RLinf-VLA: A Unified and Efficient Framework for Reinforcement Learning of Vision-Language-Action Models

Anonymous CVPR submission

Paper ID 4

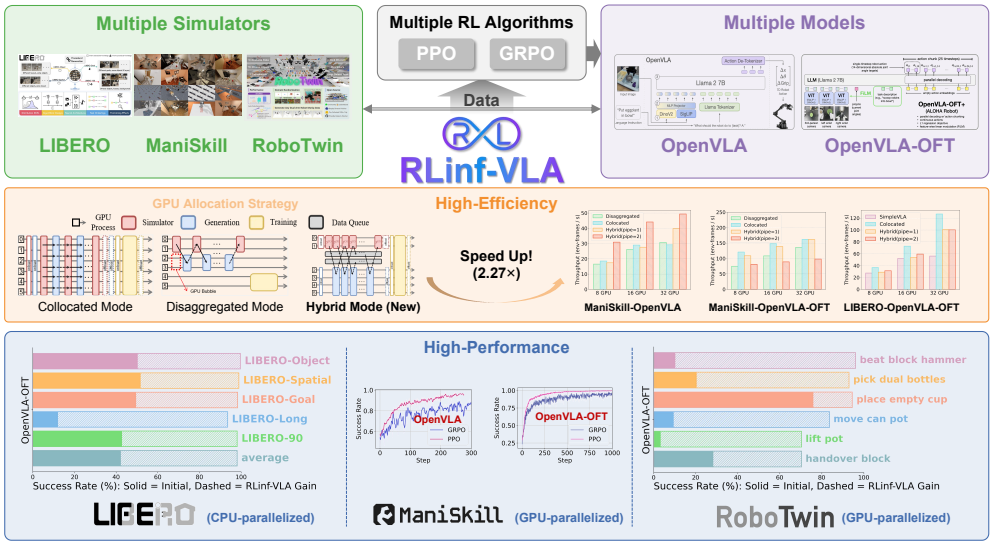


Figure 1. Overview of RLinf-VLA. Built on a unified interface, RLinf-VLA seamlessly supports diverse VLA architectures, multiple RL algorithms, and various simulators. It provides three GPU allocation modes: *collocated*, *disaggregated*, and a novel *hybrid* mode, speeding up training by 2.27× compared to the baseline. A single unified model achieves 98.11% success on 130 LIBERO tasks and 97.66% on 25 ManiSkill tasks. RLinf-VLA achieves an average 84.63% success on 6 RoboTwin tasks.

Abstract

Recent advances in vision-language-action (VLA) models have motivated the extension of their capabilities to embodied settings, where reinforcement learning (RL) offers a principled way to optimize task success through interaction. However, existing methods remain fragmented, lacking both a unified platform for fair comparison across architectures and algorithms and an efficient system design for scalable training. To address these challenges, we introduce RLinf-VLA, a unified and efficient framework for scalable RL training of VLA models. RLinf-VLA achieves unification by providing a unified interface that standardizes the integration of diverse VLA architectures, multiple RL algorithms, and heterogeneous simulators, enabling extensibility. To ensure efficiency, the system adopts a flexible resource allocation architecture for rendering, inference, and training workloads in RL pipelines. In particular, for GPU-parallelized simulators, RLinf-VLA introduces a hybrid fine-grained pipeline allocation strategy, yielding a 1.61x–1.88x training speedup. Using this unified system, models trained with RLinf-VLA demonstrate consistent performance improvements of approximately 20–85% across multiple simulation benchmarks, including LIBERO, ManiSkill, and RoboTwin. Furthermore, we distill a set of training practices for effective RL-based VLA training. We position RLinf-VLA as a foundational system to enable efficient, unified, and reproducible research in embodied intelligence.

1. Introduction

Vision-Language-Action (VLA) models represent a new generation of foundation models that aim to unify per-

ception, language understanding, and embodied control [9, 25]. By leveraging vision-language models pretrained on internet-scale data and further training them on large, heterogeneous robot demonstration datasets [16, 30], VLAs can map raw sensor observations and natural language instructions directly to robot actions. This paradigm has demonstrated strong generalization across a wide range of tasks [2, 15, 17, 23, 34, 40, 41].

However, deploying VLA models typically requires post-training to mitigate distribution mismatch between training data and deployment environments. Without effective post-training, performance can degrade significantly when models encounter novel states, instructions, or execution conditions.

Reinforcement learning (RL) has emerged as an increasingly important post-training paradigm, as it directly optimizes cumulative task rewards through trial-and-error interaction [37]. In contrast to supervised fine-tuning (SFT), another commonly used post-training approach, RL enables exploration beyond narrow expert demonstrations and equips policies with corrective and adaptive behaviors. Recent studies indicate that RL fine-tuning can yield stronger out-of-distribution generalization than SFT, particularly in terms of semantic alignment and execution robustness [22].

Despite its promise, applying RL to VLA models remains largely small-scale or fragmented [22, 24], making the development of new RL algorithms for VLAs expensive and cumbersome. Although SimpleVLA-RL [20] enables large-scale RL training for VLAs by building on VeRL [36], an RL codebase originally designed for large language models, it lacks system-level optimizations tailored to embodied settings, where simulators compete with model inference and learning for GPU resources. This limitation constrains scalability and substantially increases training time.

Moreover, online RL requires repeated and tightly coupled model–environment interactions. Without a system design explicitly tailored to this execution pattern, significant GPU idle time and pipeline bubbles can arise, leading to inefficient resource utilization. Finally, existing works often rely on disparate model architectures, RL algorithms, and evaluation protocols, making fair comparison difficult and hindering the extraction of general principles for RL-based VLA training.

To address these challenges, we present **RLinf-VLA**, a ***unified*** and ***efficient*** framework for scalable reinforcement learning of vision-language-action models. Our main contributions are summarized as follows:

- **Unified system abstraction.** RLinf-VLA provides a unified interface that supports multiple robotic simulators (ManiSkill [39], LIBERO [21], RoboTwin [8]), diverse VLA architectures (OpenVLA [17], OpenVLA-OFT [18]), and reinforcement learning algorithms (PPO [33], GRPO [35]). The system exposes three execution modes, including a novel hybrid GPU allocation mode, enabling scalable and configurable training across heterogeneous setups.
- **Efficient system and algorithm design.** RLinf-VLA introduces hybrid fine-grained pipelining for GPU-parallelized simulators and collocated execution for CPU-parallelized simulators, substantially improving training throughput, with speedups of up to 2.27×. In addition, we incorporate a set of algorithmic optimizations that further enhance training efficiency and stability.
- **Strong empirical performance and generalization.** Using a single unified model, RLinf-VLA achieves a 98.11% success rate on 130 LIBERO tasks and 97.66% on 25 ManiSkill tasks. RLinf-VLA also achieves an average success rate of 84.63% on six RoboTwin tasks, realizing an average performance improvement of 63.75% on these tasks, which demonstrates its powerful performance and generalization capabilities in post-training.
- **Open and extensible platform.** RLinf-VLA is released as an open-source and actively maintained platform, providing a practical foundation to accelerate, standardize, and scale reinforcement learning research for embodied intelligence.

2. Related Work

2.1. Reinforcement Learning for VLA

Vision-Language-Action models (VLAs) [1, 2, 4–6, 15, 17, 18, 43, 46] represent a multimodal foundation model architecture designed for robotics. RL is increasingly adopted to address the limitations of static imitation learning in VLAs. GRAPE [44] employs Direct Preference Optimization (DPO [31]) on partial trajectories to align models with human intent. Iterative approaches [12, 38] combine SFT with RL stages to balance stability and performance. VLA-RL [24] utilizes Proximal Policy Optimization (PPO [33]) to exhibit significantly stronger generalization to unseen objects and environments compared to their SFT counterparts [22]. SimpleVLA-RL [20] and related variants [38] apply Group Relative Policy Optimization (GRPO [35]) to improve training efficiency and performance without complex reward design. π_{RL} [7] implements the policy gradient algorithm to flow-matching model for the first time. $\pi^*_{0.6}$ [14] introduces RECAP to train a binarized value function and VLA. A unified and efficient infrastructure is essential for advancing VLA via RL, yet current implementations remain fragmented [22, 24]. Although frameworks such as SimpleVLA-RL [20] have emerged, they largely follow generic LLM RL post-training paradigms and fail to adequately account for the system-level optimizations and interface extensibility required by embodied intelligence tasks.

2.2. Simulators

Simulators [3, 10, 19, 26–29] offer scalable, safe, and controllable environments that bypass physical hardware constraints. They facilitate massive parallel data generation and automated resets, essential for training and evaluating generalizable VLA policies. Notable platforms include ManiSkill [39] for physics-rich manipulation, LIBERO [21] for instruction-conditioned reasoning, and RoboTwin [8] for bimanual tasks with domain randomization. Consequently, the construction and utilization of simulators are indispensable for RL. However, inconsistent interfaces across simulators often necessitate extensive code restructuring. A unified interface is therefore critical to enable seamless switching between diverse simulation benchmarks.

3. Preliminary

3.1. Reinforcement Learning

To guide the design of our efficient training infrastructure, we first formalize the problem of finetuning VLA models using Reinforcement Learning (RL). We model vision-based manipulation tasks as a partially observable Markov decision process (POMDP). A POMDP extends the standard MDP and is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \Omega, O)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space (discrete or continuous, potentially chunked), $P(s' \mid s, a)$ the state transition function, $r(s, a)$ the reward function, $\gamma \in [0, 1]$ the discount factor, Ω the observation space, and $O(o \mid s)$ the observation function mapping a state s to an observation o.

At each timestep t, the agent samples an action $a_t \sim \pi_\theta(\cdot \mid o_t)$ based on the current observation $o_t \sim O(\cdot \mid s_t)$. The environment then executes a_t, transitions to the next state s_{t+1} via P, and produces the next observation o_{t+1}. The goal of RL is to optimize the parameterized policy π_θ so as to maximize the expected cumulative discounted reward over trajectories $\tau = (s_0, a_0, \ldots, s_T)$:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]. \tag{1}$$

3.2. Pipeline of Reinforcement Learning for VLA models

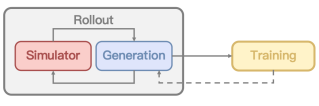


Figure 2. Pipeline of Reinforcement Learning for VLA models

The RL pipeline for VLA models comprises three distinct components: *Generation*, *Simulator*, and *Training*, as illustrated in Figure 2. GPU resources are primarily allocated between the rollout phase (*Generation* and *Simulator*) and the optimization phase (*Training*). A key distinction from LLM-based RL lies in the hierarchical action formulation. While LLMs typically operate on single tokens [35], VLA policies often predict a single action [17] or a *chunk* of actions to ensure smooth control [45]. In our framework, we unify these representations by treating a single action as a chunk of size one. Crucially, this introduces a three-level hierarchy: *Chunk* $\rightarrow$ *Atomic Action* $\rightarrow$ *Token*. Each atomic action comprises multiple tokens, where each token typically corresponds to a specific dimension of the robot's action space (e.g., end-effector pose or joint angles). Moreover, the integration of simulators imposes substantial computational overhead on the generation process, demanding significant CPU and GPU resources for physics simulation and rendering. The interaction typically proceeds as follows [18]: the *Generation* module infers an action chunk based on the current observation; the *Simulator* executes this chunk and returns the observation from the final timestep. This loop continues until trajectory collection is complete, after which the *Training* module updates the policy using the gathered data.

4. Method

Section 4.1 introduces flexible GPU Allocation Strategies for improved resource scheduling across different simulator types. Section 4.2 presents a Unified Interface that seamlessly integrates diverse simulators, VLA models, and RL algorithms. Building on this system design, Section 4.3 distills key Design Choices for scalable RL training of VLA models.

4.1. GPU Allocation Strategies

The RL training pipeline (described in Section 3.2) for VLA involves varying resource demands from *Training*, *Generation*, and *Simulator*. Resource bottlenecks depend heavily on the simulator type: CPU-parallelized simulators are typically CPU-bound, using GPUs primarily for rendering and inference, whereas GPU-parallelized simulators execute simulation, rendering, and inference entirely on the GPU. While the latter offers higher throughput, it creates severe contention for GPU memory and compute. To maximize efficient utilization across these diverse setups, flexible and optimized GPU allocation strategies are essential. Our framework supports flexible and easily configurable allocation modes: *collocated*, *disaggregated*, and a novel *hybrid* mode.

4.1.1. Collocated Allocation

In this mode (Figure 3), all components co-exist on the same set of GPUs. The original version of collocated mode aimed to maximize GPU utilization by maintaining only one com-

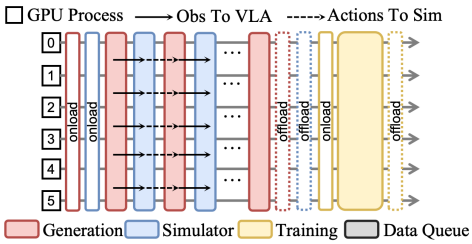


Figure 3. Collocated Mode.

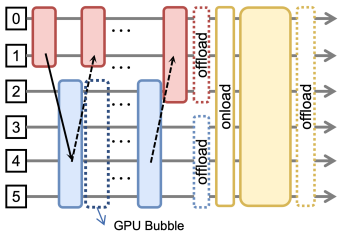


Figure 5. Hybrid Mode (see Figure 3 for legend).

ponent on all GPUs at any given time. When a component finished its computation, it would be offloaded from GPU to CPU memory, and then onloaded again when needed for its next task. However, under the embodiment setting, both *Simulator* and *Generation* require GPU resources and interact frequently in an iterative manner. The overhead introduced by repeated offload-onload operations at each interaction becomes prohibitively large and unacceptable.

Therefore, we implement a modified collocated strategy where offload and onload operations for the *Simulator* and *Generation* occur only at the beginning and end of the rollout phase, as shown in the figure. While this approach successfully avoids the frequent offload-onload overhead during rollout, it cannot fully utilize GPU resources and remains sub-optimal. Since *Generation* and *Simulator* must interact iteratively, they spend significant time waiting for each other, occupying GPU memory without performing computation, leading to resource wastage and limited scalability.

4.1.2. Disaggregated Allocation

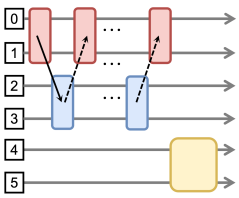


Figure 4. Disaggregated Mode (see Figure 3 for legend).

In this mode, each component is assigned to a distinct (possibly multi-GPU) partition, with no overlap across components. Figure 4 depicts the central idea. This ensures that every component can fully exploit its allocated resources. This mode is simple to implement, but it lead to GPU underutilization due to inter-component dependencies. For example, in Figure 4, the gpu 4, 5 are completely idle during the rollout phase util training.

4.1.3. Hybrid Allocation with Fine-grained Pipelining

To overcome the drawbacks of the above modes, we propose a hybrid strategy: *hybrid allocation* combined with *fine-grained pipelining*. In hybrid allocation, components can flexibly select GPUs. A typical configuration is to assign *Generation* and *Simulator* to different GPU partitions, while allowing *Training* to utilize all GPUs. Figure 5 shows an illustration. However, the resources remain underutilized. For instance, the *Simulator* must wait for actions generated by the *Generation*, leaving some GPUs idle and creating "GPU bubbles".

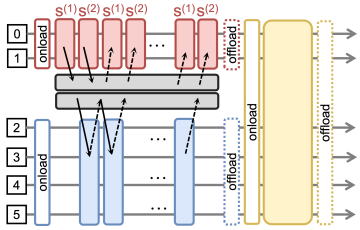


Figure 6. Hybrid Mode with Fine-grained Pipelining (see Figure 3 for legend).

On top of this, we introduce fine-grained pipelining to mitigate bubbles caused by inter-component dependencies. Specifically, a simulator instance on one GPU is partitioned into multiple sub-simulators, denoted as $S^{(1)}, S^{(2)}, \ldots, S^{(k)}$. The pipeline proceeds as follows, where superscripts denote simulator indices and subscripts indicate timesteps:

1. At step $t = 0$, $S^{(1)}$ generates the initial observation $o_0^{(1)}$, which is sent to the *Generation* component to produce action $a_0^{(1)}$.
2. Meanwhile, $S^{(2)}$ generates $o_0^{(2)}$ in parallel.
3. Once $a_0^{(1)}$ is ready, it is fed back into $S^{(1)}$ to produce the next observation $o_1^{(1)}$, while $o_0^{(2)}$ is simultaneously processed by the actor to generate $a_0^{(2)}$.

This scheduling allows *Simulator* and *Generation* to run concurrently, reducing idle time while preserving correctness. In this way, our framework avoids the frequent offloading overhead of collocated allocation while also eliminating the GPU bubbles that occur in disaggregated allocation. Figure 5 provides a schematic illustration of the hybrid allocation mode with fine-grained pipelining ($k = 2$).

4.2. A Unified Interface

To conveniently reuse existing simulators, models, and algorithms, RLinffff-VLA provides a unified interface designed for high extensibility and scalability.

4.2.1. Multiple Simulators Support.

RLinff-VLA supports multiple robotic simulators, including ManiSkill [39], LIBERO [21], and RoboTwin [8]. To facilitate this, we provide a unified interface that standardizes environment interactions across different backends, such as ManiSkill and RoboTwin's GPU-parallelized environments and LIBERO's CPU-parallelized wrappers. This interface includes standard Gym-style APIs with built-in support for action chunking and flexible episode termination. Furthermore, we define a unified interface for observation and action to ensure seamless compatibility with various algorithms and models.

4.2.2. Multiple Models Support

Benefiting from a unified data processing pipeline across simulators, our framework facilitates the seamless integration of existing models by requiring only a standardized interaction interface. We support diverse VLA architectures, exemplified in this study by OpenVLA [17] and OpenVLA-OFT [18], which represent configurations with single-step and multi-step action chunking, respectively. To enable efficient fine-tuning, we integrate Low-Rank Adaptation (LoRA) [13].

4.2.3. Multiple Algorithms Support

Our framework provides support for multiple reinforcement learning algorithms, with an initial focus on Proximal Policy Optimization (PPO) [33] and Group Relative Policy Optimization (GRPO) [11, 35]. The core functions for on-policy RL algorithms are the advantage function and the loss function. In our framework, we can configure the algorithms by specifying these two functions only, without redundant code.

4.3. Design Choices for Algorithms

RL algorithms for large models involve more complex design choices than those for smaller models. For VLAs, we adopt methodologies from both LLMs and robotics. In Section 4.3.1, we introduce general features applicable to all algorithms. Subsequently, in Section 4.3.2 and Section 4.3.3, we detail specific design choices for PPO and GRPO, respectively. Consistent with our codebase style, all these features can be easily configured by changing values in the configuration file.

4.3.1. Multi-Granularity Calculation Support

Our framework supports advantage estimation and log-probability computation at multiple levels of granularity, enabling seamless adaptation to different algorithmic requirements.

For advantage estimation, we provide two formulations for assigning advantages to action chunks $c_t = (a_{t,1}, \ldots, a_{t,C})$[1]. In the chunk-level formulation, the entire chunk is treated as a single macro-action and is assigned a summed reward and a shared advantage. In contrast, the action-level formulation assigns individual rewards and advantages to each atomic action $a_{t,i}$ within the chunk.

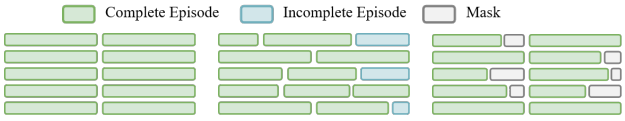
Similarly, for log-probability computation, we support three corresponding granularities:

$$\text{Chunk-level: } \pi(c_t|o_t) = \prod_{i=1}^{C} \pi(a_{t,i}|o_t, a_{t,:i-1}),$$

$$\text{Action-level: } \pi(a_{t,i}|o_t, a_{t,:i-1}) = \prod_{j=1}^{M} \pi(d_{t,i,j}|o_t, d_{t,i,:j-1}),$$

$$\text{Token-level: } \pi(d_{t,i,j}|o_t, d_{t,i,:j-1}),$$

where $d_{t,i,j}$ denotes the j-th token of the i-th atomic action.



(a) Fixed Length (b) Partial Reset (c) Valid Action Mask

Figure 7. Illustration of the three rollout modes.

4.3.2. Design Choices for PPO

Scaling PPO to VLA models introduces several design challenges. We summarize the key design decisions adopted in our framework below.

Partial Reset Support During rollouts, a sub-environment typically terminates either upon reaching the maximum episode length or upon successfully completing a task. We consider two standard strategies for handling such terminations. In the *Fixed Episode Length* mode, all sub-environments are reset simultaneously only after reaching the maximum episode length. In contrast, the *Partial Reset* mode resets each sub-environment immediately upon termination, independent of others. Figure 7a and Figure 7b illustrate the differences between these two modes.

Critic Design Adapting PPO to VLA models places stringent requirements on critic design. Maintaining a separate critic network is computationally prohibitive and complicates GPU resource management due to the large scale of VLA models. We therefore adopt a parameter-sharing strategy between the actor and critic. Following RL4VLA [22], we attach a lightweight value head to the language model component of the VLA for efficient state value estimation.

Value for Action Chunks As discussed in Section 4.3.1, the two advantage estimation formulations naturally induce two corresponding value estimation strategies for an action chunk $c_t = (a_{t,1}, a_{t,2}, \ldots, a_{t,C})$. In the *chunk-level*

[1]Here, t denotes the index of the chunk, and C denotes the chunk size.

formulation, the critic estimates a single scalar value for the entire chunk, treating it as a macro-action, i.e., $V : \mathcal{S} \to \mathbb{R}$. In contrast, the *action-level* formulation produces a C-dimensional value vector, providing an individual value estimate for each atomic action $a_{t,i}$ in the chunk, i.e., $V : \mathcal{S} \to \mathbb{R}^C$. Empirically, we find that the action-level formulation consistently leads to better performance.

4.3.3. Design Choices for GRPO

Transferring GRPO to embodied tasks introduces several non-trivial challenges. To address these, we adopt the following design choices.

Valid Action Mask. While rollouts are executed for a fixed episode duration, tasks often terminate early. This presents two potential strategies for policy optimization: (1) using the full trajectory regardless of task completion timing, or (2) considering only timesteps prior to task completion. Our framework supports both strategies, referring to the latter as the *Valid Action Mask* setting (Figure 7c). Empirically, we find that masking out post-completion steps generally improves policy performance in the GRPO setting.

Loss Normalization by Trajectory Length. To ensure that trajectories of different lengths contribute comparably to optimization, we normalize the policy loss by the number of valid timesteps in the *Valid Action Mask* setting. Specifically, for a trajectory τ_i with T_i^{succ} valid timesteps, the contribution of each timestep to the objective is scaled by $1/T_i^{\text{succ}}$. This prevents longer trajectories from dominating the gradient and promotes balanced learning across successful and partially completed trajectories.

Success Rate Filter. Inspired by the dynamic sampling strategy in DAPO [42], we introduce a success-rate filter for GRPO. This filter discards trajectory groups where all trajectories either succeed or fail, as computing non-zero advantages requires a mix of successful and failed outcomes. In practice, this mechanism accelerates convergence and consistently improves policy performance.

5. Experiment Results

In this section, we systematically investigate three key questions concerning the effectiveness of the proposed RLinf-VLA framework:

(1) Is RLinf-VLA *high-performance*? We evaluate RLinf-VLA on three representative testbeds, LIBERO, ManiSkill and RoboTwin. The results demonstrate that RLinf-VLA achieves approximately 20–85% improvement, highlighting its strong capability to support large-scale multi-task learning.

(2) Is RLinf-VLA *high-efficiency*? We benchmark the framework across both GPU-parallelized and CPU-parallelized simulators, and observe that the optimal configuration improves training throughput, with speedups of up to $1.88\times$. This finding underscores the necessity of supporting diverse allocation modes. Notably, RLinf-VLA achieves up to $2.27\times$ speedup over existing frameworks.

(3) What are the *actionable practices* for applying PPO and GRPO to VLA training? Through extensive ablation studies, we identify the key factors that govern training performance, offering practical guidelines for effectively deploying RL in VLA settings.

Table 1. Evaluation results across three simulation benchmarks. Values denote success rates (%).

(a) Evaluation on ManiSkill.

Method	In-Distribution	OOD Avg.
OpenVLA (Base)	53.91	39.10
OpenVLA (RLinf-GRPO)	84.38	75.15
OpenVLA (RLinf-PPO)	96.09	**81.93**
OpenVLA-OFT (Base)	28.13	18.29
OpenVLA-OFT (RLinf-GRPO)	94.14	60.64
OpenVLA-OFT (RLinf-PPO)	**97.66**	77.05

(b) Evaluation on LIBERO.

OpenVLA-OFT	Object	Spatial	Goal	Long	90	Avg.
Base	50.20	51.61	49.40	11.90	42.67	42.09
RLinf-GRPO	99.67	98.93	98.32	93.55	98.12	98.11

(c) Evaluation on RoboTwin.

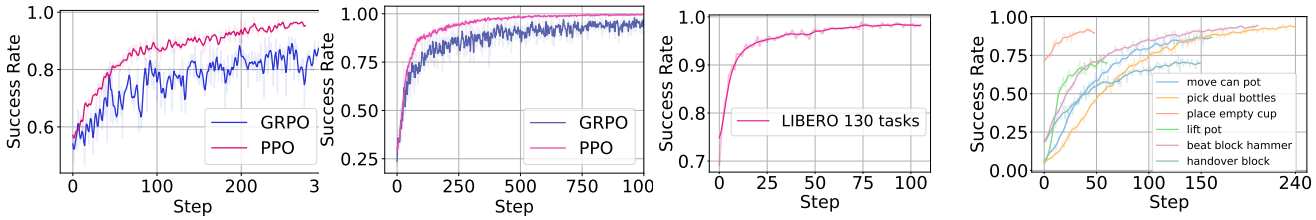
OpenVLA-OFT	Cup	Hammer	Bottles	Can	Pot	Hand	Avg.
Base	75.78	10.15	20.31	9.37	3.13	28.13	24.48
RLinf-GRPO	94.53	96.09	92.96	83.59	70.31	70.31	84.63

5.1. Policy Performance

In this section, we evaluate the effectiveness of RLinf-VLA policies across multiple benchmarks, focusing on task success rates and generalization performance.

Experimemt Setup. We evaluate RLinf-VLA on three benchmarks: ManiSkill, LIBERO, and RoboTwin, with full settings in Appendix III.A.

For ManiSkill, we train on 25 pick-and-place tasks with object and receptacle variations. We follow the out-of-distribution evaluation protocol of RL4VLA [22], measuring generalization in *vision*, *language*, and *action*. Each sub-setting is evaluated with 256 random episodes. For LIBERO, we use the public task groups [21], including LIBERO-Spatial, Object, Goal, Long, and 90, for training and evaluation. Instead of training within a single group, we train one unified model on the combined 130 tasks over 5 groups, testing large-scale multitask learning. Performance is reported across groups, averaged over 50 episodes per task with three random seeds. For RoboTwin [8], we select six tasks: "beat block hammer", "move can pot", "place empty cup", "pick dual bottles", "lift pot", and "handover



(a) OpenVLA on ManiSkill. (b) OpenVLA-OFT on ManiSkill. (c) OpenVLA-OFT on LIBERO. (d) OpenVLA-OFT on RoboTwin.

Figure 8. Training curves on different benchmarks. The x-axis shows the number of training epochs, and the y-axis indicates the corresponding success rate. Light-colored lines represent the raw data, while the dark-colored curves are smoothed using a Gaussian filter with $\sigma = 1$.

block". Training uses 1,000 fixed randomized scene seeds per task, and evaluation samples 128 unseen seeds to assess out-of-distribution generalization.

Training Results. Figure 8 presents the training curves for each setup. Figure 8a and Figure 8b report the training performance on ManiSkill, utilizing OpenVLA and OpenVLA-OFT models trained via PPO and GRPO. Across all settings, RL delivers substantial performance gains, improving success rates by 45%–70% over the baseline. Notably, PPO consistently outperforms GRPO and exhibits greater stability for both OpenVLA and OpenVLA-OFT in the ManiSkill setting.

Figure 8c illustrates the training curve of OpenVLA-OFT with the GRPO algorithm on 130 tasks of LIBERO. Results demonstrate that the success rate improves substantially from approximately 73% to 98%, yielding an overall performance gain of about 30%. These findings underscore the effectiveness of the GRPO design in RLinf-VLA for enhancing multi-task training.

Figure 8d depicts our training curves for OpenVLA-OFT on RoboTwin. The training process remains stable with minimal fluctuations. Moreover, the model converges to high performance even on tasks with extremely low initial success rates (as low as 3%).

Evaluation Results. Table 1 summarizes RLinf-VLA results across three benchmarks, with ManiSkill detailed in Table 1a. "OpenVLA (Base)" and "OpenVLA-OFT (Base)" denote the RL base models for OpenVLA and OpenVLA-OFT.

Direct OOD comparison between OpenVLA and OpenVLA-OFT is not strictly fair because their base performance differs. OpenVLA improves from 39.10% to 81.93% after RL, while OpenVLA-OFT rises from 18.29% to 77.05%. Although starting weaker, OpenVLA-OFT shows slightly larger relative gains, indicating that base model choice strongly affects final generalization.

Table 1b reports OpenVLA-OFT trained on 130 tasks with RLinf-VLA over five LIBERO groups. LIBERO-Object reaches 93%–99% success, with an overall average of 98.11% and a mean improvement of 56.02%. Notably, we use a single unified model across all tasks, unlike standard RL-for-VLA methods that train separate models per group, demonstrating RLinf-VLA's support for large-scale multi-task RL.

OOD results on RoboTwin are shown in Table 1c. RLinf-VLA achieves an average success rate of 84.63%, over 60% improvement, showing strong generalization on complex tasks such as long-horizon bimanual manipulation.

5.2. System Efficiency

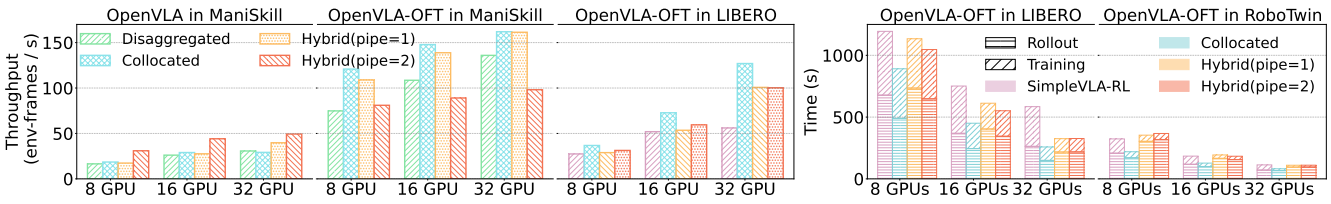
In this section, we evaluate the efficiency of our system by comparing different GPU allocation strategies. Our results show that the optimal strategy varies across simulators and models, highlighting the importance of flexible GPU allocation, which is a core feature of RLinf-VLA.

Experiment Setup. We summarize key settings here, with full details in Appendix III.B. We evaluate representative tasks from three benchmarks. For ManiSkill, we use "PutCarrotOnPlateInScene-v2" with 256 parallel environments. For LIBERO, we adopt the LIBERO-Long suite, and for RoboTwin, the "place empty cup" task. For LIBERO and RoboTwin, parallel environments scale linearly with nodes (64, 128, and 256 for 1, 2, and 4 nodes).

We consider three GPU allocation strategies (Section 4.1). *Collocated* shares GPUs across components, *Disaggregated* separates training from rollout, and *Hybrid* pipelines simulation by splitting instances into stages. For comparisons, ManiSkill lacks multi-GPU baselines, so we compare *Collocated* and *Hybrid* (pipelined) modes against a naive *Disaggregated* baseline. For LIBERO and RoboTwin, we benchmark against SimpleVLA-RL [20] (collocated only) and evaluate our *Collocated* and *Hybrid* modes; the disaggregated mode is omitted since hybrid consistently performs better when GPUs are underutilized.

Results. Figure 9 reports the end-to-end throughput of RLinf-VLA and baselines for VLA+RL training under different cluster sizes and placement strategies.

From Figure 9a, for OpenVLA in ManiSkill, RLinf-VLA achieves up to a 1.88× speedup over the disaggregated baseline using the Hybrid (`pipe=2`) mode on 8 GPUs. Although scaling to more GPUs introduces overheads from model loading, offloading, and state switch-



(a) Throughput for different settings. (b) Latency breakdown.

Figure 9. Evaluation of system efficiency on ManiSkill, LIBERO, and RoboTwin, comparing OpenVLA and OpenVLA-OFT. Throughput (total environment frames per second) improves with increasing pipeline stages ("`pipe`").

ing, fine-grained pipelining reduces GPU idle time and maintains a 1.61×–1.69× advantage over the disaggregated mode. Deeper pipelines (`pipe=2` vs. `pipe=1`) further improve performance by reducing rollout-stage bubbles.

For OpenVLA in ManiSkill and OpenVLA-OFT in ManiSkill, as GPU count increases, the hybrid (`pipe=1`) mode outperforms the collocated mode. Since ManiSkill is GPU-parallelized, rollout throughput scales with parallel environments, making simulator resource allocation critical. To improve utilization, we enable offloading in the collocated mode (Section 4.1.1), but communication overhead grows with cluster size. In contrast, hybrid mode splits GPUs evenly between components, reducing interference.

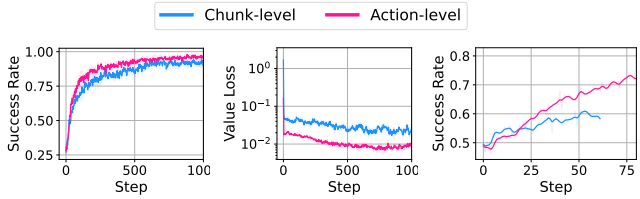
For OpenVLA-OFT in LIBERO and ManiSkill, the behavior differs: collocated and hybrid (`pipe=1`) modes outperform disaggregated and hybrid (`pipe=2`). First, OpenVLA-OFT generates action chunks while the simulator executes them sequentially, shifting the generation-to-execution ratio from 1:1 to about 1:15. Second, LIBERO's CPU-parallelized simulator becomes the main bottleneck. This imbalance diminishes the benefit of pipelining, effectively reverting execution to a near-sequential process.

From Figure 9b, where "Rollout" denotes the total time for multi-step interactions between the generator and simulator, for OpenVLA-OFT in LIBERO and RoboTwin, RLinf-VLA still achieves a 1.34×–2.27× speedup over SimpleVLA-RL under collocated settings. The gains come from both rollout and training. During rollout, RLinf-VLA uses vectorized environments that are more efficient than SimpleVLA-RL's multi-process workers. During training, system-level optimizations such as adaptive communication and the removal of redundant log-probability recomputation further reduce latency, with benefits increasing at scale.

Summary. Different simulators demand different allocation strategies. RLinf-VLA's flexible multi-mode support enables consistently high efficiency across diverse execution regimes.

5.3. Ablation Study

In this section, we present ablation studies under different setups and summarize key insights for PPO and GRPO training.

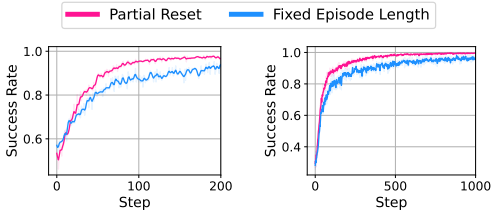


(a) succ. in ManiSkill (b) loss in ManiSkill (c) succ. in LIBERO

Figure 10. For OpenVLA-OFT, action-level value estimation consistently outperforms chunk-level estimation across different tasks.

(1) Tips for PPO.

(a) Action-level value estimation outperforms chunk-level estimation for PPO with action chunks. Figure 10 shows the PPO training curves comparing action-level and chunk-level value estimation as described in Section 4.3.2 on the ManiSkill "PutOnPlateInScene25Main" task using the OpenVLA-OFT model. Action-level estimation consistently yields higher success rates and lower value loss, demonstrating more effective learning and faster policy improvement. The performance divergence between different levels of value estimation remains consistent across diverse tasks, as corroborated by similar results in the LIBERO-Goal benchmark (Figure 10c).



(a) succ. in OpenVLA (b) succ. in OpenVLA-OFT

Figure 11. Partial reset consistently outperforms Fixed Episode Length mode in PPO.

(b) Partial reset substantially improves sample efficiency. Figure 11 presents the PPO training curves for the *Partial Reset* and *Fixed Episode Length* rollout modes discussed in Section 4.3.2. Since the optimization objective is to achieve success at least once per rollout ("success_once"), *Partial Reset* leads to a significantly higher success rate. For a given

number of training steps, the success rate under *Partial Reset* consistently exceeds that of the *Fixed Episode Length* mode. This trend is observed regardless of the model type (OpenVLA or OpenVLA-OFT).

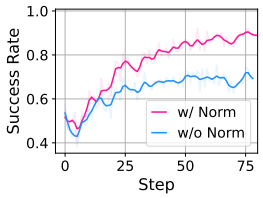
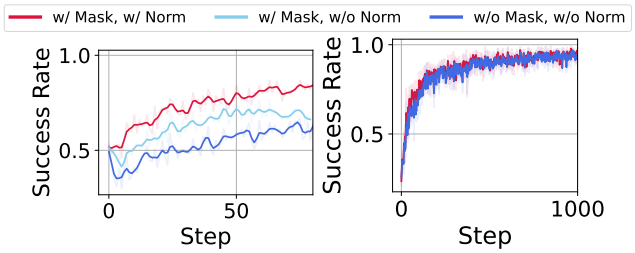


Figure 12. Ablation study on trajectory length normalization. Success rate in LIBERO-Goal with OpenVLA-OFT.

(2) Tips for GRPO.

(a) Trajectory length normalization in GRPO. As discussed in Section 4.3.3, normalizing the loss by trajectory length aims to reduce bias when episodes vary in length. Figure 12 shows that incorporating trajectory length normalization ("w/ Norm") can lead to substantially higher performance compared with the unnormalized setting ("w/o Norm").

(b) Valid action mask in GRPO. We also study the effect of valid action masking, introduced in Section 4.3.3. Since the optimization objective is defined with respect to "success_once", excluding actions after reaching the success state improves sample efficiency and avoids redundant updates. Moreover, applying the valid action mask naturally results in shorter trajectories, which further benefits trajectory length normalization. Figure 13a shows results on LIBERO-Goal. Comparing the "w/o Mask, w/o Norm" and "w/ Mask, w/o Norm" curves, the setting with a valid action mask achieves consistently better performance. The "w/ Mask, w/ Norm" curve demonstrates that combining the mask with trajectory length normalization provides an additional improvement. However, the effect of these techniques is task-dependent. In the ManiSkill setting, we do not observe clear benefits from either valid action masking or trajectory length normalization.



(a) succ. in LIBERO-Goal (b) succ. in ManiSkill

Figure 13. Ablation studies on valid action mask in GRPO with OpenVLA-OFT.

(c) Success rate filtering can improve training stability in some settings of GRPO. As discussed in Section 4.3.3, we implement a success rate filter for GRPO that discards groups in which all trajectories have identical cumulative rewards. This mechanism can improve the stability of GRPO training. For instance, in the OpenVLA ManiSkill setting, training without the filter ("w/o Filter") exhibits a clear collapse around step 400, whereas enabling the filter ("w/ Filter") largely alleviates this issue. However, the benefit of the filter is not universal: in the OpenVLA-OFT ManiSkill setting and the OpenVLA-OFT LIBERO-Goal setting, its effectiveness is much less pronounced.

6. Conclusion

In this work, we introduced RLinf-VLA, a unified and efficient framework for reinforcement learning-based training of Vision-Language-Action models. RLinf-VLA integrates multiple simulators, algorithms, and VLA architectures, while providing flexible execution modes and system-level optimizations that significantly improve training efficiency. Extensive experiments demonstrate that models trained with RLinf-VLA achieve approximately 20–85% improvement on a wide range of simulated tasks. Moreover, our study distills actionable practices for both PPO and GRPO, guiding future research in RL-based VLA training. By open-sourcing RLinf-VLA with ongoing maintenance, we provide the community with a foundation to accelerate, standardize, and scale research in embodied intelligence.

References

[1] Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model. *arXiv preprint arXiv:2512.13030*, 2025. 2

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael 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, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. 2

[3] Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xu Huang, Shu Jiang, et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025. 3

[4] Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions. *arXiv preprint arXiv:2505.06111*, 2025. 2

[5] Jun Cen, Siteng Huang, Yuqian Yuan, Kehan Li, Hangjie Yuan, Chaohui Yu, Yuming Jiang, Jiayan Guo, Xin Li, Hao

Luo, Fan Wang, Fan Wang, and Deli Zhao. Rynnvla-002: A unified vision-language-action and world model. *arXiv preprint arXiv:2511.17502*, 2025.

[6] Chilam Cheang, Sijin Chen, Zhongren Cui, Yingdong Hu, Liqun Huang, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Xiao Ma, et al. Gr-3 technical report. *arXiv preprint arXiv:2507.15493*, 2025. 2

[7] Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, et al. π_{rl}: Online rl fine-tuning for flow-based vision-language-action models. *arXiv preprint arXiv:2510.25889*, 2025. 2

[8] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation, 2025. 2, 3, 5, 6, 15

[9] Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research (IJRR)*, 44(5):701–739, 2025. 2

[10] Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, et al. Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025. 3

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 5

[12] Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. Improving vision-language-action model with online reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 2

[13] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. 5, 15, 18

[14] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025. 2

[15] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. 2

[16] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024. 2

[17] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *Conference on Robot Learning (CoRL)*, 2024. 2, 3, 5, 15

[18] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025. 2, 3, 5, 15

[19] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Jose Martinez, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1, 000 everyday activities and realistic simulation. *CoRR*, abs/2403.09227, 2024. 3

[20] Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. SimpleVLA-RL: Scaling VLA training via reinforcement learning. 2026. 2, 7, 15, 16

[21] 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. 2, 3, 5, 6, 15

[22] 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 *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 2, 5, 6, 15

[23] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: A diffusion foundation model for bimanual manipulation. *International Conference on Learning Representations (ICLR)*, 2025. 2

[24] 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. 2

[25] Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision-language-action models for embodied ai. *arXiv preprint arXiv:2405.14093*, 2024. 2

[26] Reginald McLean, Evangelos Chatzaroulas, Luc McCutcheon, Frank Röder, Tianhe Yu, Zhanpeng He, K.R. Zentner, Ryan Julian, J K Terry, Isaac Woungang, Nariman Farsad, and Pablo Samuel Castro. *Meta-World+: An Improved, Standardized, RL Benchmark*. Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, 2025. 3

[27] 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 (RA-L)*, 7(3):7327–7334, 2022.

[28] Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

[29] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. *RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots*. Robotics: Science and Systems (RSS), 2024. 3

[30] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903, 2024. 2

[31] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2

[32] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2, 5, 12

[34] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. *Conference on Robot Learning (CoRL)*, pages 492–504, 2022. 2

[35] 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. 2, 3, 5, 13

[36] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, page 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. 2

[37] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. 2

[38] Shuhan Tan, Kairan Dou, Yue Zhao, and Philipp Krähenbühl. Interactive post-training for vision-language-action models. *arXiv preprint arXiv:2505.17016*, 2025. 2

[39] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. *ManiSkill3: GPU Parallelized Robotics Simulation and Rendering for Generalizable Embodied AI*. Robotics: Science and Systems, 2025. 2, 3, 5, 13, 15

[40] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. *Octo: An Open-Source Generalist Robot Policy*. Robotics: Science and Systems, 2024. 2

[41] Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*, 2025. 2

[42] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 6

[43] Zhecheng Yuan, Tianming Wei, Shuiqi Cheng, Gu Zhang, Yuanpei Chen, and Huazhe Xu. Learning to manipulate anywhere: A visual generalizable framework for reinforcement learning. *arXiv preprint arXiv:2407.15815*, 2024. 2

[44] 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. *arXiv preprint arXiv:2411.19309*, 2024. 2

[45] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems (RSS)*, pages 1–16. Robotics: Science and Systems, 2023. 3

[46] Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025. 2

Appendix for RLinf-VLA

7. Theoretical Background

7.1. Definitions of RL

Definition 1 (Return)**.** The return R_t is the γ-discounted cumulative reward starting from timestep t:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}). \tag{2}$$

Definition 2 (Value Function)**.** The value function $V_\pi(s)$ is the expected return when starting from state s and following policy π:

$$V_\pi(s) = \mathbb{E}_\pi[R_t \mid s_t = s], \tag{3}$$

where t denotes an arbitrary timestep at which the agent is in state s.

Definition 3 (Action-Value Function)**.** The action-value function $Q_\pi(s, a)$ is the expected return when executing action a in state s and thereafter following policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a], \tag{4}$$

where t denotes an arbitrary timestep at which the agent is in state s and takes action a.

Definition 4 (Advantage Function)**.** The advantage function $A_\pi(s, a)$ quantifies how much better taking action a in state s is compared to the expected value of the state:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \tag{5}$$

7.2. Introduction of PPO

PPO [33] is one of the most widely adopted reinforcement learning algorithms in robotics. PPO enhances training stability by constraining policy updates within a trust region, thereby preventing overly large changes that could destabilize learning. This is achieved through a clipped surrogate objective, which balances exploration and exploitation while maintaining sample efficiency. Due to its robustness and simplicity, PPO has become a standard baseline in robotics RL and serves as the foundation of our framework.

In PPO, the advantage function is commonly estimated using Generalized Advantage Estimation (GAE) [32]:

$$\hat{A}_t = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \big(r_{t+k} + \gamma V(s_{t+k+1}) - V(s_{t+k})\big), \tag{6}$$

where r_t denotes the reward at timestep t, $V(s)$ is the value function, γ is the discount factor, λ controls the bias–variance trade-off, and T is the episode horizon.

The PPO optimization objective is defined as:

$$J^{\text{PPO}}(\theta) = \mathbb{E}_t\Big[\min\big(\rho_t(\theta)\hat{A}_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big], \tag{7}$$

where

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid o_t)}{\pi_{\theta_{\text{old}}}(a_t \mid o_t)}, \tag{8}$$

$\pi_{\theta_{\text{old}}}$ denotes the rollout policy, and ϵ is the clipping parameter.[2]

[2]The definition of action a_t differs across LLM and robotics RL settings. Here we treat it as a generalized action. The same applies to GRPO.

7.3. Introduction of GRPO

GRPO [35] is a recent variant of policy optimization designed to simplify reinforcement learning pipelines. Unlike PPO, which requires training both a policy model and a value model, GRPO eliminates the need for an explicit value function by leveraging group-based relative comparisons of trajectories. This design reduces the overall model complexity and avoids potential inaccuracies from value estimation. As a result, GRPO provides a lightweight yet effective alternative to PPO, making it especially attractive for large-scale VLA training where efficiency and simplicity are important. In GRPO, the important ratio can be defined as:

$$\rho_t^{(i)}(\theta) = \frac{\pi_\theta(a_t^{(i)} \mid o_t^{(i)})}{\pi_{\theta_{\text{old}}}(a_t^{(i)} \mid o_t^{(i)})}, \tag{9}$$

where i denotes i-th trajectory. But the advantage can be defined as:

$$\hat{A}^{(i)} = \hat{A}_t^{(i)} = \frac{\mathcal{R}^{(i)} - \text{mean}(\{\mathcal{R}^{(j)}\}_{j=1}^{G})}{\text{std}(\{\mathcal{R}^{(j)}\}_{j=1}^{G})}, \quad \forall t \in \{0, 1, \ldots, |\tau^{(i)}|\}, \tag{10}$$

where $\mathcal{R}^{(i)}$ denotes the total reward of trajectory $\tau^{(i)}$. Note that $\mathcal{R}$ differs from the discounted return R, and G represents the group size. Given an initial observation o_0 from sample buffer D, the behavior model $\pi_{\theta_{\text{old}}}$ generates G trajectories $\{\tau_i\}_{i=1}^{G}$. The GRPO optimization objective is:

$$J^{\text{GRPO}}(\theta) = \mathbb{E}_{o_0 \sim D,\, \{\tau^{(i)}\} \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\tau^{(i)}|} \sum_{t=1}^{|\tau^{(i)}|} \min\Big(\rho_t^{(i)}(\theta)\, \hat{A}^{(i)},\ \text{clip}\big(\rho_t^{(i)}(\theta),\, 1-\epsilon,\, 1+\epsilon\big)\, \hat{A}^{(i)} \Big) \right], \tag{11}$$

where $|\tau^{(i)}|$ is the length of trajectory $\tau^{(i)}$, ϵ is the clip parameter.

8. Implementation Details of RLinf-VLA

8.1. The Unified Interface

The unified interface consists of two categories: *core functions* and *utility functions*.

Core Functions We implement standard Gym-style APIs, including "`reset`" and "`step`". The "`step`" function additionally supports an "`auto_reset`" option: when enabled, any sub-environment that terminates or is truncated will be automatically reset, thereby improving sample efficiency by avoiding idle environments. Following the ManiSkill [39] convention, we also support the "`ignore_terminations`" option. When enabled, termination signals are ignored and only truncation signals are respected, meaning that an episode ends only when the maximum episode length is reached. This feature allows us to flexibly support different implementation variants, such as "Partial Reset" and "Valid Action Mask".

Beyond these, we extend the interface with a "`chunk_step`" function to handle action chunks. Instead of naively looping over the chunk with repeated "`step`" calls, this function manages episode termination more carefully. Two modes are supported: (1) reset immediately when a sub-environment finishes within the chunk, or (2) defer reset until the entire chunk has been executed. This flexibility ensures the correct handling of episode boundaries when chunked actions are used.

Utility Functions Utility functions offer convenient support for training and evaluation. For instance, visualization utilities enable effortless generation of videos during rollouts or evaluation. In addition, we provide specialized utilities required by specific algorithms. For example, GRPO necessitates that all environments within a group share the same initial state, which can be ensured by setting "`use_fixed_reset_state_ids=True`".

8.2. GPU Allocation Modes

Configuring GPU Allocation Modes Our framework exposes a simple configuration interface that allows users to flexibly choose GPU allocation strategies without explicitly specifying the mode. Instead, users only need to assign GPU IDs for each component via the following fields:

- `cluster.component_placement.env` for the *Simulator*,
- `cluster.component_placement.rollout` for *Generation*,

- `cluster.component_placement.actor` for *Train*.

In addition, offloading can be enabled or disabled independently for each component using:

- `env.enable_offload`,
- `rollout.enable_offload`,
- `actor.enable_offload`.

Finally, fine-grained pipelining is controlled by the parameter `rollout.pipeline_stage_num`. A value greater than 1 splits each simulator instance on a GPU into multiple pipeline stages, while a value of 1 disables pipelining.

8.3. Details for Design Choices

8.3.1. Details for Multi-Granularity Support

In the main text, we refer to three levels of granularity: *token-level*, *action-level*, and *chunk-level*. Figure 14 provides an illustrative example of these levels. An *action chunk* consists of multiple atomic actions, which are the actual control signals executed by the simulator at each step. Each atomic action, in turn, is composed of multiple tokens, where each token corresponds to a single action dimension.

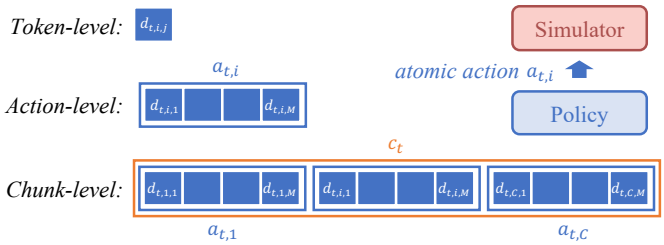


Figure 14. Illustration of different log-probability granularities.

The complete set of supported combinations between advantage computation granularity and log-probability granularity is summarized in Table 2.

Table 2. Supported combinations of advantage and log-probability granularities.

Advantage \ Log-Probability	Chunk-level	Action-level	Token-level
Chunk-level	✓	✓	✓
Action-level	✗	✓	✓

8.3.2. Details for Loss Normalization by Trajectory Length

As mentioned in the main text, we normalize the policy loss by the trajectory length under the *Valid Action Mask* setting. Specifically, if a trajectory τ_i has T_i^{succ} valid timesteps, the contribution of each timestep to the objective is scaled by $1/T_i^{\text{succ}}$. The GRPO objective under this setting is:

$$J^{\text{GRPO}}(\theta) = \mathbb{E}_{o_0 \sim D, \{\tau_i\} \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{T_i^{\text{succ}}} \sum_{t=1}^{T_i^{\text{succ}}} \min \left(\rho_{i,t}(\theta) \hat{A}_i,\ \text{clip}(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_i \right) \right]. \tag{12}$$

This prevents longer trajectories from disproportionately dominating the gradient and ensures balanced learning across trajectories of varying lengths.

9. Experiment Setup

9.1. Details for Performance Experiments

This part is the details of training and evaluation for High-Performance experiments, including metric for evaluation, base models selection and hyperparameter settings for training. Unless otherwise specified, all curves are smoothed using a Gaussian filter ($\sigma = 1$), and success rate is computed under the "`success_once`" criterion, where an episode is considered successful if the success state is reached at least once.

Table 3. Hyperparameter settings for training.

Parameter	OpenVLA [17] ManiSkill [39]		OpenVLA-OFT [18] ManiSkill		OpenVLA-OFT LIBERO [21]	OpenVLA-OFT RoboTwin [8]
	PPO	GRPO	PPO	GRPO	GRPO	GRPO
# Parallel Envs	128	256	128	256	64	128
Max Episode Steps	80	80	80	80	512	200[a]
Max Steps per Rollout Epoch	160	80	160	80	512	200[a]
Group Size	1	8	1	8	8	8
Rollout Epoch	1	1	1	1	64	8
Partial Reset	True	False	True	False	False	False
Valid Action Mask	False	True	False	True	True	True
Action Chunk Size	1	1	8	8	8	25
Global Batch Size	640	640	640	640	16384	1024
Micro Batch Size	40	40	40	40	32	32
is_lora [13]	True	True	True	True	—	True
Learning Rate	1e-4	1e-5	1e-4	1e-4	2e-5	1e-4
adam_eps	1e-8	1e-8	1e-8	1e-8	1e-8	1e-5
Clip Ratio (ϵ)	(0.2, 0.28)	(0.2, 0.28)	(0.2, 0.28)	(0.2, 0.28)	(0.2, 0.28)	(0.2, 0.28)
γ	0.99	—	0.99	—	—	—
GAE λ	0.95	—	0.95	—	—	—
Temperature (train)	1.0	1.0	1.0	1.0	1.6	1.6

[a] Note that the "pick dual bottles" and "handover block" task in RoboTwin requires 100 and 400 steps respectively.

9.1.1. Base Models for Training

For the OpenVLA (Base) in ManiSkill, we adopt the pre-trained checkpoint OpenVLA (Base) from RL4VLA [22]. For the OpenVLA-OFT (Base) in ManiSkill, we perform our own LoRA fine-tuning to get OpenVLA-OFT (Base) using motion planning data collected from the "PutOnPlateInScene25Main-v3" task.

For OpenVLA-OFT (Base) in LIBERO 130 tasks, we fine-tune OpenVLA-OFT with LoRA using demonstrations from all five task suites in LIBERO.

For OpenVLA-OFT (Base) in RoboTwin, we adopt the pre-trained checkpoint OpenVLA-OFT (Base) from SimpleVLA-RL [20]. Besides, we also reuse the train/eval seeds from SimpleVLA-RL [20].

9.1.2. Hyperparameter Settings for Training

In Table 3, we list the specific hyperparameters used for different experiments.

9.1.3. Metric for Evaluation

For supervised fine-tuned models, we set `do_sample=False` during evaluation. And for RL-trained models, we set `do_sample=True` during evaluation, and conduct three times evaluation for each model. And we report the mean and standard deviation of the success rates. (Now only the LIBERO setting has standard deviation.)

9.2. Details for Efficiency Experiments

Table 4. The setting of `enable_offload` among different GPU allocation modes.

	Simulator	Generation	Training
Disaggregated	False	False	False
Colocated	True	True	True
Hybrid	True	True	True

9.2.1. Hyperparameters for Efficiency Experiments

See Table 4 for how we enable the offload configurations and Table 5 for how we configure the GPU allocation.

Table 5. Hyperparameters of GPU allocation modes.

# GPU	Allocation Mode	Simulator	Generation	Training
8 GPUs	Disaggregated	0-1	2-3	4-7
	Colocated	0-7	0-7	0-7
	Hybrid	0-3	4-7	0-7
16 GPUs	Disaggregated	0-3	4-7	8-15
	Colocated	0-15	0-15	0-15
	Hybrid	0-7	8-15	0-15
32 GPUs	Disaggregated	0-7	8-15	16-31
	Colocated	0-31	0-31	0-31
	Hybrid	0-15	16-31	0-31

9.2.2. Metric for Efficiency

We use *throughput* as the evaluation metric for efficiency. Specifically, throughput is defined as the total number of rollout environment frames divided by the total wall-clock time of one training epoch, which approximately equals the sum of rollout time and training time.

9.2.3. Tasks for Efficiency

We evaluate our framework on all tasks in ManiSkill, LIBERO and RoboTwin.

For **ManiSkill**, we select the "PutCarrotOnPlateInScene-v2" task for a quick test. It is adapted from "PutCarrotOnPlateInScene-v1" in ManiSkill, with modifications to the reset function. In the original reset implementation, additional simulation steps were applied to ensure that all objects remained static. However, this significantly increased the reset time compared to the step function and was incompatible with partial reset. In practice, we observed that these extra steps had a negligible impact on object states in the "PutCarrotOnPlateInScene" task, since the initial position of the carrot was already carefully calibrated. We therefore delete the additional simulation steps in the reset function. The evaluation is conducted on 8, 16, and 32 NVIDIA H100 (80GB) GPUs, and we use 256 parallel environments, each running for 80 steps.

For **LIBERO**, we adopt the LIBERO-Long task set for a quick test, which is the longest task set in LIBERO. The number of parallel environments is set to 64, 128, and 256 for 8-, 16-, and 32-GPU setups, respectively, with the corresponding number of environment steps set to 4096, 2048, and 1024.[3]

For **RoboTwin**, we select the `place_empty_cup` task for a quick test, which requires the robot to use an arm to place the empty cup on the coaster. The number of parallel environments is set to 64, 128, and 256 for 8-, 16-, and 32-GPU setups, respectively, with the corresponding number of environment steps set to 800, 400, and 200. We have the same GPU allocation setting as LIBERO-Long, due to the GPU resources being the upper bound for the rollout.

9.2.4. Baselines for Efficiency Experiments

For ManiSkill, no existing framework supports multi-GPU rollout and training. Thus, we take the naive disaggregated allocation mode as the baseline, and compare it with our colocated mode and hybrid modes, where the hybrid mode includes both one-stage and two-stage fine-grained pipelining configurations.

For LIBERO and RoboTwin, we compare against SimpleVLA-RL [20], an open-source framework for RL of VLA Models training built on VeRL. Since SimpleVLA-RL only supports the colocated mode, we use it as the baseline and additionally evaluate our colocated mode and hybrid mode (with one-stage and two-stage fine-grained pipelining). We omit the disaggregated mode results for LIBERO and RoboTwin, as during training the *Training* component never reuses the same GPUs as the rollout process. Consequently, even when accounting for the offload–onload overhead, the hybrid (one-stage) configuration provides superior performance to the disaggregated mode.

10. Additional Experimental Results

10.1. Evaluation of OOD for ManiSkill

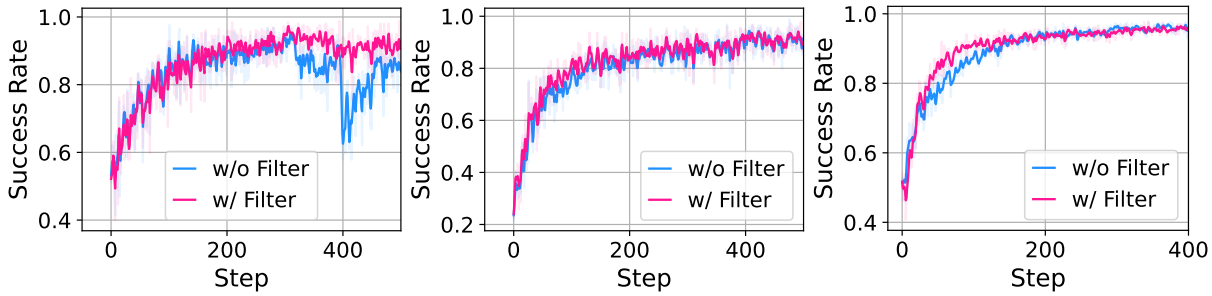
Table 6 is the detailed results ManiSkill's experiments.

[3] Since vectorized environments in LIBERO rely on multi-processing, the number of physical CPU cores per node becomes the upper bound for efficient rollout. Therefore, we scale the number of parallel environments in proportion to the number of nodes, and consequently to the total number of GPUs.

Table 6. Detailed breakdown of Out-Of-Distribution evaluation results on ManiSkill. Values denote success rates (%).

Method	Out-Of-Distribution		
	Vision	Semantic	Execution
OpenVLA (Base)	38.75	35.94	42.11
OpenVLA (RLinf-GRPO)	74.69	72.99	77.86
OpenVLA (RLinf-PPO)	82.03	**78.35**	**85.42**
OpenVLA-OFT (Base)	27.73	12.95	11.72
OpenVLA-OFT (RLinf-GRPO)	84.69	45.54	44.66
OpenVLA-OFT (RLinf-PPO)	**92.11**	64.84	73.57

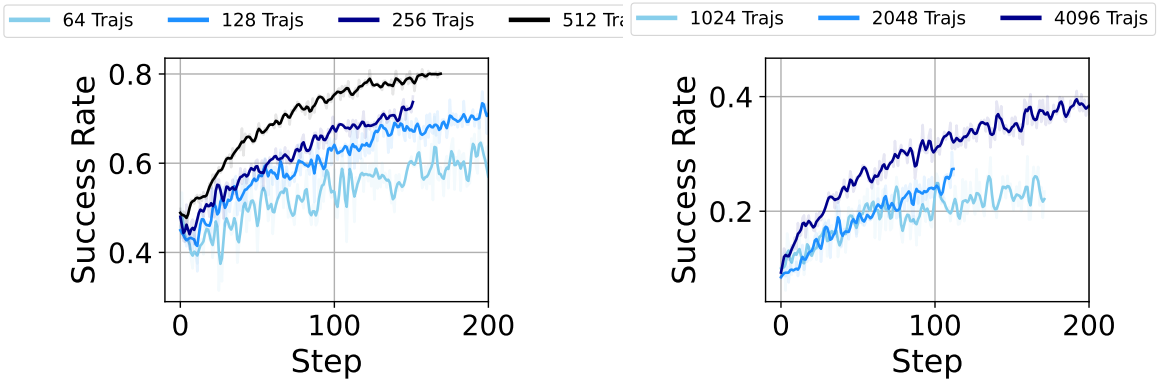
10.2. Ablation Studies



(a) OpenVLA, ManiSkill (b) OpenVLA-OFT, ManiSkill (c) OpenVLA-OFT, LIBERO-Goal

Figure 15. Ablation studies on success rate filtering under different settings.

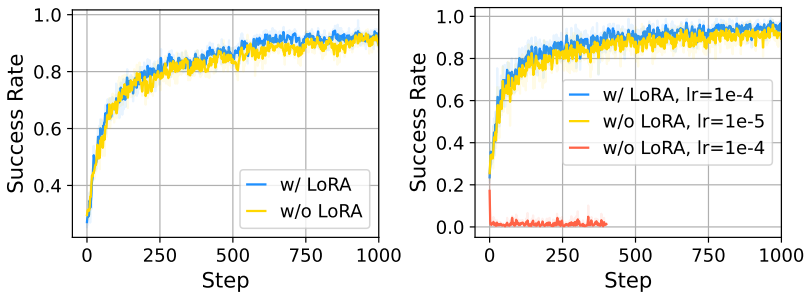
Success rate filtering As shown in Figure 15, we implement a success rate filter for GRPO that discards groups in which all trajectories have identical cumulative rewards. This mechanism can improve the stability of GRPO training. For instance, in the OpenVLA ManiSkill setting, training without the filter ("w/o Filter") exhibits a clear collapse around step 400, whereas enabling the filter ("w/ Filter") largely alleviates this issue. However, the benefit of the filter is not universal: in the OpenVLA-OFT ManiSkill setting and the OpenVLA-OFT LIBERO-Goal setting, its effectiveness is much less pronounced.



(a) PPO, LIBERO-Long (b) GRPO, LIBERO 130 tasks

Figure 16. Ablation study on rollout data size. Darker colors correspond to larger rollout datasets.

Effect of rollout data size The rollout batch size has a notable influence on RL performance, especially for on-policy algorithms such as PPO and GRPO. In general, larger rollouts per epoch enable more substantial policy improvement within each training iteration. As illustrated in Figure 16, larger rollouts consistently achieve higher success rates when evaluated by training epochs.



(a) PPO (b) GRPO

Figure 17. Ablation of LoRA in ManiSkill with OpenVLA-OFT.

Using LoRA may not directly affect performance but often requires different hyperparameters Figure 17 shows the training curves with and without LoRA [13]. The x-axis denotes the number of training epochs, while the y-axis shows the success rate in ManiSkill. The curves are overall similar, suggesting that LoRA itself does not substantially change performance. However, the choice of whether to use LoRA can influence the optimal hyperparameters. For example, in GRPO experiments (Figure 17b), using the same learning rate of 1×10^{-4} leads to normal improvement with LoRA, but the success rate without LoRA collapses to zero. In contrast, when the learning rate is reduced to 1×10^{-5}, the non-LoRA setting also achieves stable improvement. These results indicate that different LoRA configurations may require separate hyperparameter tuning.