RL-VLA^3: A FLEXIBLE AND ASYNCHRONOUS REINFORCEMENT LEARNING FRAMEWORK FOR VLA TRAINING

Zhong Guan[1,4*], **Haoran Sun**[2,4*], **Yongjian Guo**[3,4*], **Shuai Di**[4*], **Xiaodong Bai**[4], **Jing Long**[2,4], **Tianyun Zhao**[3,4], **Mingxi Luo**[4], **Chen Zhou**[4], **Yucheng Guo**[4], **Qiming Yang**[4], **Wanting Xu**[4], **Wen Huang**[3,4], **Yunxuan Ma**[2,4], **Hongke Zhao**[1], **Likang Wu**[1], **Xiaotie Deng**[2], **Xi Xiao**[3], **Sheng Wen**[5], **Yicheng Gong**[4], **Junwu Xiong**[4†]

[1]Tianjin University, [2]Peking University, [3]Tsinghua University, [4]JDT AI Infra,
[5]Swinburne University of Technology

ABSTRACT

In recent years, Vision-Language-Action (VLA) models have emerged as a crucial pathway towards general embodied intelligence, yet their training efficiency has become a key bottleneck. Although existing reinforcement learning-based training frameworks like RLinf can enhance model generalization, they still rely on synchronous execution, leading to severe resource underutilization and throughput limitations during environment interaction, policy generation (rollout), and model update phases (actor). To overcome these challenges, this paper, for the first time, proposes and implements a fully-asynchronous policy training pipeline -RL-VLA^3, encompassing the entire 3 asynchronous pipeline from environment interaction, rollout generation, to actor policy updates. Systematically drawing inspiration from asynchronous optimization ideas in large model RL, our framework designs a multi-level decoupled architecture. This includes asynchronous parallelization of environment interaction and trajectory collection, streaming execution for policy generation, and decoupled scheduling for training updates. We validated the effectiveness of our method across diverse VLA models and environments. On the LIBERO benchmark, the framework achieves throughput improvements of up to 59.25% compared to existing synchronous strategies. When deeply optimizing separation strategies, throughput can be increased by as much as 126.67%. We verified the effectiveness of each asynchronous component via ablation studies. Scaling law validation across 8 to 256 GPUs demonstrates our method's excellent scalability under most conditions.

1 INTRODUCTION

Recent years have witnessed substantial advancements in multimodal understanding and generation facilitated by vision-language foundation models (Zhou et al., 2025), prompting researchers to increasingly focus on extending these abilities to embodied environments. VLA models, as an emerging paradigm of foundation models (Kawaharazuka et al., 2025), aim to unify perception, language understanding, and robot control, providing a crucial pathway toward general robotic intelligence (Allshire et al., 2025). Currently, mainstream training methods still rely on supervised fine-tuning (Shukor et al., 2025; Dana Aubakirova et al., 2025). Examples include NVIDIA's GR00T (Bjorck et al., 2025) and the open-source project Lerobot from Hugging Face (Cadene et al., 2024), both of which have gained wide recognition for enabling large-scale training of various VLAs. Although this approach performs well in fitting expert demonstration data, the high cost of large-scale expert demonstrations limits its applicability. Moreover, its vulnerability to distribution shifts restricts the model's generalization ability and robustness. Due to the compounding errors over time steps, the policy is prone to entering states not covered by the training data, leading to performance degradation in unfamiliar tasks or environments.

*Equal Contribution
†Corresponding To xiongjunwu.1@jd.com

To overcome the above limitations, reinforcement learning offers a complementary approach by directly optimizing cumulative task rewards through interactive trial and error. Unlike supervised paradigms, RL allows the agent to explore beyond the expert's data manifold, fostering the discovery of robust error-recovery strategies crucial for zero-shot generalization (Li et al., 2025). However, applying RL to VLA models faces systematic challenges: most existing studies are limited in scale and fragmented across platforms, lacking a unified framework to support large-scale and diverse experiments. Simultaneously, reinforcement learning training for VLAs requires continuous interaction with environments, which are typically implemented via simulators. These simulators compete with model training and inference for memory and computational resources, further complicating system optimization.

RLinf (Zang et al., 2025), as the industry's first open-source framework implementing reinforcement learning for VLAs, provides critical infrastructure by designing unified interfaces, supporting multiple types of simulators and algorithms, and introducing flexible GPU allocation strategies such as colocated, disaggregated, and hybrid configurations. However, its training pipeline is inherently constrained by a synchronous execution paradigm, which underutilizes the parallel processing capabilities of heterogeneous computing clusters. The serial dependency between simulator interaction, policy generation, and model updates leads to idle computational resources and throughput bottlenecks, limiting further improvements in training efficiency. In current practices of RL for large models, asynchronous training mechanisms (Han et al., 2025; Fu et al., 2025; Team et al., 2025; Sheng et al., 2025) have been proven to significantly enhance training efficiency and system throughput, with related research giving rise to various mature asynchronous optimization strategies. Nevertheless, in the field of reinforcement learning training for VLA models, such asynchronous training methods remain largely unexplored and have yet to be adequately practiced.

To bridge these gaps, this paper systematically introduces asynchronous training and inference mechanisms based on the existing unified framework. Drawing inspiration from asynchronous optimization ideas in large-model reinforcement learning, we design a 3-level asynchronous execution architecture RL-VLA^3,including asynchronous environment interaction and trajectory collection, streamed asynchronous execution of policy generation, and decoupled scheduling of training updates. These mechanisms significantly reduce waiting times between components, achieving sustained and saturated utilization of computational resources. Experiments demonstrate that the proposed asynchronous framework achieves significant improvements in training throughput and overall training time while maintaining good training stability and policy performance.

The main contributions of this paper are as follows:

- We propose a reinforcement learning training framework for VLA models that supports hierarchical asynchronous execution, systematically alleviating the resource idling issues in synchronous training.
- Through the decoupled orchestration of streamed generation and environment interaction, we achieve high-concurrency execution of inference and simulation.
- We validate the efficiency and generalization capability of the framework across multiple benchmark simulation tasks and real robot deployments, providing a more efficient training platform for large-scale embodied intelligence research.

The framework proposed in this paper will be open-sourced subsequently to promote reproducible research and technological iterations in this field.

The paper proceeds as follows. We begin by discussing related work in Section 2. Section 3 introduces our proposed strategy. We then present and analyze extensive experimental results in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

Vision-Language-Action Models and Supervised Baselines. Current VLA models typically integrate a pre-trained Vision Transformer (ViT) with an LLM backbone to map multimodal observations directly to robotic actions. OpenVLA (Kim et al., 2024) stands out as a prominent open-source implementation in this domain. Built upon the Llama 2 architecture while incorporating DINOv2 and SigLIP visual encoders, OpenVLA discretizes the action space into tokens to generate actions

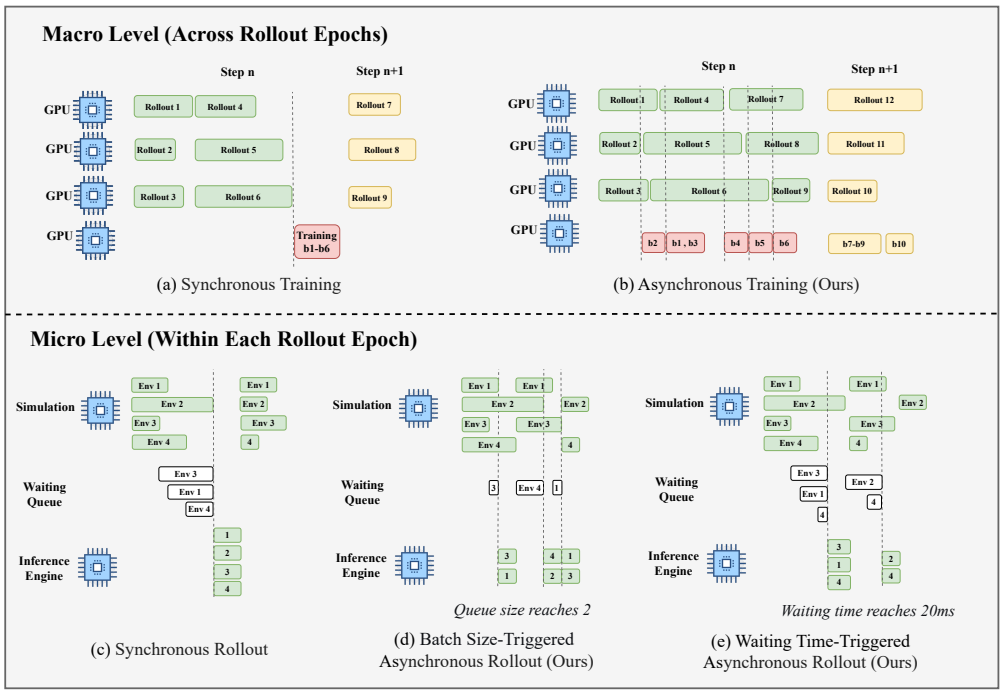


Figure 1: An illustration of the proposed asynchronous framework for VLA training. The design operates asynchronously on two levels: (1) macroscopically, between the rollout and training pipeline stages, and (2) microscopically, within each rollout epoch. The latter is governed by two trigger strategies: batch size and waiting time.

autoregressively. Conversely, $\boldsymbol{\pi_0}$ (Black et al., 2024) introduces a flow-matching mechanism tailored for robotic control. Unlike the discrete tokenization of OpenVLA, π_0 models continuous action distributions using diffusion-based processes. Regarding training frameworks, prior work predominantly focuses on Imitation Learning (IL) Zare et al. (2023) and SFT Xiao et al. (2025). NVIDIA's GR00T (Bjorck et al., 2025) leverages the Isaac Lab Mittal et al. (2025) simulation environment to generate massive amounts of synthetic demonstration data, facilitating the learning of complex manipulation and locomotion skills through supervised regression. Similarly, LeRobot (Cadene et al., 2024) provides an open-source library that standardizes the training of diverse policy architectures on unified datasets.

Reinforcement Learning and Infrastructure for VLAs. To address the limitations of SFT, RL has been increasingly applied to fine-tune VLA models. In the VLA domain, RLinf (Zang et al., 2025) serves as the primary infrastructure. While RLinf successfully mitigates memory contention between physics simulators and VLA models through resource disaggregation, it relies on a synchronous execution model. This lockstep synchronization between environment stepping and model inference leaves computational resources idle during sequential execution. The asynchronous parallelism (Sheng et al., 2025) that has proven effective in LLM training has not yet been effectively adapted to the specific constraints of VLA-RL, where the rendering overhead and strict synchronization requirements of physics engines present unique scheduling challenges.

3 METHODOLOGY

This section delineates the methodological framework designed to overcome key efficiency bottlenecks in the reinforcement learning post-training of VLA models. We introduce three core, synergistic innovations that collectively enable a high-throughput, asynchronous training paradigm. First, we describe an asynchronous training architecture that decouples trajectory generation from policy

optimization in Subsection 3.1. Subsequently, we detail a fine-grained interaction strategy for environment inference in Subsection 3.2 and a streamed execution mechanism for the training phase in Subsection 3.3, which together minimize idle computational resources and pipeline stalls.

3.1 ASYNCHRONOUS TRAINING AND INFERENCE

Currently, in reinforcement learning post-training for VLA models, conventional synchronous training mechanisms typically rely on a colocated GPU allocation strategy. This approach collects trajectory data for policy updates only after all rollout processes are completed. While straightforward and stable in many tasks, this method faces challenges when trajectory generation times are unevenly distributed, and certain environment interactions exhibit significant long-tail latencies. The synchronous waiting mechanism leads to idle computational resources, significantly impairing overall training throughput. Furthermore, when alternately loading rollout environments and policy models on a single GPU, frequent context switching and data migration introduce additional system overhead, further constraining training efficiency.

To address these challenges, we have designed and implemented an asynchronous training architecture based on a disaggregated GPU allocation strategy. In this architecture, rollout workers and actor workers are deployed on separate GPU devices, communicating through a high-throughput pipeline for trajectory data transmission. Once any rollout worker completes the generation of its current trajectory, it immediately places the trajectory into the transmission queue and proceeds to generate subsequent trajectories based on the current policy version, without waiting for other ongoing environment interactions. This mechanism effectively avoids blockage of overall progress by long-tail rollouts, enabling rollout workers to maintain near-continuous trajectory generation. When the accumulated trajectories reach the predefined training batch size, the actor worker asynchronously collects these trajectories from the communication pipeline to perform policy optimization and parameter updates. Notably, while the actor is training, rollout workers continue generating new trajectories using the pre-updated policy weights until the actor completes the current parameter update, at which point the new policy weights are synchronized to all rollout workers. This design not only achieves decoupling and parallelization between training and generation but also ensures the stability and data validity of the training process.

Through this asynchronous training mechanism, we have, for the first time in reinforcement learning training for VLA models, achieved efficient pipeline execution between rollout and training. This framework significantly reduces the time overhead caused by synchronous waiting and resource switching, enhancing system throughput while shortening the overall training cycle. It provides a more efficient training infrastructure for embodied intelligence research aimed at large-scale, multi-task scenarios.

3.2 ASYNCHRONOUS INTERACTION STRATEGY

In the RL training of VLA models, the efficiency of sample trajectory generation serves as the primary bottleneck for system throughput. Conventional VLA training frameworks typically employ a synchronous batch parallel mode, where interactions between the simulation environments and the inference model occur at the granularity of entire batches. This paradigm introduces rigid synchronization dependencies:

- **Environment-side Long-tail Effect:** Global progression is constrained by the slowest simulation instance, leading to significant idle computational resources due to synchronization overhead.
- **Inference-side Pipeline Stalls:** Large-batch inference requests must be fully processed before being returned to the environments, causing substantial idle time in environmental resources.

To mitigate these issues, we propose a fine-grained **Asynchronous Interaction Strategy**. This mechanism manages simulation instances at a higher resolution and decouples environment step from model inference. It allows ready requests from a subset of environments to enter the inference queue prematurely, thereby bypassing the requirement for global synchronization.

Furthermore, our approach differs from the "continuous batching" techniques used in LLMs. While LLMs benefit from token-level generation and management that enables the dynamic insertion and removal of requests during execution, VLA models often integrate Diffusion modules. Since the action generation process in these modules is characterized by non-autoregressive or multi-step denoising properties, it precludes the direct application of token-level dynamic insertion. Consequently, we implement a **Dynamic Batching Scheduler**. This scheduler utilizes the maximum inference batch size (B_{max}) and the maximum waiting latency (T_{max}) as constraints:

$$\text{Trigger Inference if: } (\text{Batch Size} \geq B_{max}) \vee (\text{Wait Time} \geq T_{max}) \tag{1}$$

By implementing this strategy, we effectively compress the time required for batch trajectory generation without compromising model performance.

3.3 Streamer Generation

In the asynchronous training and inference architecture, although the rollouts and training processes have been decoupled, the actor still needs to accumulate a sufficient number of trajectories to form a complete training batch, which may still cause intermittent GPU idleness. To fully utilize these waiting intervals, we further introduce a fine-grained pipeline partitioning mechanism during the training phase. Specifically, we divide the global training batch into several micro-batches. Whenever the number of samples accumulated in the trajectory buffer reaches the size of a single micro-batch, the actor initiates a forward and backward computation. After all micro-batches have been computed sequentially, gradient aggregation and parameter updates are performed collectively. This design allows the actor to begin partial training computations ahead of time without waiting for the entire batch of trajectories to be fully generated, thereby effectively masking data preparation time, significantly reducing end-to-end training latency, and further improving GPU utilization and system throughput.

Furthermore, we observe that in typical VLA training scenarios, the time required for a single training step is often significantly shorter than the rollout time needed to generate the corresponding trajectories. To achieve balanced utilization of computational resources, we systematically analyze and dynamically adjust the GPU resource allocation ratio between rollout workers and actor workers in the asynchronous framework. Experiments show that by adjusting the GPU allocation ratio between them to configurations such as 3:1, 2:1 or 1:1, the execution time distribution across stages can be significantly altered. When the time costs of both processes approach equilibrium, the training and rollout processes can achieve nearly complete computational masking in the asynchronous pipeline, thereby optimizing overall system utilization. Detailed experimental results and resource configuration analysis are provided in the experiment section of this paper in Section 4.2.2. This resource allocation mechanism further enhances the adaptability and efficiency of the asynchronous training framework, providing empirical guidance for optimal resource deployment in different task scenarios.

4 Experimental Results

In this section, we integrate our asynchronous strategies into the existing VLA RL training framework proposed by Zang et al. (2025). We conduct extensive experiments across different backbone models, environments, training algorithms, and GPU resource scales. Through a comprehensive comparison with baseline methods, we demonstrate that our proposed asynchronous strategies improve throughput while preserving training performance.

4.1 Experiment Setup

Models. We evaluate several pretrained VLAs, including diffusion-based models: GR00T N1.5 (Bjorck et al., 2025) and the π series (π_0 and $\pi_{0.5}$) (Black et al., 2024), and an autoregressive model, OpenVLA-OFT (Kim et al., 2025). Following Zang et al. (2025), we adapt OpenVLA-OFT to predict a chunk of actions rather than a single action. This selection of heterogeneous models allows us to validate the universality and robustness of our method across different VLA architectures and action prediction paradigms.

Benchmarks. We consider two environments for RL training: LIBERO (Liu et al., 2023) and ManiSkill (Mu et al., 2021). Both are widely used for evaluating pretrained VLA models and post-training algorithms. For LIBERO, we primarily use the LIBERO-10 task split from LIBERO-100, which focuses on lifelong robot decision-making. For ManiSkill, we focus on pick-and-place tasks with varying objects and receptacles.

A key distinction between these environments lies in their parallelization design: LIBERO relies on CPU multi-processing, whereas ManiSkill uses GPU tensor parallelization. As noted by Zang et al. (2025), these different parallelization approaches lead to distinct optimal RL strategies. We will demonstrate in the next subsection how our strategy adapts to these different environment backends to maintain efficiency.

Implementation and Baselines. Our implementation builds on RLinf (Zang et al., 2025), a framework that partitions RL training into three worker types: Env, $\mathrm{Rollout}$, and Actor workers. We modify all three worker types and introduce additional communication channels to support asynchronous execution. The modified framework is illustrated in Figure 1 and detailed in Section 3.

We compare against two distributed training strategies supported by RLinf:

- *Colocated*: All three worker types are placed on the same devices, meaning all devices simultaneously perform environment interaction, model rollout, or model training.
- *Disaggregated*: Workers are placed on separate GPU devices, though model training and rollout remain sequentially synchronized.

For the disaggregated strategy, unless otherwise noted, we co-locate Env and $\mathrm{Rollout}$ workers on one set of devices, and place Actor workers on another.

To rigorously evaluate performance, we conduct experiments across GPU clusters ranging from 8 to 256 GPUs. For the disaggregated strategy, we further explore different GPU allocation ratios among worker types.

Evaluation Metrics. Since our primary objective is to accelerate training with minimal impact on final performance, we evaluate both system **throughput** and **training success rate curves**.

Currently, there is no standardized throughput metric for VLA models. Metrics from language modeling are not directly applicable, as VLA models typically predict variable-sized action chunks. Therefore, we measure overall system efficiency using the number of environment state computations per unit time:

$$\mathrm{Throughput} = \frac{N_{re} \times N_{env} \times N_{es}}{T_s}, \tag{2}$$

where N_{re} is the total rollout epochs, N_{env} is the number of environment instances per epoch, N_{es} is the average environment steps per instance, and T_s is the average step execution time. In our experiments, N_{es} is fixed per instance. For fair comparison, we keep all hyperparameters consistent across experiments, as listed in Table 2.

For the training success rate curves, we adopt the standard evaluation protocol for VLA models. The detailed analysis is presented in Appendix B.

4.2 Result Analysis

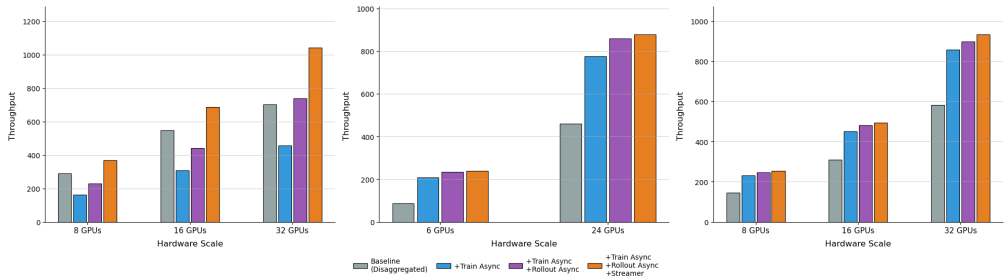
4.2.1 Throughput Comparison

We conduct an experiment comparing the throughput of two existing placement strategies, Colocated and Disaggregated, with our proposed methods that employ varying degrees of asynchrony. Table 1 presents the throughput results across different models, environments, and GPU resource scales. Our proposed methods are evaluated in three progressively enhanced configurations: 1) **Train Async**, which introduces asynchrony between the training and rollout phases; 2) **Rollout Async**, which adds asynchrony between environment simulation and inference within the rollout phase; and 3) **Streamer**, a streaming scheduler that further optimizes pipeline execution.

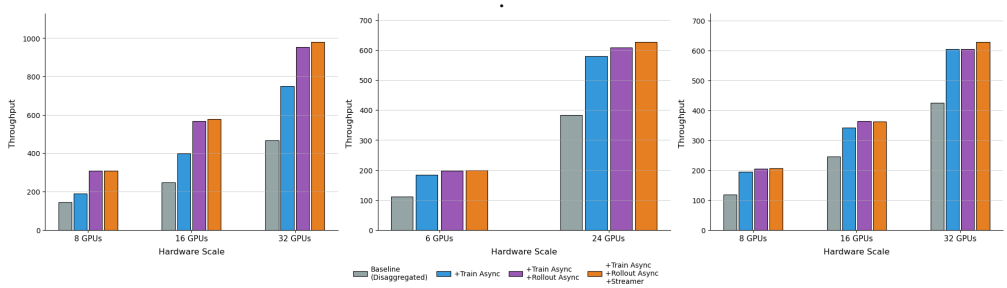
As shown in the Table 1, our asynchronous strategies significantly improve throughput compared to the Disaggregated baseline and surpass the Colocated strategy in most scenarios. The Train

Table 1: Throughput comparison of training strategies. Our asynchronous methods (Train Async, Rollout Async) outperform the Disaggregated baseline and, in most cases, the Colocated baseline across different models and environments. The best value for each column is **bolded**. And the Increase% is compare to colocated stratedies.

Configuration	LIBERO+$\pi_{0.5}$			LIBERO+GR00T N1.5			ManiSkill+π_0		
	8 GPUs	16 GPUs	32 GPUs	8 GPUs	16 GPUs	32 GPUs	8 GPUs	16 GPUs	32 GPUs
Colocated	289.23	547.55	703.85	371.80	680.46	1125.62	**132.56**	232.23	370.26
Disaggregated (1:1)	162.75	307.84	457.23	220.81	409.60	729.98	60.64	150.59	257.21
Asynchronous improvements on Disaggregated (cumulative):									
+ Train Async	229.68	441.49	737.46	243.57	477.20	951.33	126.65	**244.61**	**436.32**
+ Rollout Async	369.56	686.80	1041.36	434.20	816.48	**1620.39**	69.07	139.32	275.36
+ Streamer	**383.40**	**713.38**	**1120.91**	**439.64**	**816.48**	1592.40	71.61	141.49	280.07
Increase %	**↑32.5%**	**↑30.29%**	**↑59.25%**	**↑ 18.25%**	**↑19.99%**	**↑ 43.96%**	**↓-4.46%**	**↑ 5.33%**	**↑17.84%**



(a) LIBERO + $\pi_{0.5}$. The left, center, and right figures represent rollouts with GPU allocation ratios of 1:1, 2:1, and 3:1 between actors, respectively.



(b) LIBERO + OpenVLA-OFT. The left, center, and right figures represent rollouts with GPU allocation ratios of 1:1, 2:1, and 3:1 between actors, respectively.

Figure 2: Comparative throughput analysis of different separation strategies.

Async strategy, which parallelizes the rollout phase (involving the Env and Rollout workers) with the training phase (the Actor worker), provides a substantial throughput benefit across all configurations. The effect of the Rollout Async strategy, however, varies between environments. For LIBERO, it further improves training efficiency from 18.25% to 59.25%, and most over 30%, while for ManiSkill, it leads to a significant degradation. This discrepancy stems from ManiSkill's ability to utilize GPU-parallelization for environment computations. Our current Rollout Async implementation splits batches into smaller pieces to achieve true parallelism between the Env and Rollout workers, which can undermine the efficiency of batched environment computations in ManiSkill. Even so, we only experienced a 4.46% throughput loss in the small-scale 8-GPU setup. As the scale increases, the environmental overhead caused by manikill is offset, and at 32-GPU scale, our method achieves a 17.84% throughput improvement, validating the effectiveness of our solution for large-scale deployments.

4.2.2 Analysis of Placement Strategy

We further analyze the robustness and performance of our proposed optimization method under different resource allocation ratios between the rollout and training phases. The experimental results, presented in Figure 2, demonstrate that our method consistently outperforms the baseline asynchronous training strategy across all tested resource configurations (ratios of 1:1, 2:1, and 3:1). The performance advantage increases progressively as we introduce Train Async, Rollout Async, and finally the Streamer scheduler. Specifically, the maximum improvements in training throughput r.each 126.67% and 74.29% for the 1:1 and 3:1 allocation ratios, respectively, confirming the robustness and adaptability of our method.

To elucidate these results, we perform an ablation study of the individual asynchronous components. In the LIBERO environment, the rollout phase typically dominates the time cost. Validation experiments showed that with 6 GPUs for rollout and 2 GPUs for actor updates, the two phases are approximately balanced. Consequently, when only Train Async is employed, a 3:1 GPU allocation ratio allows for effective overlap of execution times, yielding a 58.67% throughput improvement over the baseline. Introducing Rollout Async reduces the rollout time, enabling a shift of computational resources toward the actor worker. In this setup, balanced efficiency can be achieved even with a 1:1 allocation ratio, resulting in a 118.80% improvement. Finally, incorporating the Streamer scheduler minimizes actor worker idle time, culminating in the maximum observed improvement of 126.67%.

4.2.3 Scaling with GPU Resources

We further investigate the scaling behavior of the different strategies by increasing the number of GPU resources. Figure 3 plots the throughput for the LIBERO+$\pi_{0.5}$ configuration across a range of GPU counts. In an ideal scenario, throughput would scale linearly with the number of GPUs.

Our results indicate that our method exhibits near-optimal scaling from 8 to 24 GPUs. Scaling efficiency moderates between 24 and 128 GPUs and degrades further from 128 to 256 GPUs. This sublinear scaling at high resource counts is attributed to the growing communication overhead between the increasing number of workers. Enhancing the method to achieve more ideal scaling capability at extreme scales remains an important direction for future work.

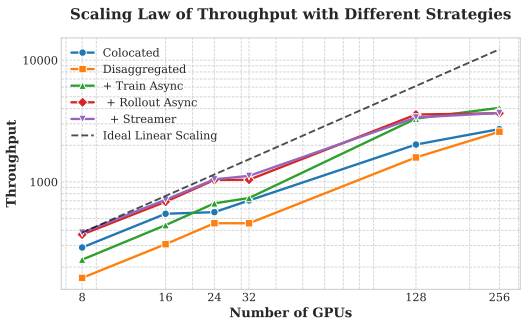


Figure 3: Scaling behavior with increasing GPU resources for the LIBERO+$\pi_{0.5}$ configuration.

5 Conclusion and Future Work

In this study, we address the shortcomings of asynchronous strategies in the post-training phase of VLA reinforcement learning by introducing a novel end-to-end asynchronous strategy involving the environment, rollout, and actor. We have validated the effectiveness of our approach through extensive experiments. In the LIBERO environment, our method achieves up to a 59.25% increase in throughput, significantly surpassing existing synchronous strategies. When exploring different decoupling strategies, our approach enhances throughput by up to 126.67% over existing decoupling methods. Additionally, we have verified our strategy's scaling law through large-scale experiments ranging from 8 to 256 GPUs, demonstrating its superior scalability in most scenarios.

There are several promising future research In future research, we aim to optimize the communication overhead in large-scale training to further enhance the optimal scalability of our method. We plan to extend AsyncVLA to a broader range of high-fidelity simulation backends, such as NVIDIA Isaac Sim and the BEHAVIOR-1K benchmark. Integrating these environments will allow us to evaluate the framework's robustness against complex physics and varied rendering overheads. Finally, we intend to explore the application of AsyncVLA in cross-embodiment learning, leveraging

its high-throughput capabilities to train generalist policies across heterogeneous robot morphologies. We hope this work provides a foundational infrastructure for the community to develop next-generation, large-scale general robotic intelligence.

REFERENCES

Arthur Allshire, Hongsuk Choi, Junyi Zhang, David McAllister, Anthony Zhang, Chung Min Kim, Trevor Darrell, Pieter Abbeel, Jitendra Malik, and Angjoo Kanazawa. Visual imitation enables contextual humanoid control, 2025. URL https://arxiv.org/abs/2505.03729.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. https://github.com/huggingface/lerobot, 2024.

Mustafa Shukor Dana Aubakirova, Jade Cholgari, and Leandro von Werra. Vlab: Your laboratory for pretraining vlas. https://github.com/huggingface/vlab, 2025.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, et al. Asyncflow: An asynchronous streaming rl framework for efficient llm post-training. *arXiv preprint arXiv:2507.01663*, 2025.

Kento Kawaharazuka, Jihoon Oh, Jun Yamada, Ingmar Posner, and Yuke Zhu. Vision-language-action models for robotics: A review towards real-world applications. *IEEE Access*, 13: 162467–162504, 2025. ISSN 2169-3536. doi: 10.1109/access.2025.3609980. URL http://dx.doi.org/10.1109/ACCESS.2025.3609980.

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

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

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. *arXiv preprint arXiv:2509.09674*, 2025.

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.

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

T Mu, Z Ling, F Xiang, D Yang, X Li, S Tao, Z Huang, Z Jia, and H Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

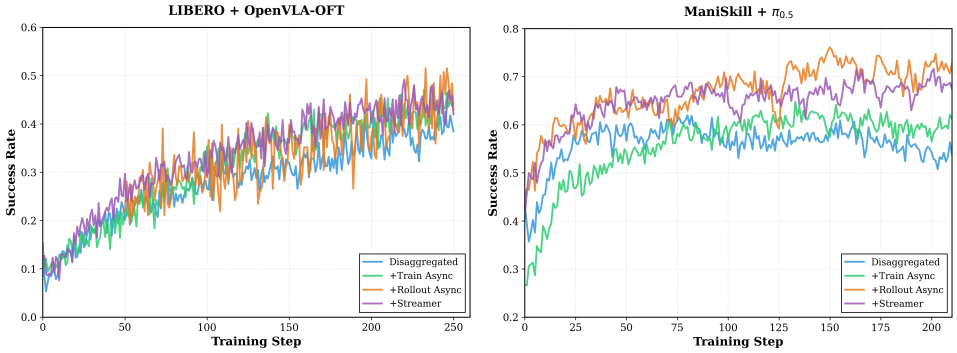


Figure 4: Training success rate curves for LIBERO+OpenVLA-OFT and ManiSkill+$\pi_{0.5}$ under different training strategies. Our asynchronous training pipeline shows a steady growth in success rate, matching the trend of baseline methods.

Guangming Sheng, Yuxuan Tong, Borui Wan, Wang Zhang, Chaobo Jia, Xibin Wu, Yuqi Wu, Xiang Li, Chi Zhang, Yanghua Peng, et al. Laminar: A scalable asynchronous rl post-training framework. *arXiv preprint arXiv:2510.12633*, 2025.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Yuqi Xie, Fengyuan Hu, Jimmy Wu, Zhengyi Luo, Linxi "Jim" Fan, Guanya Shi, and Yuke Zhu. Self-improving vision-language-action models with data generation via residual rl, 2025. URL https://arxiv.org/abs/2511.00091.

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Liangzhi Shi, Yuqing Xie, Zhexuan Xu, et al. Rlinf-vla: A unified and efficient framework for vla+ rl training. *arXiv preprint arXiv:2510.06710*, 2025.

Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges, 2023. URL https://arxiv.org/abs/2309.02473.

Chenyue Zhou, Mingxuan Wang, Yanbiao Ma, Chenxu Wu, Wanyi Chen, Zhe Qian, Xinyu Liu, Yiwei Zhang, Junhao Wang, Hengbo Xu, Fei Luo, Xiaohua Chen, Xiaoshuai Hao, Hehan Li, Andi Zhang, Wenxuan Wang, Kaiyan Zhang, Guoli Jia, Lingling Li, Zhiwu Lu, Yang Lu, and Yike Guo. From perception to cognition: A survey of vision-language interactive reasoning in multimodal large language models, 2025. URL https://arxiv.org/abs/2509.25373.

A Implementation Details

In this section, we provide detailed hyperparameters used in our experiments. To ensure fair comparison between different strategies, we keep most hyperparameters consistent across runs for the same environment and model. Table 2 lists critical hyperparameters for single-node training (8 GPUs). When GPU resources increase, we scale $\mathrm{global_batch_size}$ and $\mathrm{total_env}$ accordingly while keeping other parameters unchanged. Hyperparameters related to the optimizer and learning algorithm—which do not affect throughput—are omitted from the table for brevity. These can be found in our code, which will be made publicly available.

Table 2: Detailed hyperparameters for different experimental setups under single GPU node training.

Parameter	Setup A	Setup B	Setup C	Setup D	Setup E	Setup F
Base Configuration						
Dataset	LIBERO	LIBERO	LIBERO	LIBERO	ManiSkill	ManiSkill
Model	π_0	OpenVLA-OFT	$\pi_{0.5}$	GR00T N1.5	π_0	OpenVLA-OFT
Environment Settings						
Total Env/Node	128	128	64	64	256	256
Total Env/Node (3:1)	192	192	192	192	768	768
Max Episode Steps	480	512	480	480	80	80
Training Hyperparameters						
Rollout Epoch	4	4	8	8	4	4
Update Epoch	4	4	1	1	4	4
Micro Batch Size	128	32	128	128	64	32
Global Batch Size/Node	2048	16384	2048	1024	4096	2048
Model Specifications						
Model Num Step	4	4	4	4	4	4
Chunk Size	10	8	10	10	5	8

B Further Experimental Results

B.1 Success Rate

Figure 4 shows the training curves for LIBERO + OpenVLA-OFT and ManiSkill + $\pi_{0.5}$ under different training strategies. As shown, the success rate under our asynchronous training pipeline grows steadily and follows a trend similar to that of the baseline strategies. Together with the analysis in Section 4, these results demonstrate that our proposed methods significantly improve training efficiency with minimal impact on final performance.

B.2 Additional Throughput Results

Beyond the main results in Section 4, we conducted extensive throughput evaluations. Table 3 provides comprehensive results for baseline methods, including Colocated, Hybrid, and Disaggregated placement strategies, across GPU scales from 8 to 32 GPUs. We also performed extensive experiments with four different optimization strategy combinations, exploring various placement ratios for rollout ($\mathrm{Rollout}$ and Env workers) and training (Actor workers) under different resource scales. We hope these results will support future research in developing efficient training algorithms and infrastructure for vision-language-action models.

Table 3: Baseline performance across different models and placement strategies on LIBERO.

Configuration	Throughput	Time (s)	Rollout (s)	Actor (s)
π_0 Model				
Colocated				
8 GPUs	304.54	807.4	577.4	228.6
16 GPUs	591.35	831.19	591.0	240.25
32 GPUs	1085.51	905.6	672.4	227.8
Hybrid				
8 GPUs	206.52	1190.0	1068.0	117.0
16 GPUs	358.56	1370.83	1195.0	126.25
32 GPUs	683.62	1438.0	1318.0	116.6
Disaggregated				
1:1 - 8 GPUs (up=2)	196.63	1249.8	1207.8	222.0
1:1 - 8 GPUs	128.54	1911.8	1050.0	861.0
1:1 - 8 GPUs (T+R)	87.43	2810.0	1950.0	860.0
1:1 - 16 GPUs	326.8	1504.0	1058.0	442.8
1:1 - 32 GPUs	624.0	1574.0	1126.0	441.6
3:1 - 8 GPUs	96.67	2542.0	1114.0	1428.0
3:1 - 8 GPUs (T+R)	71.9	3418.0	2024.0	1394.0
3:1 - 16 GPUs	309.52	2382.0	1092.0	1286.0
3:1 - 32 GPUs	583.29	2528.0	1214.0	1308.0
2:1 - 6 GPUs	195.0	1884.4	1025.4	859.0
2:1 - 6 GPUs (T+R)	87.44	4216.0	2930.0	1286.0
2:1 - 24 GPUs	459.36	2140	1240	892
OpenVLA-OFT Model				
Colocated				
8 GPUs	260.58	1006.0	674.6	312.6
16 GPUs	466.03	1125.0	767.83	352.17
32 GPUs	784.86	1336.0	965.0	353.6
Hybrid				
8 GPUs	161.02	1628.0	1298.0	312.4
16 GPUs	275.22	1905.0	1543.33	355.17
Disaggregated				
1:1 - 8 GPUs	144.35	1816.0	1210.0	589.8
1:1 - 16 GPUs	248.01	2114.0	1288.0	807.8
1:1 - 32 GPUs	466.73	2246.67	1500.0	707.33
3:1 - 8 GPUs	118.22	3326.0	1450.0	1876.0
3:1 - 16 GPUs	245.15	3208.0	1354.0	1838.0
3:1 - 32 GPUs	425.48	3696.67	1706.67	1940.0
2:1 - 6 GPUs	110.98	2362.0	1236.0	1106.0
2:1 - 6 GPUs (T+R)	75.04	5240.0	3570.0	1670.0
2:1 - 24 GPUs	384.37	2728.0	1440.0	1288.0

Note: "T+R" denotes training and environment disaggregated in different GPUs. "up=2" indicates update epoch = 2.

Table 4: Performance comparison with different optimization strategies on LIBERO.

	π_0 Model				OpenVLA-OFT Model			
Configuration	**Thr.**	**Time**	**Roll.**	**Actor**	**Thr.**	**Time**	**Roll.**	**Actor**
Optimization #1: Async Training								
1:1 - 8 GPUs	200.95	1223.0	1088.0	506.5	189.21	1385.5	1225.0	616.5
1:1 - 16 GPUs	431.16	1140.0	1030.0	439.0	397.19	1320.0	–	–
1:1 - 32 GPUs	854.82	1150.0	1030.0	438.0	748.98	1400.0	–	–
3:1 - 8 GPUs	231.12	1595.0	1005.0	1300.0	194.47	2022.0	1040.0	1660.0
3:1 - 16 GPUs	451.21	1634.0	1028.0	1302.0	342.82	2294.0	1064.0	1840.0
3:1 - 32 GPUs	857.30	1720.0	1110.0	1310.0	604.95	2600.0	–	–
2:1 - 6 GPUs	156.20	1180.0	963.2	858.2	137.49	1430.0	1010.0	1128.0
2:1 - 24 GPUs	582.37	1266.0	1040.0	876.8	434.49	1810.0	1072.0	1288.0
Optimization #2: Async Parallel Rollout								
1:1 - 8 GPUs	223.80	1098.1	611.6	485.2	202.12	1297.0	666.4	612.0
3:1 - 8 GPUs	196.50	1876.0	605.2	1266.0	164.39	2392.0	713.2	1658.0
3:1 - 16 GPUs	379.26	1944.0	657.4	1280.0	293.23	2682.0	834.0	1830.0
2:1 - 6 GPUs	130.35	1414.0	565.0	849.4	110.92	1772.6	646.6	1126.0
2:1 - 24 GPUs	452.93	1627.8	736.0	891.0	345.23	2278.0	984.4	1294.0
Optimization #3: Async Training + Async Parallel Rollout								
1:1 - 8 GPUs	352.80	696.6	586.2	446.0	308.04	851.0	627.8	630.6
1:1 - 16 GPUs	687.44	715.0	–	–	567.41	924.0	–	–
1:1 - 32 GPUs	1365.33	720.0	–	–	953.25	1100.0	–	–
3:1 - 8 GPUs	245.43	1502.0	562.0	1302.0	204.37	1924.0	602.0	1664.0
3:1 - 16 GPUs	481.25	1532.0	590.2	1296.0	364.43	2158.0	634.8	1838.0
3:1 - 32 GPUs	899.12	1640.0	–	–	604.95	2600.0	–	–
2:1 - 6 GPUs	176.55	1044.0	539.0	869.0	148.27	1326.0	569.0	1128.0
2:1 - 24 GPUs	644.48	1144.0	601.0	876.0	456.17	1724.0	656.0	1302.0
Optimization #4: All (Async Training + Async Parallel Rollout + Stream)								
1:1 - 8 GPUs	397.78	617.8	536.8	441.3	307.68	852.0	640.0	624.0
1:1 - 16 GPUs	712.35	690.0	–	–	579.32	905.0	–	–
1:1 - 32 GPUs	1414.45	695.0	–	–	979.98	1070.0	–	–
3:1 - 8 GPUs	253.88	1452.0	563.8	1290.0	205.66	1912.0	607.4	1664.0
3:1 - 16 GPUs	492.83	1496.0	587.2	1300.0	363.08	2166.0	639.6	1854.0
3:1 - 32 GPUs	933.27	1580.0	–	–	629.15	2500.0	–	–
2:1 - 6 GPUs	179.65	1026.0	537.0	866.0	149.70	1313.3	567.0	1120.0
2:1 - 24 GPUs	658.29	1120.0	596.0	873.0	470.64	1671.0	639.0	1305.0

Note: "Thr." = Throughput, "Roll." = Rollout (s), "Actor" = Actor (s), "Time" = Time (s). "–" indicates missing data.