As illustrated in Figure 2, StructRL operates in two stages. First, an LLM decomposes $c$ into a set of verifiable, goal-aligned subtasks and organizes them into ordered dependency groups. Second, StructRL converts this structure into chunk-level rewards through structure-aware reward gating and dynamic reward pacing, then optimizes $\pi_\theta$ with PPO (Section 3.2).

### 3.1 Subtask Decomposition

**Semantic decomposition**. A long-horizon command may combine heterogeneous skills, including grasping, base navigation, and articulated-object manipulation. We decompose the command into intermediate subtasks that satisfy two requirements: completion can be detected using a binary environment-state criterion, and the completed subtask represents progress toward the final task goal.

For each command $c$, we prompt an LLM to propose natural-language candidate subtasks, such as *open the box*, and to return only candidates that pass two checks: (i) *Verifiability Check*: completion can be detected from the environment state using a binary criterion. (ii) *Progress Check*: satisfying that criterion represents progress toward the final task goal rather than an incidental or non-progressive behavior.

Figure 2 illustrates both checks for the *Pack Lunch* task. *Approach Box* fails the Verifiability Check because its completion is difficult to define using a reliable binary environment-state criterion, while *Open Gripper* fails the Progress Check because it does not by itself indicate a successful task progress and may also occur during a failed attempt. The retained subtasks form $\mathcal{V} = \{v_1, \ldots, v_N\}$, where $N$ is the number of subtasks.

**Dependency structure assignment**. A flat set of subtasks is insufficient to fully characterize progress in a long-horizon task, because the subtasks in $\mathcal{V}$ are not mutually independent: some must be completed before others, while others may be executed in arbitrary order. Ignoring these relations can treat an out-of-order completion as valid task progress. For example, the *Pack Lunch* decomposition follows the dependency pattern *Open Box* $\rightarrow$ {*Add Chicken*, *Add Apple*} $\rightarrow$ *Close Box*, where the two additions may occur in either order, but *Close Box* should only be considered valid progress after both additions have been completed.

Accordingly, the final LLM output organizes the retained subtasks into ordered dependency groups. Subtasks within the same group may be completed in any order, whereas all subtasks in an earlier group must be completed before those in a later group. We convert this stage-wise ordering into prerequisite sets for reward construction. For each subtask $v_i \in \mathcal{V}$, we define $X(v_i) \subseteq \mathcal{V}$ as the set of subtasks that must be completed before $v_i$, with $X(v_i) = \varnothing$ for subtasks without prerequisites:

$$\mathcal{V} \longrightarrow \{(v_1, X(v_1)), \ldots, (v_N, X(v_N))\}. \tag{2}$$

This representation determines when each detected subtask completion becomes eligible for intermediate reward. Appendix B provides the complete prompt and generated decompositions, and Section 4.5 analyzes decomposition granularity. We next describe how this structured decomposition is converted into rewards.

### 3.2 Structured Reward Construction

Based on the decomposed subtasks and the dependency structure among them, StructRL constructs the reward for each action chunk through two decisions. Structure-aware reward gating determines whether a detected subtask completion is eligible for reward, based on the task dependencies. Dynamic reward pacing determines the magnitude of each eligible reward, based on completion pace. We describe these eligibility and magnitude components in turn.

#### 3.2.1 Structure-Aware Reward Gating

Consider a rollout $\tau$ with action chunks $\{a_1, \ldots, a_M\}$. A detected completion of $v_i$ is eligible for intermediate reward only if every prerequisite in $X(v_i)$ has already been completed and $v_i$ has not previously been rewarded. Thus, each subtask can contribute intermediate reward at most once. The reward assigned to each chunk $a_k$ is therefore defined by

$$r_k = \underbrace{\sum_{i=1}^{N} \mathbb{1}_k(v_i, X(v_i)) \cdot \lambda_i}_{\text{subtask rewards}} + \underbrace{\mathbb{1}_k(c) \cdot \lambda_c}_{\text{terminal reward}}. \tag{3}$$

Here, $\mathbb{1}_k(v_i, X(v_i)) = 1$ when completion of $v_i$ is detected at chunk $a_k$, every prerequisite in $X(v_i)$ has been completed, and $v_i$ has not previously been rewarded. The terminal indicator $\mathbb{1}_k(c) = 1$ when the complete task first succeeds at chunk $a_k$. We use a fixed terminal reward $\lambda_c$ and determine each intermediate reward $\lambda_i$ dynamically from its completion pace.

#### 3.2.2 Dynamic Reward Pacing

A straightforward choice of $\lambda_i$ is to use a fixed reward for all subtasks. However, such a design does not explicitly distinguish fast, direct completions from delayed ones involving unnecessary wandering, which can blur credit assignment for task progress in long-horizon rollouts. Therefore,

StructRL instead scales the reward according to completion pace:

$$\lambda_i = \beta \cdot \frac{1}{1 + T(v_i)/T_d(v_i)}. \tag{4}$$

Here, $T(v_i)$ is the number of action chunks elapsed since the previous rewarded subtask completion; for the first rewarded subtask, it is measured from the beginning of the rollout. For each subtask $v_i$, we compute $T_d(v_i)$ from the demonstration interval that begins when all prerequisites of $v_i$ have first become complete and ends when $v_i$ is completed. We average this interval across demonstrations in which both events are observed. The scale parameter $\beta$ upper-bounds the reward for one subtask.

Finally, we train the VLA with the PPO objective introduced in Eq. (1), resulting in the optimized policy $\pi_\theta$.

#### 3.2.3 Extension to GRPO

PPO is our default optimizer, but the same structured rewards can be used with Group Relative Policy Optimization (GRPO) (Shao et al., 2024). For each task instruction paired with one fixed initial state, we collect a group of $G = 8$ rollouts. We run $256$ environments in parallel for two rollout epochs, so each training iteration collects $64$ groups, or $512$ rollouts, the same number as PPO. Within each rollout $\tau$, we sum the chunk-level rewards $R(\tau) = \sum_{k=1}^{M} r_k$, and GRPO normalizes these returns within each rollout group to compute group-relative advantages. Section 4.6 reports the corresponding results.

## 4 Experiments

### 4.1 Experimental Setup

**Benchmarks.** We evaluate StructRL on two manipulation benchmarks: RoboCasa365 (Nasiriany et al., 2026) and LIBERO-Long (Liu et al., 2023). RoboCasa365 Composite-Seen contains 16 tasks in kitchen environments. A 7-DoF Franka Panda arm mounted on a 4-DoF Omron mobile base must coordinate arm manipulation and base navigation. The tasks combine skills such as pick-and-place, articulated-fixture operation, and navigation into extended execution sequences. We evaluate 100 episodes per task across the 10 held-out scenarios. LIBERO-Long contains 10 tabletop manipulation tasks performed by a fixed-base Franka Panda arm. We evaluate each task on its 50 official initial scenarios. On both benchmarks, we report mean task success rate (SR).

**Baselines.** We report three types of quantitative comparisons. First, we evaluate VLA models after SFT: $\pi_0$ (Black et al., 2025b), $\pi_{0.5}$ (Black et al., 2025a), RLDX-1 (Kim et al., 2026), and GR00T-N1.5 (Bjorck et al., 2025). We use a benchmark-specific released checkpoint when available; otherwise, we perform SFT following the official recipe. Second, we compare online VLA RL methods that can be evaluated under a common protocol: Sparse-RL (Zang et al., 2026), SimpleVLA-RL (Li et al., 2026), and PolicyTrim (Wang et al., 2026). These methods and StructRL start from the same SFT checkpoints of GR00T-N1.5 and $\pi_{0.5}$. Each method retains its original optimization design and receives the same number of environment interactions. For each benchmark, all online RL methods train one policy jointly across all tasks. Third, online methods construct dense intermediate rewards using learned progress or value estimators (Shu et al., 2025; Zhang et al., 2026; Tan et al., 2025a; Liang et al., 2026). We instantiate the Robometer (Liang et al., 2026) baseline, a vision-language reward model that predicts per-frame task progress, by integrating its released checkpoint into the RLinf-VLA (Zang et al., 2026) training stack and converting its frame-level progress predictions into chunk-level rewards. Appendix F details the reward conversion and implementation.

**Implementation.** We implement StructRL in the RLinf-VLA (Zang et al., 2026) framework. Before RL training, Claude Opus 4.8 (Anthropic, 2026) generates the subtask decomposition and dependency structure for each task. These decompositions are fixed and reused throughout training, so no LLM is required during rollout collection. Each retained subtask is then grounded before training to a binary predicate over the simulator state using a fixed rule-based procedure. The verb phrase determines the predicate type, while its arguments identify the task objects or fixtures. We implement these predicates using the benchmark's native state representations and success-check utilities. For example, *place $o$ in $r$* is mapped to a containment predicate that checks whether object $o$

Table 1: Success rate (%) by task-horizon bucket. From shortest to longest, the RoboCasa365 buckets contain 3, 5, and 8 tasks, while the LIBERO-Long buckets contain 5, 3, and 2 tasks. *Overall* is the mean SR across all tasks in the benchmark. Within each backbone block, blue cells mark the strongest online RL baseline in each column. $\Delta$ is the difference between StructRL and that baseline, in percentage points.

| **Benchmark** | **RoboCasa365** | | | | **LIBERO-Long** | | | |
|---|---|---|---|---|---|---|---|---|
| Horizon (steps) | 800–1000 | 1000–1400 | 1400–2900 | Overall | 250–340 | 340–400 | 400–550 | Overall |
| $\pi_0$ (Black et al., 2025b) | 26.0 | 23.0 | 8.1 | 15.5 | 89.6 | 90.0 | 42.0 | 80.2 |
| RLDX-1 (Kim et al., 2026) | 59.0 | 55.0 | 30.8 | 43.6 | 98.0 | 94.7 | 85.0 | 94.4 |
| GR00T-N1.5 (Bjorck et al., 2025) | 51.7 | 48.0 | 27.9 | 38.6 | 96.0 | 82.0 | 90.0 | 90.6 |
| w/ Sparse-RL (Zang et al., 2026) | 60.3 | 52.3 | 25.1 | 40.2 | 92.4 | 90.0 | 90.0 | 91.2 |
| w/ SimpleVLA-RL (Li et al., 2026) | 55.1 | 50.8 | 30.6 | 41.5 | 94.0 | 90.7 | 91.0 | 92.4 |
| w/ PolicyTrim (Wang et al., 2026) | 49.3 | 50.6 | 29.5 | 39.8 | 92.0 | 90.0 | 92.0 | 91.4 |
| w/ StructRL (*ours*) | 63.0 | 59.0 | 37.6 | 49.1 | 96.0 | 97.3 | 97.0 | 96.6 |
| $\Delta$ | +2.7 | +6.7 | +7.0 | **+7.6** | +2.0 | +6.6 | +5.0 | **+4.2** |
| $\pi_{0.5}$ (Black et al., 2025a) | 41.0 | 46.4 | 32.9 | 39.3 | 94.0 | 96.0 | 74.0 | 90.6 |
| w/ Sparse-RL (Zang et al., 2026) | 43.0 | 48.4 | 35.8 | 41.1 | 97.6 | 94.7 | 77.0 | 92.6 |
| w/ SimpleVLA-RL (Li et al., 2026) | 50.7 | 46.8 | 35.6 | 41.9 | 97.2 | 96.7 | 82.0 | 94.0 |
| w/ PolicyTrim (Wang et al., 2026) | 48.7 | 46.0 | 33.5 | 40.3 | 95.6 | 90.7 | 83.0 | 91.6 |
| w/ StructRL (*ours*) | 54.0 | 51.2 | 39.4 | 45.8 | 98.0 | 98.0 | 89.0 | 96.2 |
| $\Delta$ | +3.3 | +2.8 | +3.6 | **+3.9** | +0.4 | +1.3 | +6.0 | **+2.2** |

is inside receptacle $r$. Appendix B.2 provides details on the grounding procedure. We precompute the reference durations $T_d(v_i)$ for each subtask from the demonstrations used for SFT. Unless otherwise specified, we set $\beta = 0.6$ and $\lambda_c = 2.0$. The action-chunk length is $L = 16$ for GR00T-N1.5 and $L = 10$ for $\pi_{0.5}$. All online RL methods are trained for 100 iterations under the same environment-interaction budget. On RoboCasa365, each iteration collects 512 rollouts using two rollout epochs with 256 parallel environments; on LIBERO-Long, each iteration collects 768 rollouts using three rollout epochs with 256 parallel environments. Training uses four nodes with eight NVIDIA A100 GPUs per node and takes approximately 48 hours for RoboCasa365 or 24 hours for LIBERO-Long.

### 4.2 Main Results

Table 1 reports SR by task-horizon bucket on RoboCasa365 and LIBERO-Long. For GR00T-N1.5, StructRL improves overall SR over the strongest online RL baseline from 41.5% to 49.1% on RoboCasa365 and from 92.4% to 96.6% on LIBERO-Long, gains of 7.6 and 4.2 percentage points, respectively. For $\pi_{0.5}$, the corresponding improvements are 3.9 percentage points on RoboCasa365 and 2.2 percentage points on LIBERO-Long.

At the bucket level, StructRL exceeds the strongest online RL baseline in every column. The largest positive gains are 7.0 percentage points on RoboCasa365 (1400–2900) and 6.6 percentage points on LIBERO-Long (340–400), both with GR00T-N1.5. These results show that the structured reward improves online RL performance across both evaluated backbones and benchmarks, with substantial gains on several longer-horizon buckets.

### 4.3 Reward Source: Structured Events *vs*. Learned Progress Model

To isolate the effect of reward construction, we compare a Robometer-based reward baseline and StructRL under a matched PPO training setup. Robometer (Liang et al., 2026) is a vision-language reward model built on Qwen3-VL-4B (Bai et al., 2025) and trained on 1M trajectories, including LIBERO-Long, to predict scalar per-frame progress. We integrate its released checkpoint into the same RLinf-VLA stack and convert its progress estimates into rewards delivered once per action chunk. For each backbone, both methods use the same SFT initialization, PPO optimizer and hyperparameters, interaction budget, terminal reward, and evaluation protocol. We additionally match Robometer's intermediate-reward

Table 2: Reward source comparison under a matched PPO training protocol on LIBERO-Long.

| | **LIBERO-Long** | |
|---|---|---|
| Reward Source | GR00T-N1.5 | $\pi_{0.5}$ |
| SFT | 90.6 | 90.6 |
| Robometer (Liang et al., 2026) | 94.2 | 93.0 |
| StructRL (*ours*) | **96.6** | **96.2** |

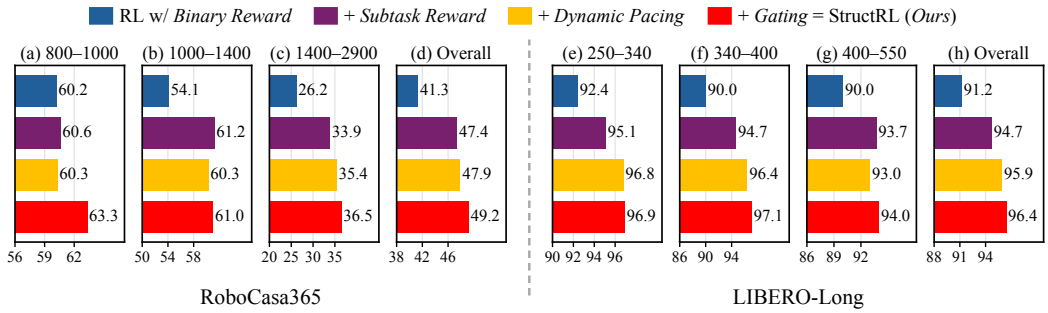



Figure 3: Reward-component ablation with PPO and GR00T-N1.5, adding one component at a time to the terminal binary reward: fixed subtask rewards, dynamic reward pacing, and structure-aware gating, which together form StructRL. Panels report SR (%) by task-horizon bucket and overall on RoboCasa365 (a–d) and LIBERO-Long (e–h). Values are means over three evaluation runs.

scale to that of StructRL on reference rollouts (Appendix F). Thus, the comparison contrasts learned progress rewards with simulator-verified, dependency-aware completion rewards.

Table 2 summarizes the results. StructRL outperforms Robometer on both backbones, by 2.4 points with GR00T-N1.5 and 3.2 points with $\pi_{0.5}$. This suggests that simulator-verified structured completion signals provide more effective intermediate supervision than the progress estimates inferred from visual observations. StructRL also does not require a learned reward model or reward model inference during rollout collection.

### 4.4 Reward-Component Ablation

We measure the contribution of each reward component by adding the components one at a time to PPO with GR00T-N1.5. We compare four configurations: (1) a terminal binary reward only; (2) fixed subtask rewards without gating, which pay $\beta/2$, the value of Eq. (4) at $T = T_d$, to each newly detected subtask completion regardless of its prerequisites; (3) dynamic reward pacing, which replaces the fixed reward with the pace-dependent reward in Eq. (4) while remaining ungated; and (4) structure-aware gating applied together with dynamic pacing, yielding the complete StructRL reward. Figure 3 reports SR by task-horizon bucket and overall.

Each added component raises overall SR on both benchmarks. Subtask rewards provide the largest gain, increasing SR from 41.3% to 47.4% on RoboCasa365 and from 91.2% to 94.7% on LIBERO-Long. This demonstrates the primary benefit of providing intermediate supervision through verifiable subtask completions. Dynamic pacing further improves SR by 0.5 and 1.2 points, respectively, while structure-aware gating adds another 1.3 and 0.5 points, reaching final SRs of 49.2% and 96.4%. The complete reward achieves the highest SR in most horizon buckets. Its largest end-to-end gain occurs on the longest RoboCasa365 tasks, where SR increases by 10.3 points, from 26.2% to 36.5%.

### 4.5 Effect of Subtask Decomposition Density

Finer decompositions provide more opportunities for intermediate supervision, but adding more completion signals may not always improve learning. We study this tradeoff using GR00T-N1.5 on the 16 RoboCasa365 tasks. Let $\bar{N}$ denote the average number of subtask completion signals per task. We evaluate eight settings, from $\bar{N} = 0$, which uses only the terminal binary reward, to increasingly fine-grained decompositions generated by the LLM. For this controlled sweep, we relax the default filtering criteria and permit fine motion checkpoints that would normally be excluded as weak indicators of task progress. Every retained checkpoint still has an executable binary predicate so that it can be evaluated during RL. We train a separate

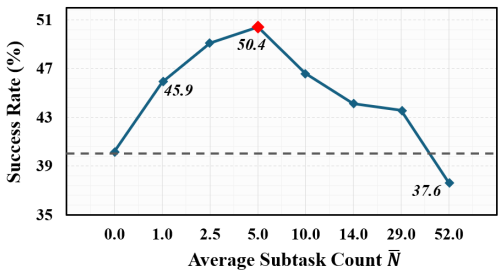



Figure 4: Effect of subtask decomposition density on RoboCasa365 using GR00T-N1.5.

Table 3: Effect of the policy optimizer on RoboCasa365. We train StructRL with PPO or GRPO on GR00T-N1.5 and $\pi_{0.5}$ and report SR (%) by task-horizon bucket. SimpleVLA-RL is included for reference.

| Method | 800–1000 | 1000–1400 | 1400–2900 | Overall |
|---|---|---|---|---|
| *GR00T-N1.5* | | | | |
| w/ SimpleVLA-RL | 55.1 | 50.8 | 30.6 | 41.5 |
| w/ StructRL | 63.0 | 59.0 | 37.6 | **49.1** |
| w/ StructRL-GRPO | 61.3 | 55.4 | 32.2 | 44.9 |
| $\pi_{0.5}$ | | | | |
| w/ SimpleVLA-RL | 50.7 | 46.8 | 35.6 | 41.9 |
| w/ StructRL | 54.0 | 51.2 | 39.4 | **45.8** |
| w/ StructRL-GRPO | 51.0 | 48.4 | 37.0 | 43.2 |

Table 4: Zero-shot evaluation on additional RoboCasa365 suites using GR00T-N1.5. Policies from Table 1 are evaluated without further training on 16 held-out long-horizon tasks from Composite-Unseen and 65 atomic tasks from Atomic-Seen.

| Method | Composite Unseen | Atomic Seen | **Average** |
|---|---|---|---|
| GR00T-N1.5 | 3.5 | 17.0 | 14.4 |
| w/ SimpleVLA-RL | 4.3 | 16.9 | 14.5 |
| w/ StructRL (*ours*) | 4.8 | 20.5 | 17.6 |
| $\Delta$ | +0.5 | +3.5 | **+3.1** |

policy for each setting and report its overall SR in Figure 4. Appendix C illustrates the decomposition ladder.

Intermediate supervision helps even at low density: increasing $\bar{N}$ from 0 to 1 raises SR from 40.2% to 45.9%. The benefit is not monotonic, however. Performance reaches 50.4% at $\bar{N} = 5$ and then declines as the decomposition becomes finer. At $\bar{N} = 52$, SR falls to 37.6%, below the terminal-only configuration. Thus, within this controlled sweep, a moderate decomposition density is more effective than either terminal-only reward or an excessively fine decomposition. Controlled analyses in Appendix C.3 indicate that the decline reflects both earlier, less task-aligned completion signals and growth in the total intermediate reward as more signals are added.

### 4.6 Compatibility with GRPO

We evaluate whether the structured reward remains effective when PPO is replaced by GRPO. On RoboCasa365, we train GR00T-N1.5 and $\pi_{0.5}$ using the rollout-level formulation from Section 3.2.3. Table 3 reports the results. StructRL-GRPO exceeds SimpleVLA-RL in every horizon bucket, improving overall SR by 3.4 percentage points with GR00T-N1.5 and 1.3 percentage points with $\pi_{0.5}$. This result shows that the structured reward is compatible with the evaluated GRPO formulation. The PPO variant remains stronger overall, by 4.2 and 2.6 percentage points on the two backbones, respectively, supporting its use as the default optimizer. One possible explanation is the difference in credit granularity: PPO associates each reward with the action chunk that triggers it, whereas GRPO aggregates all chunk-level rewards into one rollout return before computing group-relative advantages.

### 4.7 Zero-Shot Evaluation on Additional RoboCasa365 Tasks

We evaluate the GR00T-N1.5 policies from Table 1 without further training on two RoboCasa365 suites excluded from RL training: 16 held-out long-horizon tasks from Composite-Unseen and 65 atomic tasks from Atomic-Seen. Table 4 reports both evaluations.

On Composite-Unseen, SR remains low for all evaluated policies. StructRL reaches 4.8%, compared with 4.3% for the strongest non-StructRL comparison. These results indicate limited zero-shot transfer to unseen task compositions under the current setup.

Atomic-Seen provides a complementary test of forgetting: these tasks were encountered during pretraining but excluded from composite-task RL. As shown in Table 4, StructRL reaches 20.5% SR, compared with 17.0% for the SFT policy and 16.9% for SimpleVLA-RL. Thus, the composite-task RL update does not reduce Atomic-Seen performance under StructRL; instead, the evaluated checkpoint improves over SFT. One possible explanation is that intermediate completion rewards reinforce skills shared across atomic and composite tasks, although this table does not isolate that mechanism.

### 4.8 Applicability to Shorter-Horizon Tasks

Structured intermediate rewards are motivated by the difficulty of assigning credit over long horizons. We therefore ask whether the same framework remains useful when tasks are shorter and baseline

performance is already high. We evaluate StructRL on the LIBERO Spatial, Object, and Goal suites using GR00T-N1.5, training a separate policy on each suite with the same decomposition and optimization procedures used for LIBERO-Long. We make no short-task-specific modification; for example, a pick-and-place task can provide separate completion signals for grasping the object and placing it at the target.

As shown in Table 5, StructRL improves over SimpleVLA-RL from 91.2% to 92.1% on Spatial, from 98.5% to 99.4% on Object, and from 94.8% to 95.6% on Goal. These gains are modest, consistent with the high baseline success rates, but positive on all three suites. The results show that the same structured-reward pipeline remains effective when task structure is shallower, while the larger benefits remain concentrated in the long-horizon setting for which the method is designed.

Table 5: Success rate (%) on the LIBERO Spatial, Object, and Goal suites using GR00T-N1.5. A separate policy is trained on each suite using the same optimization recipe.

| Method | Spatial | Object | Goal | **Average** |
|---|---|---|---|---|
| GR00T-N1.5 | 89.3 | 97.9 | 94.6 | 93.7 |
| w/ SimpleVLA-RL | 91.2 | 98.5 | 94.8 | 94.7 |
| w/ StructRL (*ours*) | 92.1 | 99.4 | 95.6 | 95.7 |
| $\Delta$ | +0.9 | +0.9 | +0.8 | **+1.0** |

**Additional details and experimental results.** The appendix provides the complete training algorithm (Appendix A); decomposition details and decomposer sensitivity (Appendices B and B.3); density, reward-mechanism, and pacing analyses (Appendices C.1–C.4); repeated evaluations, training dynamics, and reward sensitivity (Appendices D.1–D.3); case studies (Appendix E); and Robometer implementation details (Appendix F).

## 5 Related Work

**Long-horizon VLA tasks.** VLA models have achieved strong performance on conventional manipulation tasks, including individual pick-and-place skills (Black et al., 2025a; Bjorck et al., 2025; Physical Intelligence et al., 2025). Recent benchmarks extend evaluation to tasks that compose multiple skills, using either physical robots (Wu et al., 2025; Fu et al., 2024) or reproducible simulators (Liu et al., 2023; Han et al., 2025; Nasiriany et al., 2026). For example, storing leftovers in RoboCasa365 requires the policy to sort food into containers, carry them across the kitchen, and place them in a refrigerator. Such tasks jointly test manipulation, instruction following, navigation, and extended execution. Because these policies are commonly trained through SFT on demonstrations, their performance remains sensitive to demonstration coverage and compounding execution errors. StructRL uses online interaction and intermediate rewards to improve these policies after SFT.

**Reinforcement learning for VLAs.** RL is increasingly used to post-train VLA policies. Offline approaches optimize from pre-collected rollouts (Zhang et al., 2024; 2025; Chen et al., 2025b; Huang et al., 2025), while recent online methods collect new interactions and optimize with PPO (Zang et al., 2026; Lu et al., 2025; Liu et al., 2025; Chen et al., 2025a) or GRPO (Li et al., 2026; Wang et al., 2026; Chen et al., 2025c; Tan et al., 2025b). Many online methods provide reward only after the complete rollout succeeds, making the signal increasingly sparse as the execution horizon grows. A complementary line of work provides denser supervision using learned progress or value estimators (Shu et al., 2025; Zhang et al., 2026; Tan et al., 2025a; Liang et al., 2026). However, these estimators may require a large model during rollout collection or task-specific reward-model training. StructRL derives intermediate rewards directly from simulator-verifiable subtask completions and structures them with prerequisite dependencies and demonstration-derived pacing, eliminating the need for a learned reward model during rollout collection.

## 6 Conclusion

We presented StructRL, an online RL framework that replaces terminal-only supervision with structured intermediate rewards for long-horizon VLA tasks. StructRL decomposes each instruction into verifiable subtasks, uses their dependencies to determine when a completion constitutes valid progress, and scales the resulting reward according to completion pace. Across RoboCasa365 and LIBERO-Long, StructRL improves GR00T-N1.5 and $\pi_{0.5}$ over the evaluated online RL baselines, with the largest gains on longer-horizon tasks for GR00T-N1.5. These results show that verifiable subtask rewards, organized by task structure and calibrated to demonstration pace, can improve long-horizon VLA post-training.

### AI USE STATEMENT

We used generative AI tools to assist with experimental infrastructure, log analysis, and drafting and revising manuscript text. The authors independently verified all reported results, numerical values, and claims and take full responsibility for the paper.

### ETHICS STATEMENT

Online VLA RL requires substantial computation. A typical run uses 32 NVIDIA A100 GPUs for approximately 48 hours on RoboCasa365 or 24 hours on LIBERO-Long, corresponding to about 1,536 and 768 GPU-hours, respectively.

Shaped rewards can induce unintended behavior. In some rollouts, policies collected intermediate rewards and then remained idle until timeout. Dynamic reward pacing discourages delaying future subtask completions because the available reward decreases with completion time, but it does not eliminate idling after intermediate rewards have already been collected. Other reward-induced failure modes may therefore remain. Because StructRL favors faster subtask completion, deployment on physical robots would require separate safety validation.

### REPRODUCIBILITY STATEMENT

We provide the training and evaluation code, generated subtask decompositions, and reward implementation in the supplementary material. The main text and appendix specify the reward formulation, the reward hyperparameters, action-chunk lengths, reference completion durations, and interaction budget. We release the generated subtask graphs as static configuration files, allowing the experiments to be reproduced without querying an LLM or depending on a particular model version.

Evaluation follows each benchmark's official task horizons and initial-state protocol. RoboCasa365 uses 16 composite tasks with 100 episodes per task, while LIBERO-Long uses 10 tasks with 50 episodes per task. We compute macro SR as the mean of per-task success rates and state the number of repeated runs wherever a standard deviation is reported.

## REFERENCES

Anthropic. Claude opus 4.8 system card. Technical report, Anthropic, 2026. URL https://www-cdn.anthropic.com/0b4915911bb0d19eca5b5ee635c80fef830a37ea.pdf.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang

Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. $\pi_0$: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025b.

Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Xiang Li, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, Tiejun Huang, Yu Wang, and Chao Yu. $\pi_{\texttt{rl}}$: Online RL fine-tuning for flow-based vision-language-action models. *arXiv preprint arXiv:2510.25889*, 2025a.

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. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025b.

Zengjue Chen, Runliang Niu, He Kong, Qi Wang, Qianli Xing, and Zipei Fan. TGRPO: Fine-tuning vision-language-action model via trajectory-wise group relative policy optimization, 2025c.

Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile ALOHA: Learning bimanual mobile manipulation using low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, 2024.

Gemini Robotics Team. Gemini robotics: Bringing AI into the physical world, 2025.

Songhao Han, Boxiang Qiu, Yue Liao, Siyuan Huang, Chen Gao, Shuicheng Yan, and Si Liu. RoboCerebra: A large-scale benchmark for long-horizon robotic manipulation evaluation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Dongchi Huang, Zhirui Fang, Tianle Zhang, Yihang Li, Lin Zhao, and Chunhe Xia. CO-RFT: Efficient fine-tuning of vision-language-action models through chunked offline reinforcement learning, 2025.

Dongyoung Kim et al. RLDX-1 technical report, 2026.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Conference on Robot Learning (CoRL)*, 2024.

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. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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, 2025.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026.

Physical Intelligence, Ali Amin, et al. $\pi^*_{0.6}$: a VLA that learns from experience, 2025.

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

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Junyang Shu, Zhiwei Lin, and Yongtao Wang. RFTF: Reinforcement fine-tuning for embodied agents with temporal feedback, 2025.

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, 2025a.

Shuhan Tan, Kairan Dou, Yue Zhao, and Philipp Krähenbühl. Interactive post-training for vision-language-action models, 2025b.

Xianghui Wang, Feng Chen, Wenbo Zhang, Hua Yan, Zixuan Wang, Changsheng Li, and Yinjie Lei. PolicyTrim: Boosting intrinsic policy efficiency of vision-language-action models, 2026.

Kun Wu et al. RoboMIND: Benchmark on multi-embodiment intelligence normative data for robot manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025.

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Peihong Wang, 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 reinforcement learning of vision-language-action models. In *Proceedings of Robotics: Science and Systems (RSS)*, 2026. URL https://roboticsconference.org/program/papers/89/.

Hongyin Zhang, Zifeng Zhuang, Han Zhao, Pengxiang Ding, Hongchao Lu, and Donglin Wang. ReinboT: Amplifying robot visual-language manipulation with reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2025.

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 *International Conference on Machine Learning (ICML)*, 2026.

Zijian Zhang, Kaiyuan Zheng, Zhaorun Chen, Joel Jang, Yi Li, Siwei Han, Chaoqi Wang, Mingyu Ding, Dieter Fox, and Huaxiu Yao. GRAPE: Generalizing robot policy via preference alignment, 2024.

# APPENDIX

## CONTENTS

**A Training Procedure** **14**

**B Subtask Decomposition** **14**

B.1 LLM Decomposition Prompt . . . . . 14

B.2 Grounding Subtasks to Simulator Predicates . . . . . 15

B.3 Sensitivity to the Decomposer . . . . . 15

**C Further Analysis of Subtask Decomposition Granularity** **16**

C.1 Constructing the Density Ladder . . . . . 17

C.2 Effect of Decomposition Density Across Horizon Buckets . . . . . 17

C.3 Understanding Performance Degradation Under Dense Decomposition . . . . . 18

C.4 Does Reward Pacing Require Demonstration Durations? . . . . . 19

**D Additional Experimental Results** **19**

D.1 Multi-seed Evaluation . . . . . 19

D.2 Training Dynamics . . . . . 20

D.3 Sensitivity to the Subtask-completion Reward . . . . . 20

**E Case Study** **21**

**F Robometer Baseline Implementation** **22**

**G Limitations** **23**

# A TRAINING PROCEDURE

Algorithm 1 states the full training loop of STRUCTRL in one place: rollout collection, the structure-aware reward gating and dynamic reward pacing described in Section 3, and the PPO update.

**Algorithm 1** Overall Training Process of STRUCTRL

**Require:** SFT-initialized VLA policy $\pi_\theta$; decomposition $\{(v_i, X(v_i))\}_{i=1}^{N}$ from Eq. (2); demonstration durations $\{T_d(v_i)\}_{i=1}^{N}$; hyperparameters $\beta$, $\lambda_c$

**for** each training iteration **do**
  **// Step 1: Rollout collection**
  Roll out $\pi_\theta$ to collect $\tau = (o_1, a_1, \ldots, o_M, a_M)$
  **// Step 2: Structured reward computation**
  $\mathcal{C} \leftarrow \varnothing$; $k_{\text{last}} \leftarrow 0$; $r_k \leftarrow 0$ for all $k$
  **for** $k = 1, \ldots, M$ **do**
    # Reward Gating
    **for all** $v_i \notin \mathcal{C}$ with $X(v_i) \subseteq \mathcal{C}$ **do**
      **if** $v_i$ is completed during $a_k$ **then**
        $T(v_i) \leftarrow k - k_{\text{last}}$
        # Reward Pacing
        $r_k \leftarrow r_k + \beta / \big(1 + T(v_i)/T_d(v_i)\big)$
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_i\}$; $k_{\text{last}} \leftarrow k$
      **end if**
    **end for**
  **end for**
  $r_M \leftarrow r_M + \lambda_c$ **if** the task is completed at $a_M$
  **// Step 3: Policy optimization**
  Update $\pi_\theta$ via PPO with $\tau$ and $\{r_k\}_{k=1}^{M}$ # Eq. (1)
**end for**
**return** optimized VLA policy $\pi_\theta$

# B SUBTASK DECOMPOSITION

## B.1 LLM DECOMPOSITION PROMPT

We provide the prompt used to decompose each long-horizon task (Section 3.1). The LLM receives only the requirements below and the benchmark's natural-language task command. It is given no simulator predicate vocabulary, demonstration data, or example decomposition, so the proposal depends only on the task command. Line breaks inside the prompt are for typesetting only.

```
You are a robot task planner. Decompose a kitchen manipulation
task (performed by a single-arm mobile robot) into stages of
subtasks.

Requirements:
1. Output STAGES in strict execution order: stage k+1 can only
   start after every subtask in stage k is done.
2. Subtasks WITHIN one stage may be completed in any order
   (parallel set).
3. Each subtask must be ONE atomic, physically checkable
   manipulation event, phrased as a short verb phrase in this
   controlled form:
   - "grasp <object>"
   - "place <object> in/on <receptacle or location>"
   - "open <fixture>" / "close <fixture>"
   - "turn on <fixture>" / "turn off <fixture>"
   - "press <button>"
   - "<activity> for a while" for continuous activities
     (stirring, washing, scrubbing), optionally split into
     progressive milestones.
4. Include intermediate manipulation events (like grasping an
   object before placing it), not just final outcomes.
```

```
5. Do NOT include a final "task finished" subtask, and do NOT
   include robot retreat/release-and-back-away steps. Each retained
   subtask should correspond to a meaningful state transition that
   reflects genuine progress toward the task goal.
6. Output ONLY a JSON object:
   {"stages": [["subtask", ...], ...]} - no prose.
```

The user turn is the single line `Task command: "{cmd}"` followed by `Decompose this task now.`, where `{cmd}` is the benchmark command. Requirement 3 is the *verifiability* constraint of Section 3.1 expressed as a controlled output form, and requirement 5 excludes the two classes of non-progress event that constitute the *progress* constraint. Decoding is greedy (`do_sample=False`), so a proposal is reproducible given the model. Table 7 lists the grounded decompositions used in RoboCasa365 experiments.

### B.2 Grounding Subtasks to Simulator Predicates

Before RL training, each retained natural-language subtask is deterministically grounded to a binary predicate over simulator state. Grounding is performed once per task, and the resulting predicates are fixed throughout training.

**RoboCasa365 grounding.** We implement predicates using the benchmark's simulator state and native task-success utilities, with the same thresholds where applicable. Each supported verb form maps to a predicate template, while its arguments bind the template to the relevant task objects or fixtures. Table 6 summarizes the mappings.

Table 6: Predicate templates for grounding RoboCasa365 subtasks. $o$, $r$, and $f$ denote a bound object, receptacle, and fixture.

| Subtask phrase | Predicate is true when | Example |
|---|---|---|
| place $o$ in/on $r$ | $o$ is contained in $r$ or rests on it | `chicken_in_bowl` |
| grasp $o$ | $o$ is held by the gripper; once detected, the predicate is latched | `grasp_straw` |
| open / close $f$ | the relevant door or drawer crosses the benchmark threshold | `dishwasher_closed` |
| turn on / off $f$, press $f$ | $f$ reaches the corresponding discrete state | `water_on` |
| ⟨activity⟩ for a while | a simulator-maintained activity timer reaches a specified threshold | `wash_t10` |

Object and fixture mentions are bound to the simulator handles associated with the task command. When multiple instances of the same category occur, repeated mentions are assigned to the corresponding task instances.

**Filtering.** We discard grounded subtasks if (i) no supported predicate template exists, (ii) the predicate is already satisfied at reset or does not reliably indicate task progress, (iii) it cannot be reliably timed from the SFT demonstrations, or (iv) it duplicates another predicate. For timing, we replay the 100 demonstrations for each task and remove predicates that trigger too infrequently to estimate $T_d$ or whose relevant state is not reproduced under replay. The LLM-provided dependency order is preserved after filtering, with empty groups removed.

**LIBERO.** On LIBERO, we directly use the goal conjuncts from each task's BDDL specification, evaluated by the benchmark's native goal checker. For example, *put the black bowl in the bottom drawer of the cabinet and close it* yields an `In` predicate for the bowl and drawer, followed by a `Close` predicate for the drawer. Goal conjuncts already satisfied at reset are removed. For the Spatial, Object, and Goal suites, we additionally insert a grasp predicate before each placement predicate; articulated, pushing, and knob tasks retain a single goal group.

### B.3 Sensitivity to the Decomposer

Because the decomposition is authored by an LLM, we test sensitivity to the model choice. Table 8 compares the default decomposer (Claude Opus 4.8) against a weaker open-weight model, Qwen3.5-9B Qwen Team (2026), under identical training hyperparameters. Replacing Claude Opus 4.8 with

Table 7: Subtask decompositions for the 16 RoboCasa365 composite-seen tasks at $\bar{N} = 2.375$ and 5.0. Braces group unordered subtasks within a stage, while arrows indicate strict stage ordering. $N$ denotes the number of progress signals. The $\bar{N} = 5.0$ setting additionally includes grasp milestones, intermediate fixture states, and timed activity checkpoints. Retreat signals are appended automatically and excluded from $N$.

| **Task** | $\bar{N} = 2.375$ | | $\bar{N} = 5.0$ | |
|---|---|---|---|---|
| | $N$ | **Progress signals** | $N$ | **Progress signals** |
| *DeliverStraw* | 1 | {straw_in_glass_cup} | 2 | {grasp_straw} → {straw_in_glass_cup} |
| *GetToastedBread* | 2 | {toaster_on} → {toast_on_plate} | 3 | {toaster_on} → {grasp_toast} → {toast_on_plate} |
| *KettleBoiling* | 2 | {kettle_on_stove} → {kettle_on_active_burner} | 3 | {grasp_kettle} → {kettle_on_stove} → {kettle_on_active_burner} |
| *LoadDishwasher* | 3 | {dish0_on_rack, dish1_on_rack} → {dishwasher_closed} | 6 | {grasp_dish0, dish0_on_rack, grasp_dish1, dish1_on_rack} → {door_half_closed} → {dishwasher_closed} |
| *PackIdenticalLunches* | 2 | {tupper0_complete, tupper1_complete} | 8 | {grasp_vegetable0, vegetable0_packed, grasp_vegetable1, vegetable1_packed, grasp_meat0, meat0_packed, grasp_meat1, meat1_packed} |
| *PreSoakPan* | 3 | {water_on, pan_in_sink, sponge_in_sink} | 5 | {grasp_pan, pan_in_sink, grasp_sponge, sponge_in_sink, water_on} |
| *PrepareCoffee* | 2 | {mug_at_machine} → {machine_turned_on} | 3 | {grasp_mug} → {mug_at_machine} → {machine_turned_on} |
| *RinseSinkBasin* | 3 | {washed_left, washed_center, washed_right} | 4 | {water_on} → {washed_left, washed_center, washed_right} |
| *ScrubCuttingBoard* | 2 | {contact_5steps, sweep_range_0p1m} | 6 | {grasp_sponge} → {contact_1step} → {contact_3steps, sweep_range_0p05m} → {contact_5steps, sweep_range_0p1m} |
| *SearingMeat* | 2 | {meat_in_pan} → {pan_on_active_knob} | 5 | {grasp_pan, pan_on_stove, pan_on_active_knob} → {grasp_meat, meat_in_pan} |
| *SetUpCuttingStation* | 2 | {meat_on_board, knife_on_board} | 4 | {grasp_meat, meat_on_board, grasp_knife, knife_on_board} |
| *StackBowlsCabinet* | 2 | {bowls_stacked} → {any_bowl_in_cabinet} | 4 | {grasp_any_bowl} → {bowls_stacked} → {any_bowl_in_cabinet, both_bowls_in_cabinet} |
| *SteamInMicrowave* | 3 | {veg_in_bowl} → {bowl_in_micro} → {door_closed} | 6 | {grasp_vegetable, veg_in_bowl} → {grasp_bowl, bowl_in_micro} → {door_half_closed, door_closed} |
| *StirVegetables* | 4 | {veg1_in_pot, veg2_in_pot} → {spatula_grasped} → {task_complete} | 8 | {grasp_veg1, veg1_in_pot, grasp_veg2, veg2_in_pot} → {spatula_grasped} → {stir_t1, stir_t3} → {task_complete} |
| *StoreLeftoversInBowl* | 3 | {chicken_in_bowl, vegetable_in_bowl} → {bowl_in_fridge} | 6 | {grasp_chicken, chicken_in_bowl, grasp_vegetable, vegetable_in_bowl} → {grasp_bowl, bowl_in_fridge} |
| *WashLettuce* | 2 | {water_on} → {task_complete} | 7 | {water_on} → {lettuce_under_water} → {wash_t5, wash_t10, wash_t15, wash_t20} → {task_complete} |
| **Total** | 38 | | 80 | |

Qwen3.5-9B reduces overall SR from 49.1% to 45.7%. This result shows that the pipeline can be instantiated with the evaluated open-weight decomposer, while the remaining gap indicates that decomposition quality affects downstream RL performance.

Table 8: Ablation on the LLM that authors the subtask decomposition. Given only the task language command and our decomposition requirements, each LLM proposes the stage/subtask split; STRUCTRL is then trained with identical hyperparameters. SR (%) grouped by horizon bucket. In Table 1, we use Claude Opus 4.8 by default for decomposition.

| Horizon (steps) | 800–1000 | 1000–1400 | 1400–2900 | Overall |
|---|---|---|---|---|
| Opus 4.8 | 63.0 | 59.0 | 37.6 | 49.1 |
| Qwen3.5-9B | 59.3 | 60.6 | 31.2 | 45.7 |

## C FURTHER ANALYSIS OF SUBTASK DECOMPOSITION GRANULARITY

Section 4.5 shows that performance peaks at moderate decomposition density and declines when the decomposition becomes excessively fine. This appendix details the density construction, reports

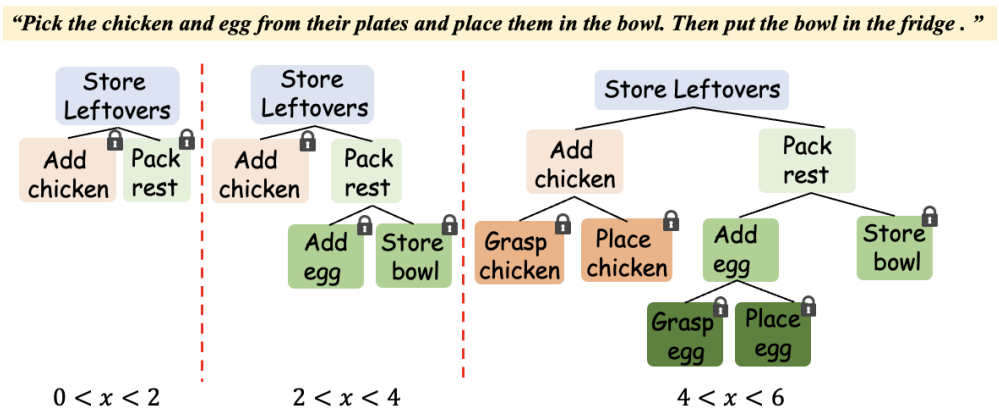



Figure 5: An example of hierarchical subtask decomposition under increasing reward density. Each level refines the previous tree by splitting parent subtasks into finer children while keeping all existing milestones unchanged; *lock icons* mark the subtasks whose completion will be detected and rewarded during RL training at the corresponding density level.

Table 9: Reward decomposition for *StoreLeftoversInBowl* at different densities. Here, $\bar{N}$ is the average number of gated subtask signals. The first four levels correspond to the trees in Fig. 5; denser levels progressively refine the same six milestones with finer verifiable motion checkpoints.

| **Density** $\bar{N}$ | **Rewarded subtasks for *StoreLeftoversInBowl*** |
|---|---|
| 0.0 | Terminal reward only. |
| 1.0 | Chicken in bowl. |
| 2.5 | Chicken in bowl; vegetable in bowl; bowl in fridge. |
| 5.0 | Grasp chicken; chicken in bowl; grasp vegetable; vegetable in bowl; grasp bowl; bowl in fridge. |
| 10.0 | Refine the six milestones with intermediate motion checkpoints: reach/approach before each grasp, and lift/carry/near-fridge before placement. |
| 14.0 | Further refine them with gripper closing, above-the-bowl checkpoints, bowl contact, and additional stages of the fridge approach. |
| 29.0 | Represent each milestone as a short chain of 5–6 checks, including staged distance reduction, contact, gripper closure, stable grasp, lift, and staged approach to the receptacle. |
| 52.0 | Use 10–11 checkpoints per milestone, covering progressively shrinking gripper–object distances, contact, finger closure, stable grasp, lift, receptacle approach, above, lowered, inside, and released states. |

results by horizon bucket, and analyzes two contributors to the decline. All experiments use GR00T-N1.5 on RoboCasa365 with fixed training hyperparameters unless stated otherwise.

### C.1 Constructing the Density Ladder

We first illustrate how a single task is progressively decomposed into subtask sets with different density levels. As shown in Figure 5, successive density levels are constructed by refining each existing milestone into finer-grained child subtasks while keeping all previously introduced milestones intact, thereby progressively increasing the number of subtasks. During this process, we remove the verifiability and progress constraints used in our default subtask decomposition procedure, allowing the LLM to decompose each task as finely as possible. Finally, Table 9 provides one representative task and lists the specific subtasks corresponding to each density level for reference.

### C.2 Effect of Decomposition Density Across Horizon Buckets

We next investigate how decomposition density affects tasks with different horizons. Figure 6 plots the SR trends for the three horizon buckets, with the exact value of each point reported in Table 10. We make two observations. First, all three horizon buckets exhibit a consistent trend: performance

Figure 6: SR trends across decomposition densities for different horizon buckets.

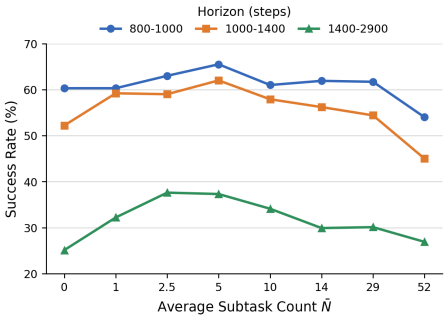



Table 10: SR values for different horizon buckets at each decomposition density. Each value corresponds to one point in Figure 6.

| Avg. Subtask $\bar{N}$ | 800–1000 | 1000–1400 | 1400–2900 | Overall |
|---|---|---|---|---|
| 0.0 | 60.3 | 52.3 | 25.1 | 40.2 |
| 1.0 | 60.3 | 59.2 | 32.2 | 45.9 |
| 2.375 | 63.0 | 59.0 | 37.6 | 49.1 |
| 5.0 | 65.5 | 62.0 | 37.3 | **50.4** |
| 10.0 | 61.0 | 57.9 | 34.1 | 46.6 |
| 14.0 | 61.9 | 56.2 | 29.9 | 44.1 |
| 29.0 | 61.7 | 54.4 | 30.1 | 43.6 |
| 52.0 | 54.0 | 45.0 | 26.9 | 37.6 |

initially improves as the decomposition becomes denser and then declines once the density becomes too high, further confirming the overall trend observed in Figure 4. Second, comparing the best density level ($\bar{N} = 5.0$) with the densest setting ($\bar{N} = 52.0$), SR decreases by 11.5, 17.0, and 10.4 points for the 800–1000, 1000–1400, and 1400–2900 horizon buckets, respectively. Relative to the peak performance of each bucket, however, these correspond to declines of approximately 18%, 27%, and 28%. Thus, the proportional degradation becomes larger for the two longer-horizon buckets, suggesting that excessively dense decomposition is increasingly detrimental as task horizon grows.

### C.3 Understanding Performance Degradation Under Dense Decomposition

Increasing subtask decomposition density changes two properties simultaneously. First, it changes when subtask completion events are detected and rewarded. Second, because each retained subtask can contribute an intermediate reward, it increases the total intermediate reward available in a rollout. Table 11 examines these factors using controls that hold subtask decomposition density and dependency structure fixed.

**Signal timing and alignment.** To isolate the effect of intermediate reward timing, we construct a controlled variant of the $\bar{N}$=5 decomposition while keeping the decomposition and dependency structure fixed. The reference setting uses 80 subtask-completion signals arranged into 41 ordered dependency groups. In the controlled variant, we modify only the predicate associated with each signal: the reference predicates detect the completion of each sub-motion, whereas the controlled predicates detect an earlier verifiable motion checkpoint within the same sub-motion. For example, for the straw-grasping subtask in *DeliverStraw*, the reference signal fires once the grasp is complete, while the controlled signal fires at an earlier verifiable state during the same grasping motion. Thus, the subtask definition and dependencies structure remain unchanged, and only the timing at which intermediate credit is provided is shifted earlier.

We retain only earlier predicates with sufficient demonstration coverage and non-degenerate firing behavior. This shifts 78 of the 80 completion signals earlier by an average of 3.1 action chunks; the remaining two retain their reference predicates because no valid earlier candidate exists. Under this controlled change, overall SR decreases from $50.4\%\pm1.0$ to $47.4\%\pm0.8$. This result indicates that intermediate reward timing matters even when decomposition density and dependency structure are held fixed.

**Total intermediate reward.** We next keep the $\bar{N}$=14 subtask decomposition unchanged, including all 224 subtask completion signals and their dependency structure, but cap the total intermediate reward available in a rollout. The cap prevents total intermediate reward from growing with the number of subtasks and raises SR from 44.1% to 47.7%, recovering more than half of the 6.3-point gap from the peak of the density sweep. Thus, the performance decline at high decomposition density is associated not only with subtask completion timing but also with the growth of total intermediate reward.

**Combined effect.** The two controls isolate comparable sources of degradation. Moving the $\bar{N}$=5 subtask completion signals to earlier motion checkpoints reduces SR by 3.0 points, while removing the reward cap at $\bar{N}$=14 reduces SR by approximately 3.6 points. Therefore, both subtask completion timing and total intermediate reward should be controlled as decomposition density changes.

Table 11: Controlled analyses of subtask reward timing and total intermediate reward on RoboCasa365 with GR00T-N1.5. Within each comparison, subtask decomposition density and dependency structure are fixed. Results are evaluated at training step 100 as mean ± sample standard deviation over three evaluation runs.

| Reward configuration | $\bar{N}$ | Overall SR (%) |
|---|---|---|
| *Signal timing and alignment* | | |
| Reference completion predicates | 5.0 | **50.4** ±1.0 |
| Earlier checkpoint predicates | 5.0 | 47.4 ±0.8 |
| *Total intermediate reward* | | |
| Unbounded (fixed per-subtask reward) | 14.0 | 44.1 ±1.8 |
| Total reward capped at $B$ | 14.0 | **47.7** ±0.4 |

### C.4 Does Reward Pacing Require Demonstration Durations?

Table 11 examined when intermediate rewards are triggered and how their total magnitude changes with decomposition density. We next isolate a different component: how dynamic reward pacing calibrates the magnitude of an eligible reward after subtask completion is detected. Holding the completion predicates, dependency structure, and decomposition density fixed at the default decomposition (Table 7), we replace only the demonstration-average completion duration $T_d(v_i)$ with $H/N$, where $H$ is the task horizon and $N$ is the number of retained subtasks. This uniform completion duration reduces SR from $49.2\%\pm1.1$ to $43.1\%\pm0.4$ over three evaluation runs (Table 12). Thus, uniform horizon allocation does not recover the performance obtained with demonstration-derived completion durations, while more adaptive demonstration-free estimates remain open.

Table 12: Completion durations used for dynamic reward pacing at the default subtask decomposition density. All other settings are fixed; results are mean ± standard deviation over three evaluation runs.

| Completion duration | Overall SR (%) |
|---|---|
| Demonstration average $T_d(v_i)$ | **49.2** ±1.1 |
| Uniform horizon allocation $H/N$ | 43.1 ±0.4 |

## D Additional Experimental Results

We include additional experiments in this section, including a multi-seed evaluation (Appendix D.1), training dynamics (Appendix D.2), and a hyperparameter sensitivity analysis (Appendix D.3).

### D.1 Multi-seed Evaluation

We first evaluate the robustness of our method across different random seeds. Specifically, for the representative baselines in Table 1 and our StructRL, we conduct three independent evaluation runs with different random seeds and report the mean and standard deviation of the resulting SR. As shown in Table 13, StructRL consistently achieves the best overall SR on both VLA backbones. On GR00T-N1.5, StructRL reaches an overall SR of $49.2\%$, outperforming the strongest baseline by 7.1 points, while on $\pi_{0.5}$ it achieves $44.8\%$, improving over the strongest baseline by 2.9 points. These consistent improvements across multiple random seeds demonstrate that the gains of StructRL are robust and are not attributable to a particular evaluation run.

Table 13: Multi-seed evaluation on RoboCasa365: mean ± std over 3 independent evaluation runs, SR (%) by horizon bucket. StructRL delivers the best overall SR on both backbones, with the largest margins on medium and long-horizon buckets.

| Method | 800–1000 | 1000–1400 | 1400–2900 | Overall |
|---|---|---|---|---|
| *GR00T-N1.5* | | | | |
| SFT | 52.6±0.8 | 48.3±0.6 | 26.6±1.2 | 38.2±0.5 |
| w/ Sparse-RL | 59.8±1.6 | 54.3±0.9 | 27.8±2.9 | 42.1±1.4 |
| w/ SimpleVLA-RL | 54.2±2.6 | 52.6±1.7 | 29.6±1.2 | 41.4±0.3 |
| w/ StructRL | 63.3±1.2 | 61.0±1.7 | 36.5±1.8 | **49.2**±1.1 |
| $\pi_{0.5}$ | | | | |
| SFT | 45.7±5.0 | 47.9±1.1 | 33.8±1.2 | 40.4±1.1 |
| w/ Sparse-RL | 44.2±1.1 | 45.9±2.2 | 35.2±0.9 | 40.2±0.9 |
| w/ SimpleVLA-RL | 47.8±1.0 | 48.1±0.8 | 35.8±0.7 | 41.9±0.4 |
| w/ StructRL | 52.3±2.6 | 51.3±0.3 | 38.0±2.2 | **44.8**±1.1 |

### D.2 Training Dynamics

We next examine the training dynamics of StructRL to evaluate its stability throughout RL training. Specifically, we save a checkpoint every 10 training steps and evaluate each checkpoint using the corresponding benchmark evaluation protocol. We conduct this experiment with GR00T-N1.5 on both RoboCasa365 and LIBERO-Long, and continue training for up to 150 steps. The resulting training curves are shown in Figure 7.

From Figure 7, StructRL exhibits stable training dynamics on both benchmarks. Its performance begins to separate from the baselines after roughly 50 training steps and remains consistently higher throughout the remainder of training, without any performance collapse as training proceeds. Moreover, the performance gains are already evident by around 100 training steps and persist afterward. Therefore, considering the additional computational cost of longer training, we uniformly report the results at training step 100 in our main experiments.

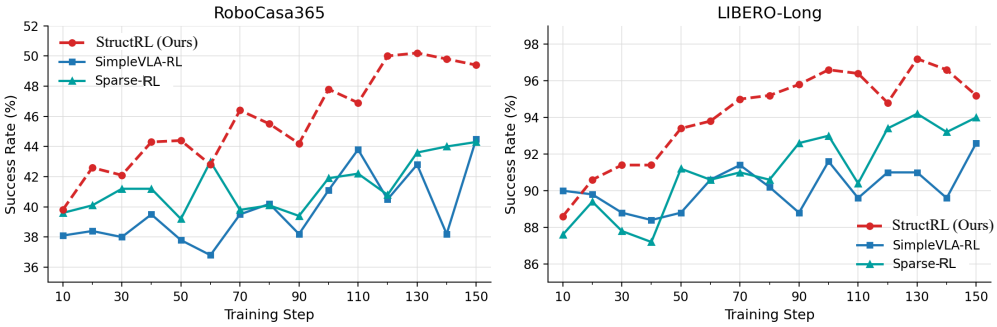



Figure 7: **Training dynamics.** Success rate of intermediate checkpoints (every 10 steps, offline evaluation) during RL training with GR00T-N1.5 on RoboCasa365 composite tasks and LIBERO-Long. StructRL consistently outperforms SimpleVLA-RL and Sparse-RL throughout training.

### D.3 Sensitivity to the Subtask-Completion Reward

In Eq. (4), we use the hyperparameter $\beta$ to control the magnitude of the dynamically paced subtask reward. We therefore examine the sensitivity of StructRL to the choice of $\beta$. Specifically, we fix the terminal reward $\lambda_c$ to 2.0 and vary $\beta$ over $\{0.2, 0.6, 1.0, 2.0\}$. We conduct the experiments on RoboCasa365 using GR00T-N1.5 and report the resulting SR in Figure 8. As shown in the figure, performance improves substantially when $\beta$ increases from 0.2 to 0.6, with SR increasing from 40.0% to 49.1%. Performance remains nearly unchanged at $\beta = 1.0$, reaching 49.0%, while a larger value of $\beta = 2.0$ reduces SR to 46.5%. These results indicate that StructRL is relatively insensitive

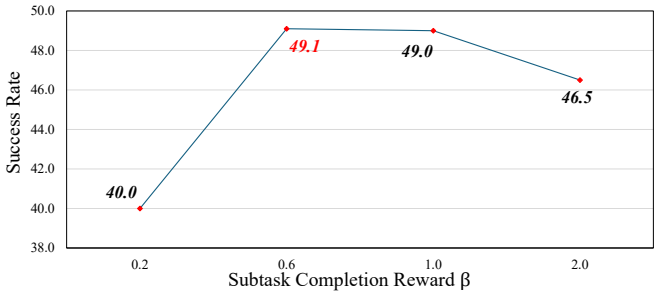



Figure 8: Sensitivity of StructRL to the subtask-completion reward $\beta$ on RoboCasa365 using GR00T-N1.5. The terminal reward is held fixed.

to $\beta$ within a moderate range, while assigning either too little or too much weight to subtask rewards can degrade performance.

## E Case Study

We further provide qualitative case studies to compare the execution behaviors of the SFT policy and StructRL on long-horizon VLA tasks. As shown in Figures 9 and 10, each example presents the task command together with a sequence of observations from the SFT policy and StructRL at several key moments during execution. By tracking the same agent view over time, the examples make it easier to compare how the two policies progress through the task and where their behaviors begin to diverge.

In Figure 9, both policies begin by manipulating the target objects, but StructRL continues to complete the required intermediate subtasks and eventually reaches the final stage of placing the bowl into the refrigerator, whereas the SFT policy fails to make comparable progress. In Figure 10, StructRL successfully progresses from grasping and placing the broccoli to moving the bowl into the microwave, while the SFT policy becomes stuck at an earlier manipulation stage. These qualitative examples illustrate that StructRL makes more consistent progress through long-horizon tasks and is less likely to get stuck at intermediate stages than the SFT policy.

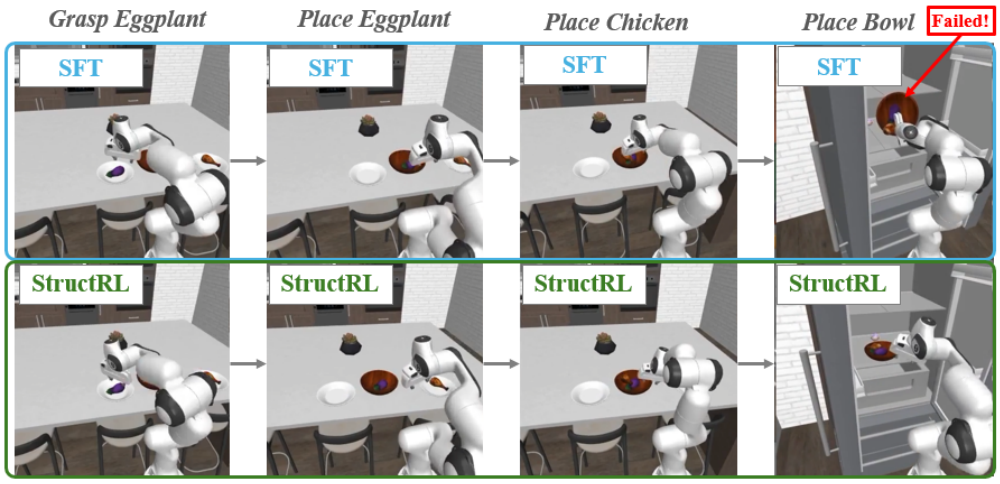



Figure 9: Qualitative comparison between SFT and StructRL on the long-horizon task "StoreLeftOversInBowl". StructRL successfully completes the intermediate object-placement steps and proceeds to placing the bowl in the refrigerator, while SFT fails before reaching the final stage.

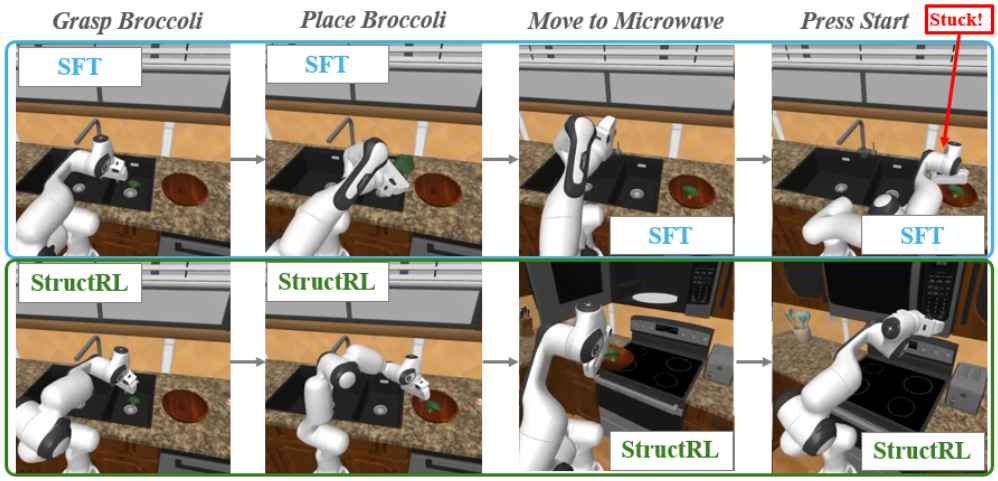



Figure 10: Qualitative comparison between SFT and StructRL on the long-horizon task "Steam-InMicrowave". StructRL completes the preceding subtasks and advances to interacting with the microwave, whereas SFT gets stuck at an earlier manipulation stage.

## F Robometer Baseline Implementation

We implement the learned dense-reward baseline using the released Robometer-4B checkpoint (Liang et al., 2026), without additional fine-tuning. Its training corpus includes LIBERO-Long (Liu et al., 2023) demonstrations and failure trajectories generated on the same tasks (Liang et al., 2026), so on LIBERO-Long the reward model is evaluated in-distribution. At each action-chunk boundary $k$, Robometer receives the task instruction and a causal visual history and predicts scalar progress $p_k \in [0, 1]$, computed as the expectation over its ten discrete progress bins. We convert this prediction to a chunk-level reward as

$$r_k^{\text{Robo}} = \beta_{\text{R}}\big(p_k - b_k\big) + \lambda_c \, \mathbb{1}[s_k = 1], \qquad b_k = \begin{cases} 0, & s_k = 1, \\ 1, & s_k = 0, \end{cases} \tag{5}$$

where $s_k = 1$ only when task success is first detected during chunk $k$, and $\lambda_c = 2.0$ is the same terminal reward used by StructRL. Before success, the reward is therefore $\beta_{\text{R}}(p_k - 1)$; at the first successful chunk, it becomes $\beta_{\text{R}} p_k + \lambda_c$. This follows the online-RL reward formulation used with Robometer (Liang et al., 2026), adapted from simulator-step to action-chunk granularity. The reward is assigned once at each chunk boundary, with zero reward assigned to the remaining simulator steps within that chunk. No further reward is emitted after the first detected success. We use the raw progress estimates without temporal differencing or clipping.

For each query, the current boundary frame is appended to a causal history maintained separately for each environment. Robometer receives $T = 8$ frames, matching its training-time input length, sampled at approximately uniform indices $\lfloor \text{linspace}(0, t, 8) \rceil$ from the first frame through the current boundary. This construction uses no future observations. Robometer is queried once per action chunk, corresponding to every 16 simulator steps for GR00T-N1.5 and every 10 steps for $\pi_{0.5}$.

On LIBERO-Long, we set $\beta_{\text{R}} = 0.07041$ for GR00T-N1.5 and $\beta_{\text{R}} = 0.04401$ for $\pi_{0.5}$. These coefficients are selected once before RL training using reference rollouts to match the intermediate-reward scale of StructRL. All other PPO hyperparameters, interaction budgets, terminal rewards, and evaluation settings are shared with StructRL. Applying the negative base once per chunk prevents unsuccessful policies from accumulating positive progress reward merely by extending a rollout, while stopping reward emission after success prevents repeated credit from a latched success state.

## G LIMITATIONS

In this work, we introduced STRUCTRL, a structured online RL framework that improves long-horizon VLA training by combining verifiable subtask supervision with structure-aware reward gating and dynamic reward pacing. While our experiments demonstrate strong empirical performance, several directions remain for further study.

**Subtask completion signals.** Our main experiments use simulator predicates to provide precise and reproducible detection of subtask completion. In real-world settings, these signals would need to be derived from available sensory observations or other task-specific feedback. Extending STRUCTRL to such settings is an important direction for future work and may involve integrating suitable progress-detection mechanisms.

**Real-world evaluation.** Our evaluation focuses on simulation, which enables controlled and reproducible comparisons across diverse long-horizon tasks and training configurations. Evaluating STRUCTRL on physical robotic systems is a natural next step to study its behavior under real-world perception, dynamics, and execution variability.