Robotics: Science and Systems 2026
Sydney, Australia, July 13–17, 2026

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

Hongzhi Zang[1,*], Mingjie Wei[6,2,*], Si Xu[3], Yongji Wu[5], Zhen Guo[3], Yuanqing Wang[4,3], Hao Lin[3], Peihong Wang[2], Hua Yuan[3], Yixian Zhang[1], Liangzhi Shi[1], Yuqing Xie[1], Zhexuan Xu[1], Zhihao Liu[7,2], Kang Chen[4,2], Wenhao Tang[1], Quanlu Zhang[3], Weinan Zhang[6], Chao Yu[1,§,†], Yu Wang[1,†]

[1]Tsinghua University [2]Zhongguancun Academy [3]Infinigence AI [4]Peking University [5]UC Berkeley
[6]Harbin Institute of Technology [7]Institute of Automation, Chinese Academy of Sciences

[*]Equal contribution [†]Corresponding Authors: zoeyuchao@gmail.com, yu-wang@tsinghua.edu.cn [§]Project Lead

https://huggingface.co/RLinf https://github.com/RLinf/RLinf

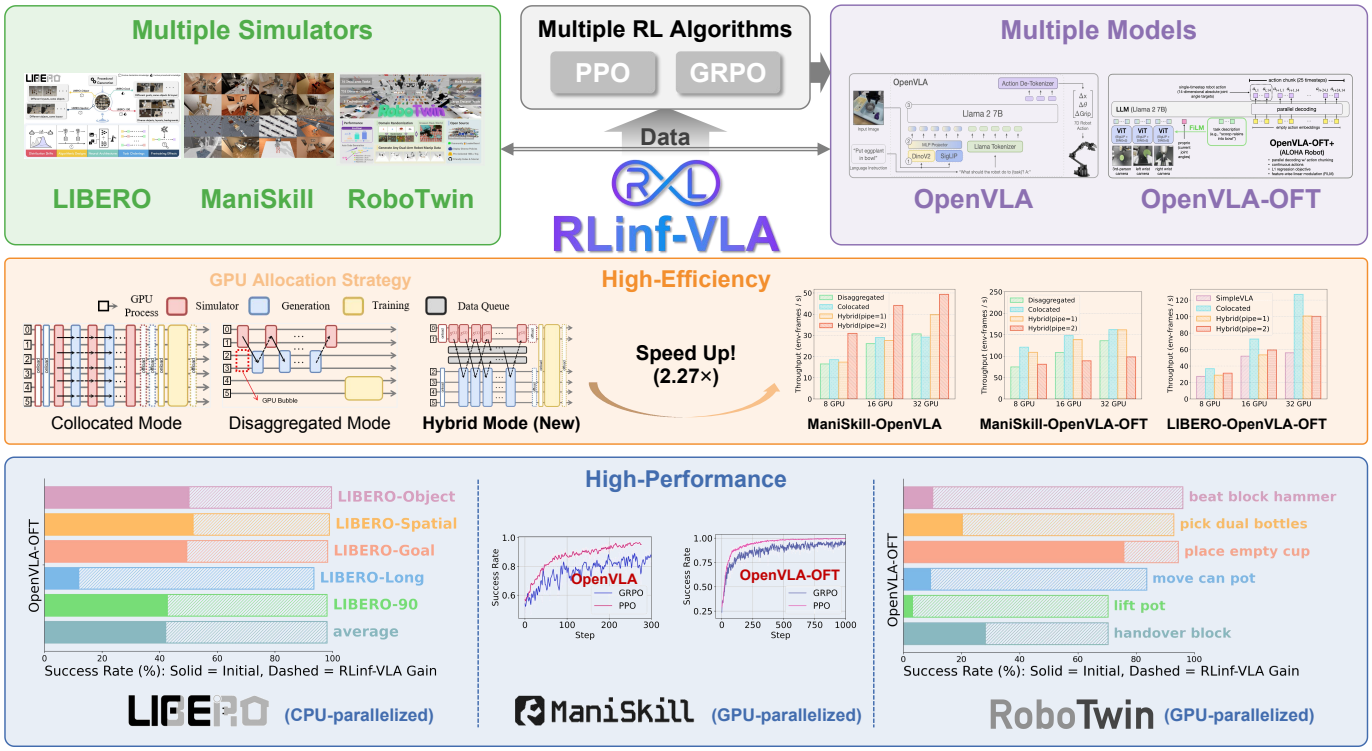


Fig. 1: Overview of RLinf-VLA. Built on a unified interface, RLinf-VLA seamlessly supports diverse VLA architectures, multiple RL algorithms, and various simulators. It provides three GPU allocation modes: *collocated*, *disaggregated*, and a novel *hybrid* mode and speeds up training by 2.27× compared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

***Abstract*—Recent studies have demonstrated the potential of reinforcement learning (RL) to improve the task performance of vision-language-action (VLA) models through interaction. However, current efforts remain fragmented, lacking a unified platform for fair comparison across architectures and algorithms, as well as an efficient system design for scalable training. Therefore, we present RLinf-VLA, a unified and efficient framework for scalable RL training of VLA models. RLinf-VLA standardizes the integration of diverse VLA architectures, RL algorithms, and heterogeneous simulators through a unified interface, enabling extensibility and reproducibility. To improve efficiency, the framework adopts a flexible resource allocation architecture for rendering, inference, and training in RL pipelines. In particular, RLinf-VLA introduces a hybrid fine-grained pipeline allocation**

strategy that achieves a 1.61×–1.88× training speedup on ManiSkill. Using this framework, RL-trained models achieve strong performance across embodied benchmarks, including 98.11% success on 130 LIBERO tasks, 97.66% success on 25 ManiSkill tasks, and 84.63% average success across 6 RoboTwin tasks. In addition, RLinf-VLA distills a set of effective practices for RL-based VLA training. We envision RLinf-VLA as a foundational framework for efficient, unified, and reproducible research in embodied intelligence.

I. Introduction

Vision-Language-Action (VLA) models represent a new generation of foundation models that aim to unify perception, language understanding, and embodied control [28, 10]. By leveraging vision-language models pretrained on internet-scale data and further training them on large, heterogeneous robot demonstration datasets [33, 19], 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 [44, 20, 26, 46, 37, 2, 18].

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 [41]. 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 [25].

Despite its promise, applying RL to VLA models remains largely small-scale or fragmented [25, 27], making the development of new RL algorithms for VLAs expensive and cumbersome. Although SimpleVLA-RL [23] enables large-scale RL training for VLAs by building on VeRL [39], 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 (e.g. ManiSkill [43], LIBERO [24], RoboTwin [9]), diverse VLA architectures (e.g. OpenVLA [20], OpenVLA-OFT [21], π_0 [2]), and reinforcement learning algorithms (e.g. PPO [36], GRPO [38]). 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 supports multiple flexible GPU allocation modes and further introduces a hybrid fine-grained pipelining mechanism. By improving resource allocation, RLinf-VLA substantially increases training throughput, achieving speedups of up to 2.27×. In addition, the framework incorporates a set of algorithmic enhancements that further improve training efficiency and stability.
- **Strong empirical performance and generalization.** After RL training with RLinf-VLA, a single model achieves 98.11% success across 130 LIBERO tasks, while another single model achieves 97.66% success across 25 ManiSkill tasks. On RoboTwin, six task-specific models achieve an average success rate of 84.63% across 6 tasks, corresponding to an average performance improvement of 63.75%. These results demonstrate RLinf-VLA's powerful performance 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.

II. Related Work

A. Reinforcement Learning for VLA

Vision-Language-Action models (VLAs) [21, 20, 2, 4, 18, 48, 5, 1, 51, 6] represent a multimodal foundation model architecture designed for robotics. RL is increasingly adopted to address the limitations of static imitation learning in VLAs. GRAPE [49] employs Direct Preference Optimization (DPO [34]) on partial trajectories to align models with human intent. Iterative approaches [14, 42] combine SFT with RL stages to balance stability and performance. VLA-RL [27] utilizes Proximal Policy Optimization (PPO [36]) to exhibit significantly stronger generalization to unseen objects and environments compared to their SFT counterparts [25]. SimpleVLA-RL [23] and related variants [42] apply Group Relative Policy Optimization (GRPO [38]) 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. $\pi^*_{0.6}$ [17] 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 [25, 27]. Although frameworks such as SimpleVLA-RL [23] 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.

B. Simulators

Simulators [22, 29, 30, 32, 31, 12, 3] 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 [43] for physics-rich manipulation, LIBERO [24] for instruction-conditioned reasoning, and RoboTwin [9] 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.

III. Preliminary

A. Reinforcement Learning Formulation

To motivate the design of our training infrastructure, reinforcement learning(RL) is commonly formulated as a partially observable Markov decision process (POMDP). A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \Omega, O)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(s' \mid s, a)$ is the transition dynamics, $r(s, a)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. Since the agent cannot directly access the underlying state, it instead receives observations $o \in \Omega$ according to the observation function $O(o \mid s)$. In vision-based manipulation tasks, observations typically consist of multi-view images, language instructions, and proprioceptive robot states, while actions correspond to continuous robot control commands or action chunks.

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 $\tau = (s_0, a_0, \ldots, s_T)$:

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

B. Action Formulation

Compared to RL for LLMs, a key distinction in RL for VLAs lies in the hierarchical formulation of actions. While LLMs typically operate on individual tokens [38], VLA policies often predict either a single action [20] or an action *chunk* containing multiple actions to ensure smooth control [50]. This leads to a three-level hierarchy consisting of *Chunk*, *Atomic Action*, and *Token*. Each chunk may contain multiple atomic actions, and each atomic action is composed of multiple tokens, where each token typically corresponds to a specific dimension of the robot action space (e.g., end-effector pose or joint angles).

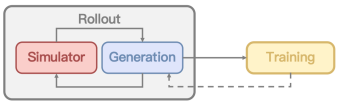


Fig. 3: Pipeline of Reinforcement Learning for VLA models

C. 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 3. GPU resources are primarily allocated between the rollout phase (*Generation* and *Simulator*) and the optimization phase (*Training*). The interaction typically proceeds as follows [21]: 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.

IV. Method

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

A. GPU Allocation Strategies

The RL training pipeline (described in Section III-C) 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.

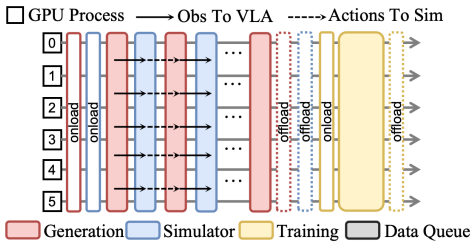


Fig. 4: Collocated Mode.

1) The Collocated Mode: In this mode (Figure 4), 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 component 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.

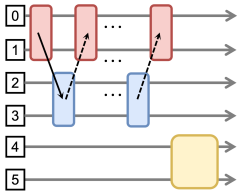


Fig. 5: Disaggregated Mode (see Figure 4 for legend).

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

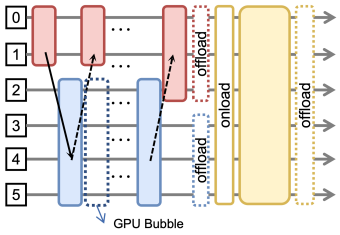


Fig. 6: Hybrid Mode (see Figure 4 for legend).

3) The Hybrid Mode with Fine-grained Pipelining: To overcome the drawbacks of the above modes, we propose a hybrid mode, where each component can flexibly select GPUs. In contrast, existing distributed RL systems are less flexible. For example, veRL [40] primarily adopts a collocated design, while AReaL [11] and SLiME [52] use a disaggregated design but still allocate GPUs to each component in contiguous blocks.

In our hybrid mode, a typical configuration is to assign *Generation* and *Simulator* to different GPU partitions, while allowing *Training* to utilize all GPUs. Figure 6 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".

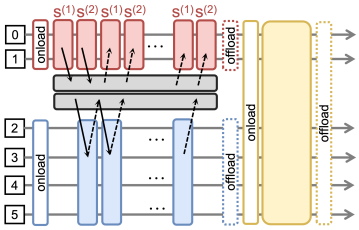


Fig. 7: Hybrid Mode with Fine-grained Pipelining ($k = 2$; see Figure 4 for legend).

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 with $k = 2$ 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 6 provides a schematic illustration of the hybrid allocation mode with fine-grained pipelining ($k = 2$).

4) The Auto-Placement Algorithm: Since RLinf-VLA supports multiple GPU allocation modes, automatically determining an efficient allocation strategy is important. To this end, we introduce a lightweight automatic GPU placement algorithm that combines runtime profiling with allocation optimization.

We begin with several practical assumptions. Given N GPUs, we assume the *Training* component utilizes all N GPUs whenever active, as this is typically optimal for compute-intensive workloads. We therefore focus on optimizing the *Rollout* phase, which consists of the *Simulator* and *Generation* components. We consider GPU splitting only between these two components and restrict the pipeline depth to at most two stages. Given N GPUs and n_S simulation environments, our goal is to determine the GPU allocation (N_S, N_G) and pipeline stage number $k \in \{1, 2\}$ that minimize rollout time, where

N_S and N_G denote the numbers of GPUs allocated to the *Simulator* and *Generation* components, respectively.

Profiling. We estimate the execution time functions $t_G(\cdot)$ and $t_S(\cdot)$ by profiling batch sizes $\{2^0, 2^1, \ldots, n_S\}$, yielding estimates of per-chunk execution latency.

Allocation. Let m denote the number of chunks per epoch. We solve $\min_{k\in\{1,2\},\ N_G+N_S=N} t_k$, where

Collocated ($k = 1$): $t_1 = m\left(t_G\left(\frac{n_S}{N}\right) + t_S\left(\frac{n_S}{N}\right)\right)$,

Pipelined ($k = 2$): t_2 is estimated from $t_G(\frac{n_S}{N_G})$ and $t_S(\frac{n_S}{N_S})$ under pipelined execution.

B. A Unified Interface

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

1) Multiple Simulators Support: RLinf-VLA supports multiple robotic simulators, including ManiSkill [43], LIBERO [24], and RoboTwin [9]. 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.

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 [20] and OpenVLA-OFT [21], which represent configurations with single-step and multi-step action chunking, respectively. To enable efficient fine-tuning, we integrate Low-Rank Adaptation (LoRA) [16].

3) Multiple Algorithms Support: Our framework provides support for multiple reinforcement learning algorithms, with an initial focus on Proximal Policy Optimization (PPO) [36] and Group Relative Policy Optimization (GRPO) [38, 13]. 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.

For off-policy reinforcement learning algorithms, their application to VLA models remains an open problem, particularly in terms of achieving stable training. Nevertheless, our pipeline is fully functional and extensible to off-policy settings. As a preliminary extension, RLinf-VLA supports DSRL [45], a variant of Soft Actor-Critic (SAC [15]), for fine-tuning π-series models. However, off-policy methods are not the primary focus of this work, and we therefore limit our discussion.

C. 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 IV-C1, we introduce general features applicable to all algorithms. Subsequently, in Section IV-C2 and Section IV-C3, 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.

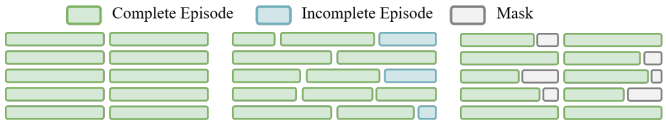
1) Multi-Granularity Calculation Support: Our framework supports advantage estimation and log-probability computation at multiple levels of granularity, enabling seamless adaptation to different algorithmic requirements.

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

Fig. 8: Illustration of the three rollout modes.

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 8a and Figure 8b 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 expensive 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 [25], we attach a

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

lightweight value head to the language model component of the VLA for efficient state value estimation.

Value for Action Chunks As discussed in Section IV-C1, 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* 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.

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 8c). 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 [47], 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.

V. 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 over the base model, 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 $1.88\times$.

TABLE I: 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**
SimpleVLA-RL	94.2	87.5	68.3	61.2	64.1	57.8	72.18

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.

(4) Can simulation-based RL benefit real-world deployment? We conduct a preliminary real-world deployment study to examine whether policies improved through RL in simulation can transfer to physical robots. Compared with SFT-only baselines, RL post-training leads to improved real-world performance in our setup, suggesting the potential practical value of simulation-based RL for embodied learning.

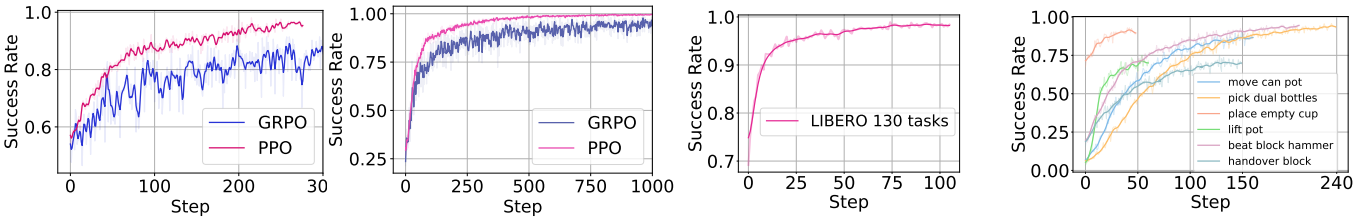
(5) How extensible is RLinf-VLA across models and algorithms? We further evaluate RLinf-VLA on additional VLA architectures and RL algorithms, including π-series models and off-policy methods. The results in Section X-A demonstrate the extensibility of the framework across diverse training settings.

A. 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-



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

Fig. 9: 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 $\sigma = 1$.

distribution evaluation protocol of RL4VLA [25], measuring generalization in *vision*, *language*, and *action*. Each subsetting is evaluated with 256 random episodes. For LIBERO, we use the public task groups [24], 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 [9], we select six tasks: "beat block hammer", "move can pot", "place empty cup", "pick dual bottles", "lift pot", and "handover 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 9 presents the training curves for each setup. Figure 9a and Figure 9b 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 9c 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 9d 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 I summarizes RLinf-VLA results across three benchmarks, with ManiSkill detailed in Table Ia. "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 Ib 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 methods that train separate models per group, demonstrating RLinf-VLA's support for large-scale multi-task RL.

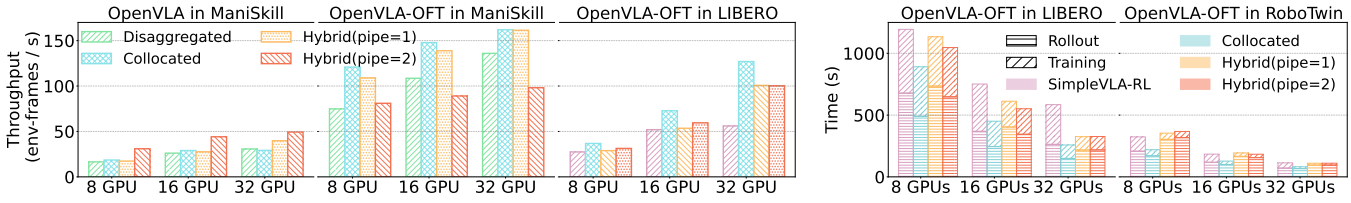
OOD results on RoboTwin are shown in Table Ic. RLinf-VLA achieves an average success rate of 84.63%, over 60% improvement. We also compare with the most relevant baseline, SimpleVLA-RL [23]. RLinf-VLA consistently outperforms it across all RoboTwin tasks, with particularly large gains on challenging tasks such as Bottles, Can, and Hand. These results demonstrate the effectiveness of our system design.

B. System Efficiency

In this section, we evaluate the efficiency of our system by comparing different GPU allocation modes. 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 modes (Section IV-A). *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 [23] (collocated only) and evaluate our *Collocated* and *Hybrid* modes; the disaggregated mode is omitted since hybrid consistently performs better when GPUs are underutilized.



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

Fig. 10: 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`").

Results. Figure 10 reports the end-to-end throughput of RLinf-VLA and baselines for RL training under different cluster sizes and GPU allocation modes.

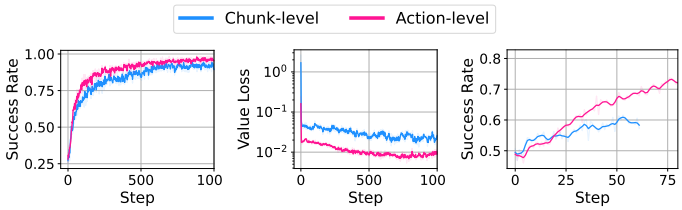
From Figure 10a, for OpenVLA in ManiSkill, RLinf-VLA achieves up to a 1.88× speedup over the disaggregated baseline using the Hybrid (`pipe=2`) mode on 8 GPUs. Although scaling to more GPUs introduces overheads from model loading, offloading, and state switching, fine-grained pipelining reduces GPU idle 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 IV-A1), 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 10b, 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 modes. RLinf-VLA's flexible multi-mode support enables consistently high efficiency across diverse execution regimes.



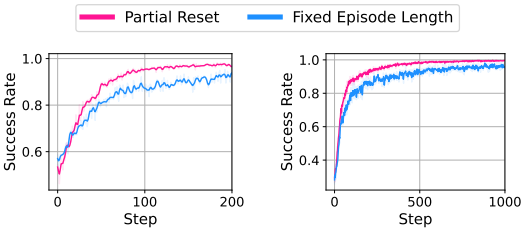
(a) succ. in ManiSkill (b) loss in ManiSkill (c) succ. in LIBERO

Fig. 11: For OpenVLA-OFT, action-level value estimation consistently outperforms chunk-level estimation across different tasks.

C. Ablation Study

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

1) Tips for PPO: **Action-level value estimation outperforms chunk-level estimation for PPO with action chunks.** Figure 11 shows the PPO training curves comparing action-level and chunk-level value estimation as described in Section IV-C2 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 11c).



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

Fig. 12: Partial reset consistently outperforms Fixed Episode Length mode in PPO.

Partial reset substantially improves sample efficiency. Figure 12 presents the PPO training curves for the *Partial Reset* and *Fixed Episode Length* rollout modes discussed in Section IV-C2. 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).

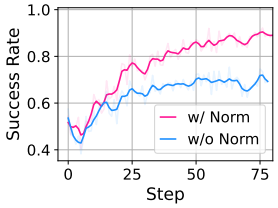
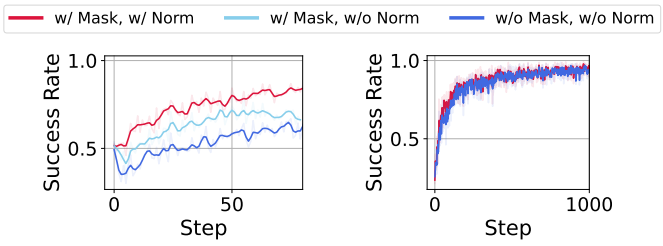


Fig. 13: Ablation study on trajectory length normalization. Success rate in LIBERO-Goal with OpenVLA-OFT.

2) Tips for GRPO: **Trajectory length normalization.** As discussed in Section IV-C3, normalizing the loss by trajectory length aims to reduce bias when episodes vary in length. Figure 13 shows that incorporating trajectory length normalization ("w/ Norm") can lead to substantially higher performance compared with the unnormalized setting ("w/o Norm").

Valid action mask. We also study the effect of valid action masking, introduced in Section IV-C3. 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 14a 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

Fig. 14: 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 IV-C3, 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.

Fig. 15: Deployment in the real world.

D. Real-World Experiment

We conduct a preliminary real-world experiment to evaluate whether RL improvements obtained in simulation can transfer to physical deployment.

Specifically, we consider a drawer-closing task using OpenVLA and compare an SFT-only model with a model further optimized through RL in simulation. The SFT model is trained using 20 real-world expert demonstrations together with 1000 simulated trajectories. The RL model is initialized from the SFT checkpoint and further trained with RL in simulation. To reduce the sim-to-real gap, the camera pose and robot arm configuration in the real-world setup are calibrated to match the simulation environment.

In real-world deployment, both the SFT and RL models achieve a success rate of 8/10. However, the RL model completes the task more efficiently (Fig. 15). We observe that the SFT model tends to imitate suboptimal human teleoperation behaviors, such as rotating before pushing the drawer, and occasionally predicts near-zero actions. In contrast, the RL-trained policy learns a more direct and fluent pushing behavior. Although limited in scale, these results suggest that simulation-based RL can improve the quality and efficiency of real-world policy execution.

VI. 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 high performance 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.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (No.62406159, 62325405), Zhongguancun Academy (Grant No. C20250301), Beijing National Research Center for Information Science, Technology (BNRist) and Beijing Innovation Center for Future Chips.

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] 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. URL https://arxiv.org/abs/2410.24164.

[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.

[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.

[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.

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

[8] Kang Chen et al. $\pi_{\texttt{RL}}$: Online rl fine-tuning for flow-based vision-language-action models, 2026. URL https://arxiv.org/abs/2510.25889.

[9] 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. URL https://arxiv.org/abs/2506.18088.

[10] 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. doi: 10.1177/02783649241281508.

[11] Wei Fu et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025. URL https://arxiv.org/abs/2505.24298.

[12] 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.

[13] 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.

[14] 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.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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. URL https://doi.org/10.48550/arXiv.2403.09227.

[23] 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. URL https://openreview.net/forum?id=TQhSodCM4r.

[24] 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.

[25] 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. URL https://openreview.net/forum?id=qmBMPInbZC.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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. doi: 10.1109/LRA.2022.3172793.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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. URL https://api.semanticscholar.org/CorpusID:3075448.

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

[37] 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.

[38] 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.

[39] 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*, EuroSys '25, page 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

[40] Guangming Sheng et al. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

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

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

[43] 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.

[44] 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.

[45] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *Conference on Robot Learning*, 2025.

[46] 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.

[47] 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. URL https://openreview.net/forum?id=2a36EMSSTp.

[48] 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.

[49] 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.

[50] 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. URL https://doi.org/10.15607/RSS.2023.XIX.016.

[51] 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.

[52] Zilin Zhu et al. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.