QCon
全球软件开发大会

InfoQ 极客传媒

# RLinf: A Scalable, Flexible and Large-Scale Reinforcement Learning Framework for Embodied Intelligence


Chao Yu (于超)

Tsinghua University

zoeyuchao@gmail.com

# 目录

InfoQ 极客传媒

北京

QCon 全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

北京

AiCon 全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

上海

QCon 全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月 — 5月 — 6月 — 8月 — 10月 — 12月

上海

AiCon 全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

深圳

AiCon 全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

北京

AiCon 全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

# 极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

# 01

# Motivation

# Embodied Intelligence

Building models that can pass the embodied Turing test will provide a roadmap for the next generation of AI.

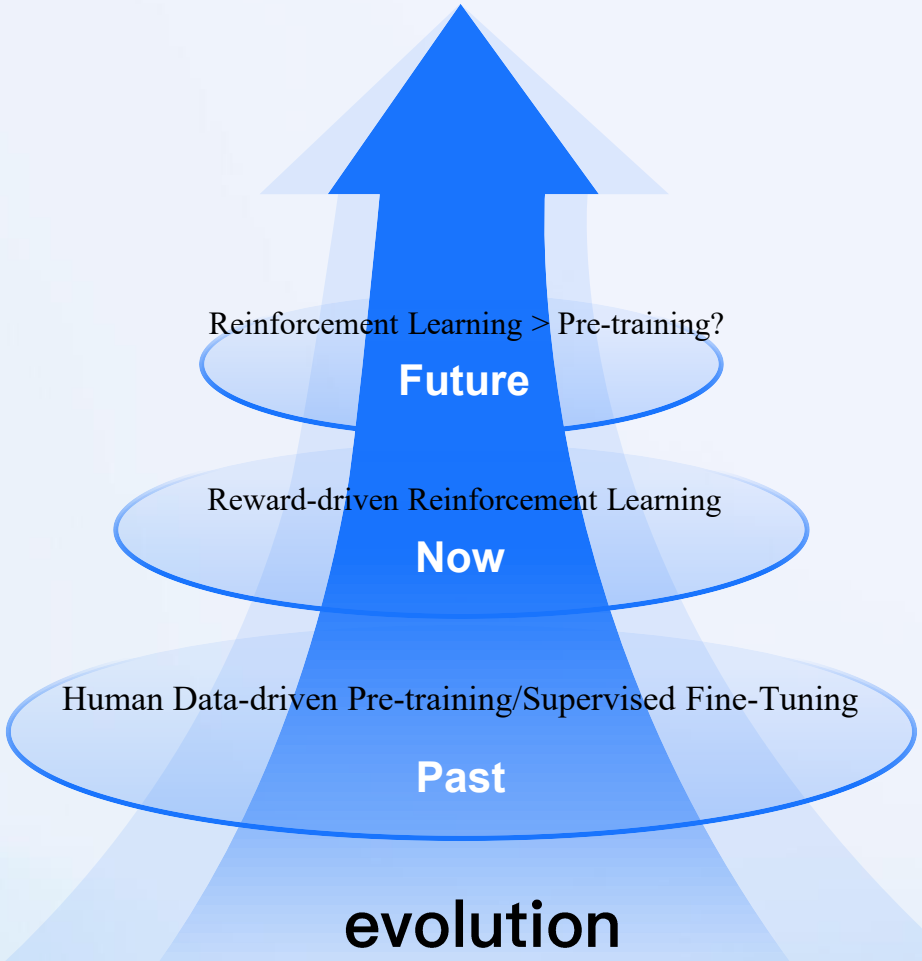
—— *Yann Lecun*

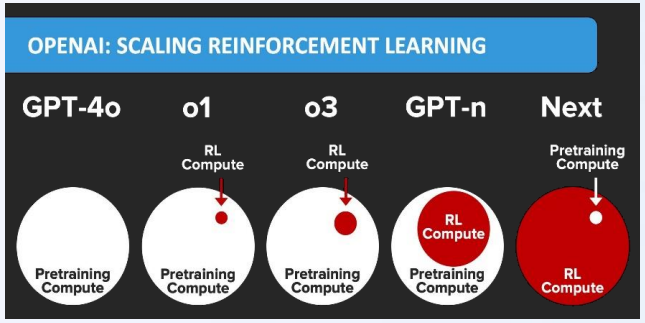


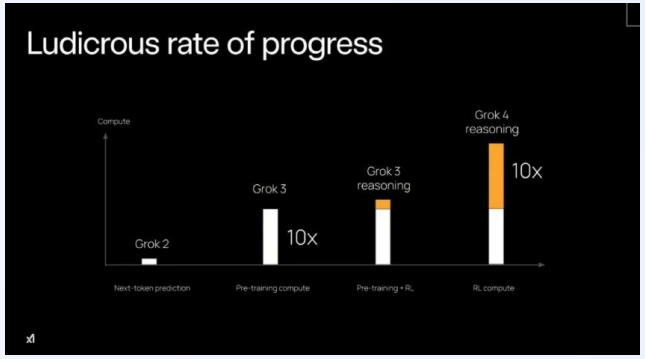

**Robotic Foundation Model**

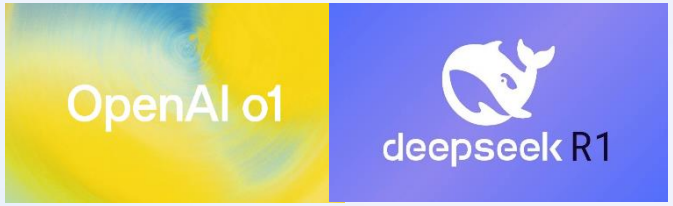
leverage the general ability of large models and the special features of robot–specific data

# Reinforcement Learning

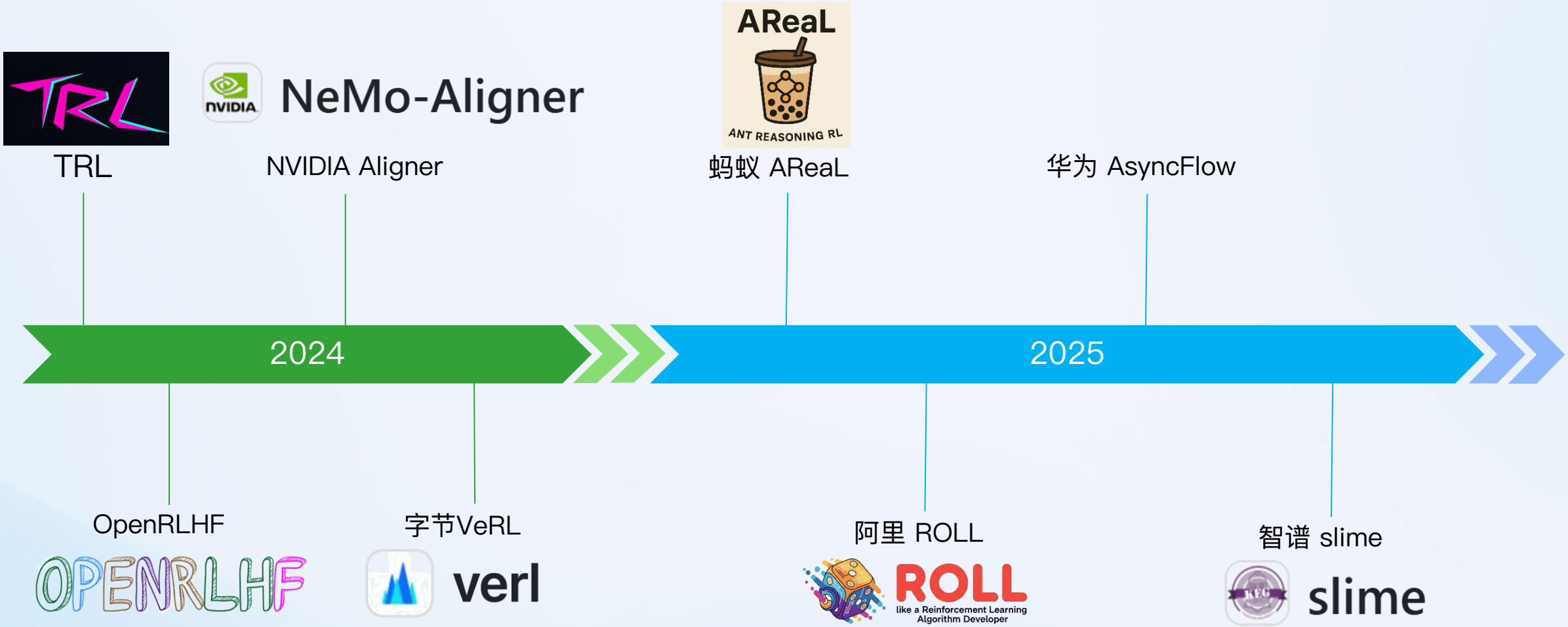


# RL Framework for Large Reasoning Model



# RL framework for VLA? None!

Large Reasoning Model

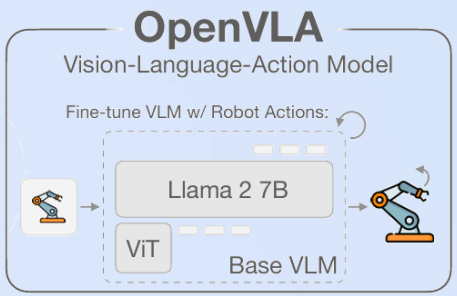
Cerebrum

Robotic Foundation Model

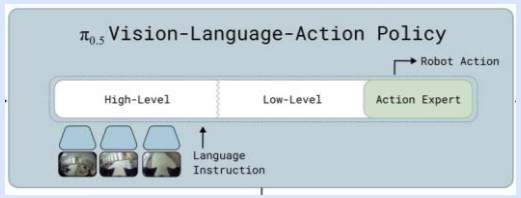
Cerebrum + Cerebellum

RoboBrain

RoboBrain




OpenVLA




Pi–0.5

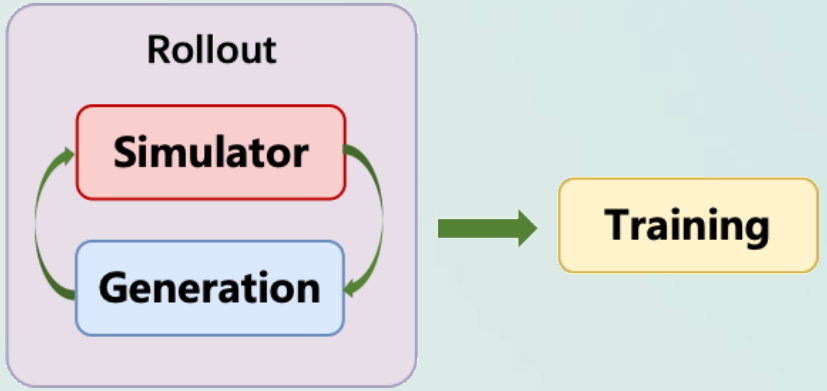
# RL framework for VLA? None!

Large Reasoning Model

One-step Decision

Robotic Foundation Model

Multi-step Decision



# RL framework for VLA? None!

Large Reasoning Model

Robotic Foundation Model

Dataset

GPU-based Simulator

NVIDIA Isaac Sim

GENESIS

GENESIS

ManiSkill

ManiSkill

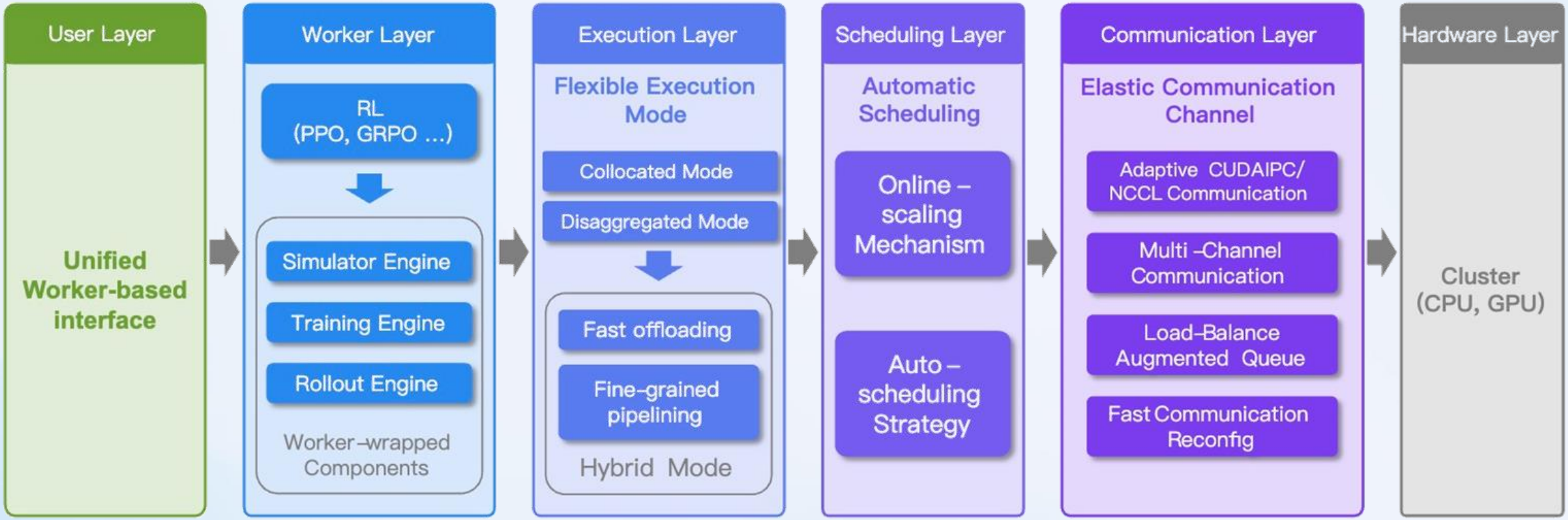
Realistic 3D Rendering

Efficient Physical Simulation

02

# System-level Design

# RLinf: An Open-source RL Framework for Embodied Intelligence




[1] Chao Yu et al., RLinf: Flexible and Efficient Large-scale Reinforcement Learning via Macro-to-Micro Flow Transformation. https://arxiv.org/abs/2509.15965

# User Layer: Macro workflow construction with RLinf

## ❑ Typical reasoning RL workflow

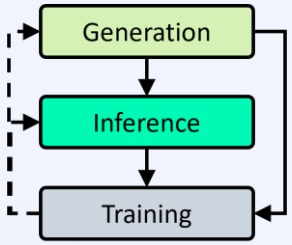
```
def run(self, batch_iterator):
    for batch in batch_iterator:
        self._update_rollout_weights()

        self.data_ch.put(batch)

        self.rollout_group.generate(
            in_channel=self.data_ch,
            out_channel=self.rollout_ch,
        )

        self.inference_group.inference(
            in_channel=self.rollout_ch,
            out_channel=self.inference_ch
        )

        self.actor_group.train(
            in_channel=self.inference_ch
        ).wait()
```



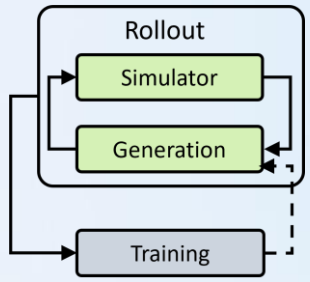

## ❑ Typical embodied RL workflow

```
def run(self, num_iters, num_steps):
    for _ in range(num_iters):
        self._update_rollout_weights()
        self.observation_ch.put(None)

        for _ in range(num_steps):
            self.rollout_group.generate(
                in_channel=self.observation_ch,
                out_channel=self.action_ch,
                train_channel=self.train_ch,
            )

            handle = self.env_group.step(
                in_channel=self.action_ch,
                out_channel=self.observation_ch,
                train_channel=self.train_ch,
            )

        handle.wait()
        self.actor_group.train(self.train_ch).wait()
```




Data flow expressed via distributed data channels

*self.rollout_group* Collective management knob of component processes

*handle.wait()* Synchronization point in the inherently async workflow

# Worker Layer: *Worker* and *WorkerGroup* construction

- Easy *Worker* main logic construction

```
class RolloutWorker(Worker):
    def generate(self, in_channel: Channel, out_channel: Channel):
        with out_channel.device_lock:
            batch_size = 0
            while batch_size < self.total_batch_size:
                batch = in_channel.get()
                res = self.model.gen(batch)
                out_channel.put(res)
                batch_size += batch.size
```

Resource protection

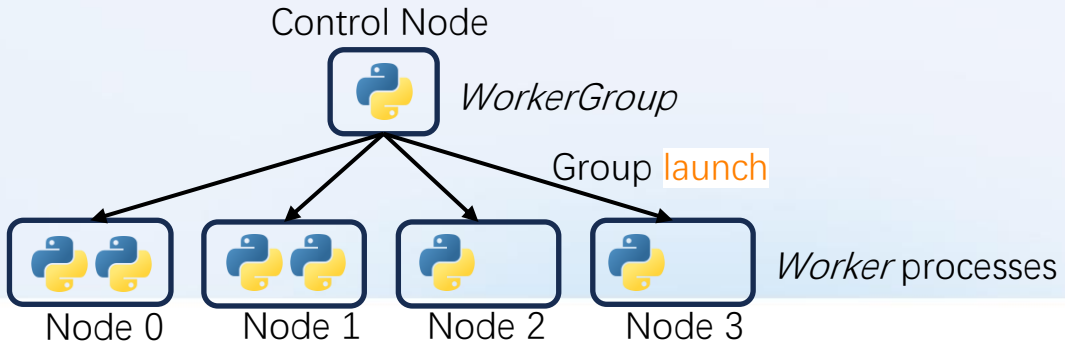
1. Get input data
2. Operate the data
3. Put output data

Almost all components can be modeled in this I/O pattern

- Collective *Worker* process management with *WorkerGroup*

```
class RLWorkflowRunner:
    def __init__(self, placement):
        cluster = Cluster(num_nodes=4)
        self.rollout_group = RolloutWorker.create_group().launch(cluster, placement)
```

Remotely launch worker processes in the cluster



# Worker Layer: *Worker* and *WorkerGroup* construction

- Easy *Worker* main logic construction

```
class RolloutWorker(Worker):
    def generate(self, in_channel: Channel, out_channel: Channel):
        with out_channel.device_lock:
            batch_size = 0
            while batch_size < self.total_batch_size:
                batch = in_channel.get()
                res = self.model.gen(batch)
                out_channel.put(res)
                batch_size += batch.size
```

Resource protection

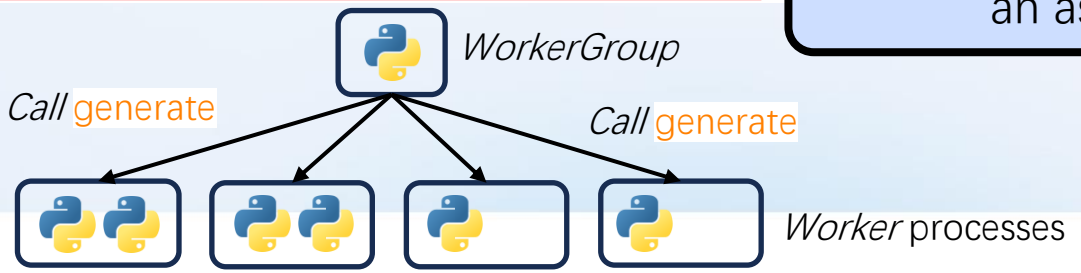
1. Get input data
2. Operate the data
3. Put output data

Almost all components can be modeled in this I/O pattern

- Collective *Worker* process management with *WorkerGroup*

```
class RLWorkflowRunner:
    def __init__(self, placement):
        cluster = Cluster(num_nodes=4)
        self.rollout_group = RolloutWorker.create_group().launch(cluster, placement)
        handle = self.rollout_group.generate(...)
        handle.wait() # [ret_val_rank_0, ret_val_rank_1, ...]
```

Execute worker functions via *WorkerGroup* in an async and SPMD manner



# Worker Layer: *Worker* and *WorkerGroup* construction

- Easy *Worker* main logic construction

```
class RolloutWorker(Worker):
    def generate(self, in_channel: Channel, out_channel: Channel):
        with out_channel.device_lock:  Resource protection
            batch_size = 0
            while batch_size < self.total_batch_size:
                batch = in_channel.get()        → 1. Get input data
                res = self.model.gen(batch)     → 2. Operate the data
                out_channel.put(res)            → 3. Put output data
                batch_size += batch.size
```

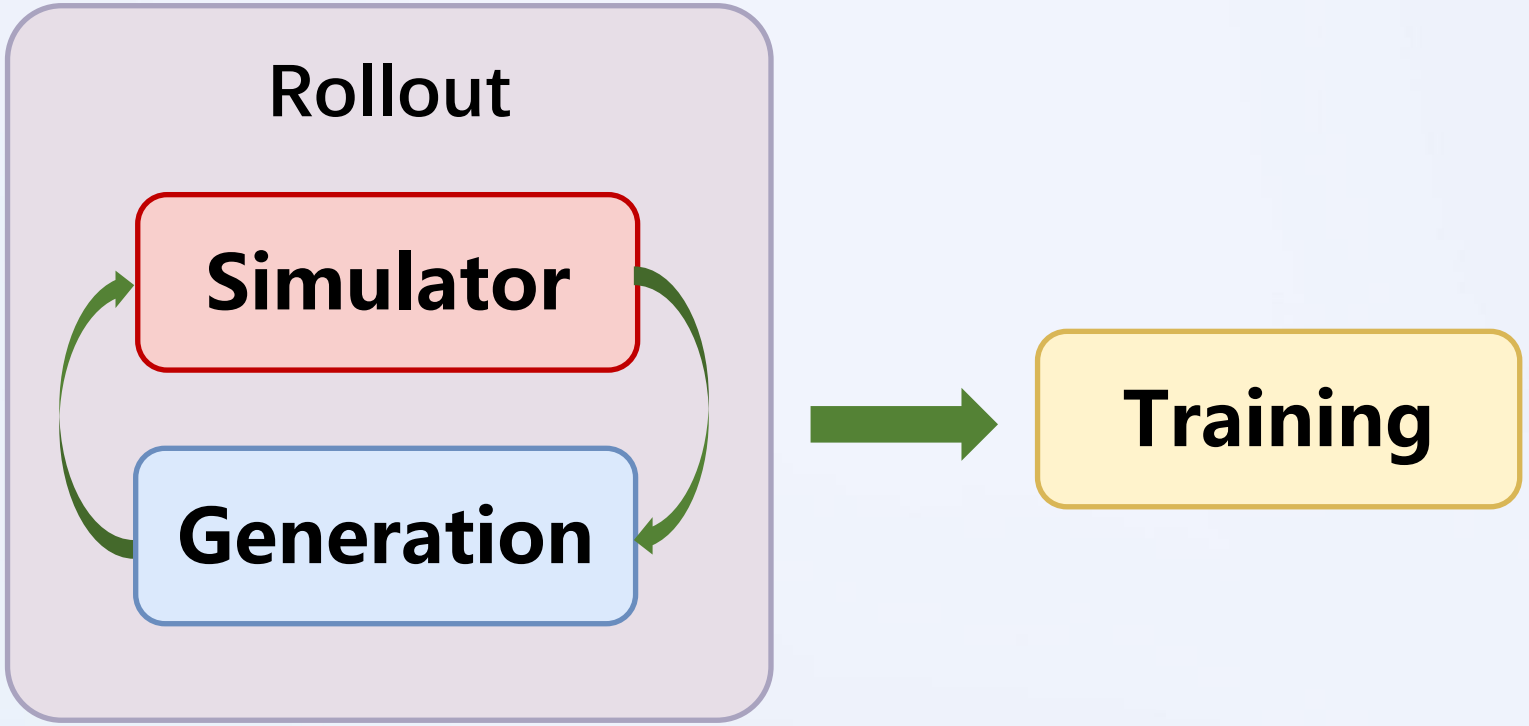
Almost all components can be modeled in this I/O pattern

- Collective *Worker* process management with *WorkerGroup*

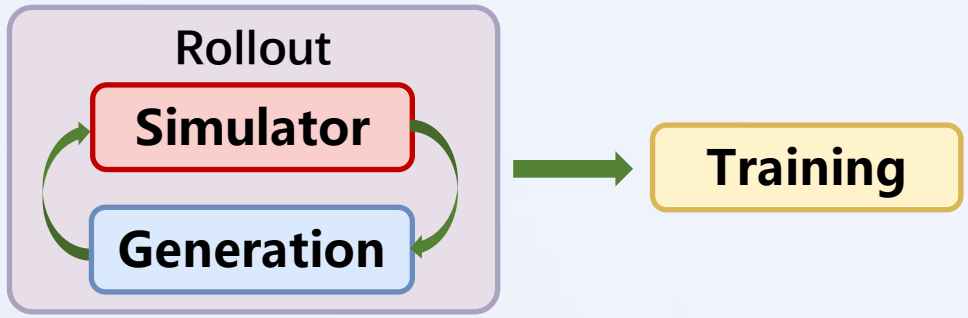
```
class RLWorkflowRunner:
    def __init__(self, placement):
        cluster = Cluster(num_nodes=4)
        self.rollout_group = RolloutWorker.create_group().launch(cluster, placement)
        handle = self.rollout_group.generate(...)
        handle.wait() # [ret_val_rank_0, ret_val_rank_1, ...]
```

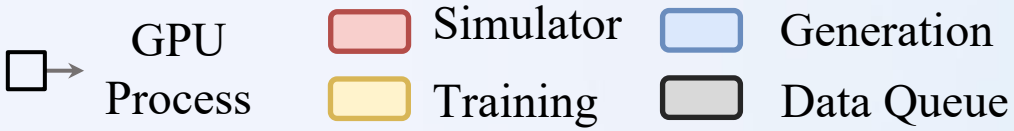
- Flexible placement and heterogenous cluster support
  - Each worker process can be assigned **any accelerator(s) of any node** across cluster
  - Accelerator can be **heterogeneous** (e.g., mixing CPU, GPU and NPU nodes) across cluster

# Execution Layer



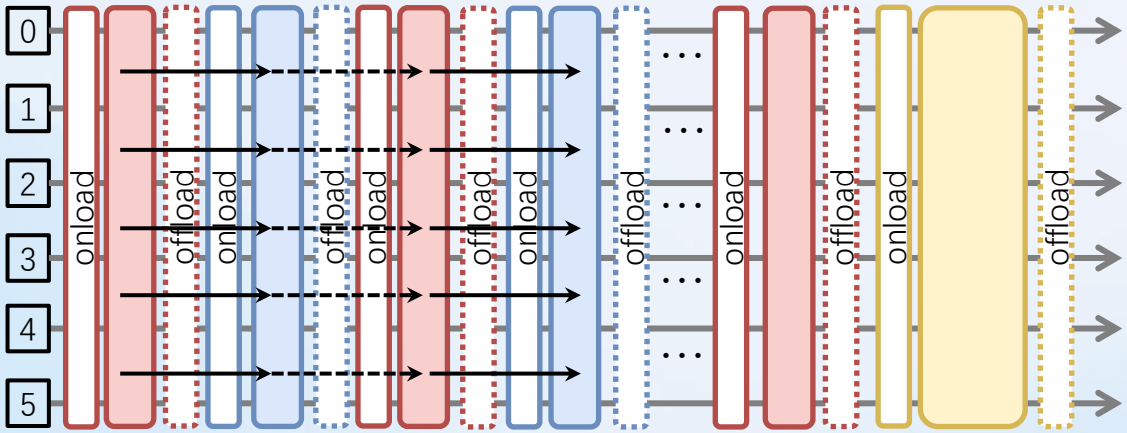
# GPU allocation Strategy: Colocated Mode



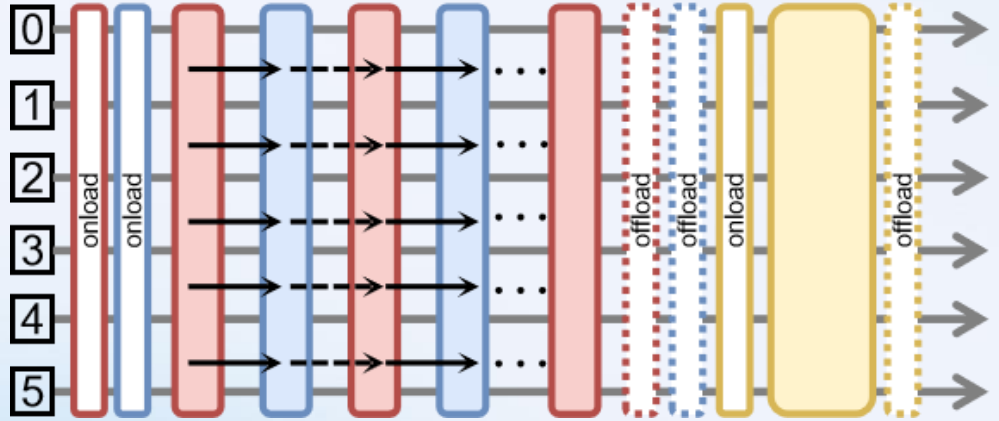

## ➢ Naive Version

- ➢ Frequent on/offload during Rollout
- ➢ Large overhead

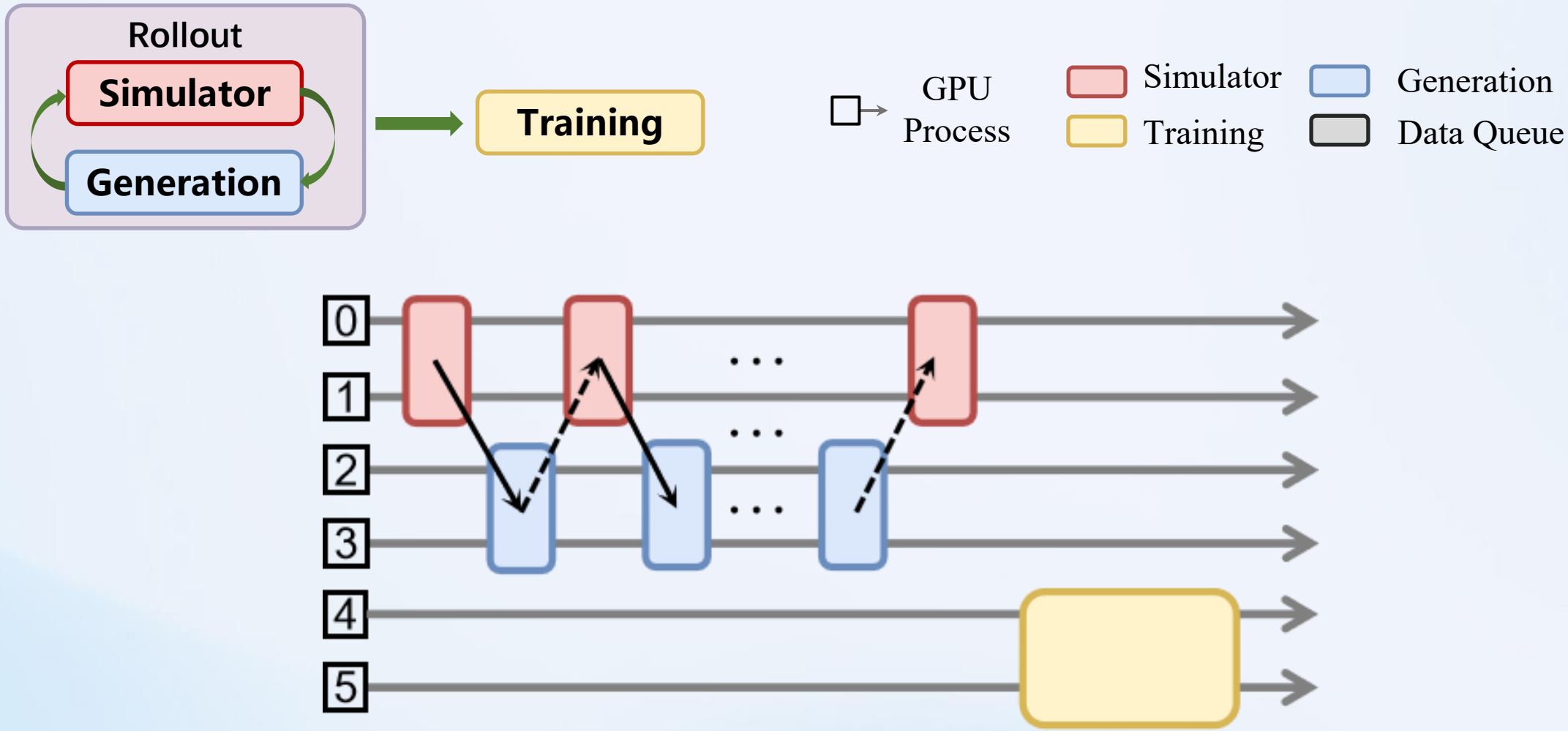



## ➢ Version in RLinf

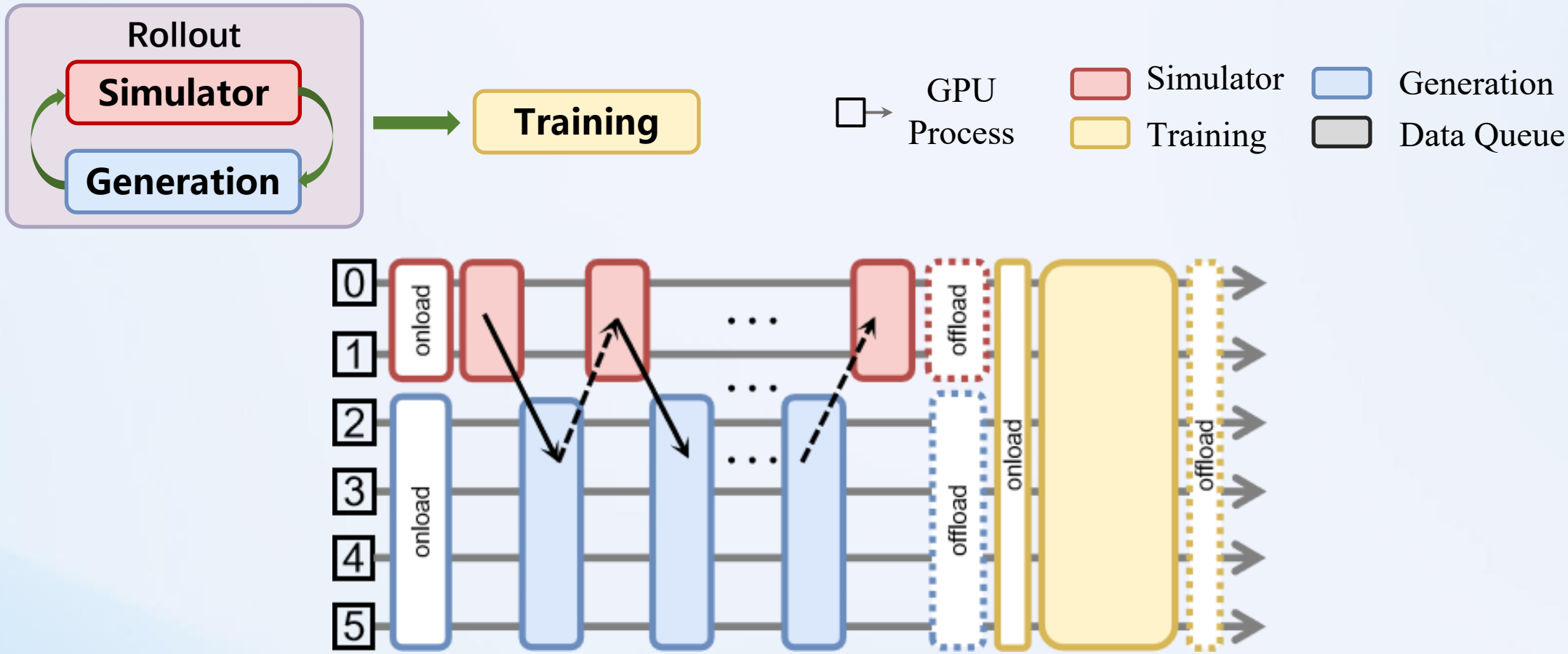
- ➢ No on/offload during Rollout
- ➢ Simulator and Generation share GPUs



# GPU allocation Strategy: Disaggregated Mode

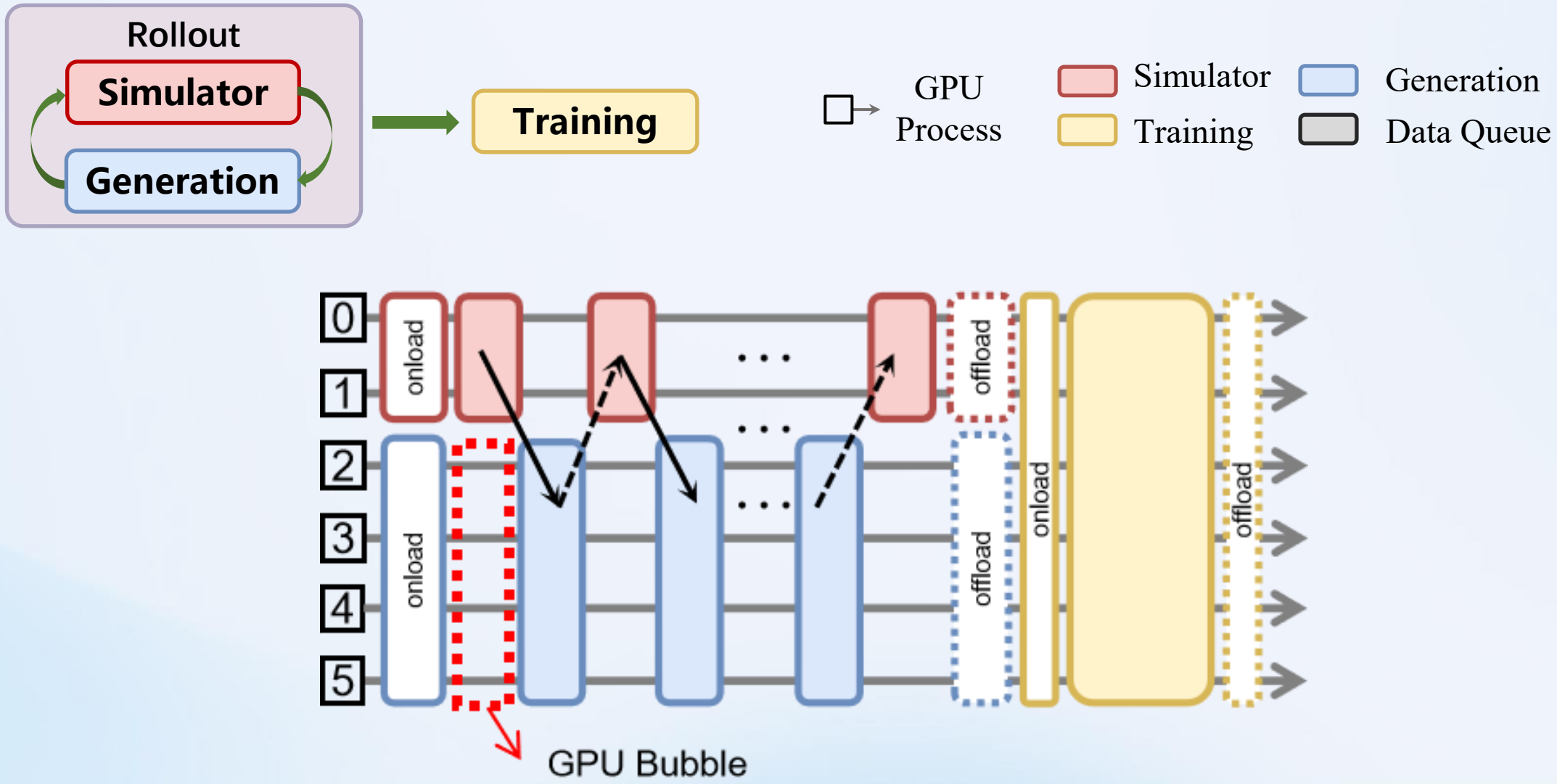


# GPU allocation Strategy: Hybrid Mode

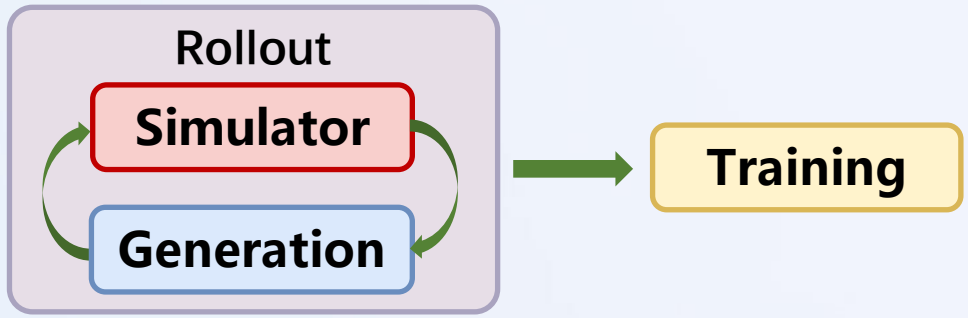



- Note: The hybrid mode can be much more flexible. The figure is just an illustration for one special case.

# GPU allocation Strategy: Hybrid Mode



# GPU allocation Strategy: Fine-grained Pipelining




Training

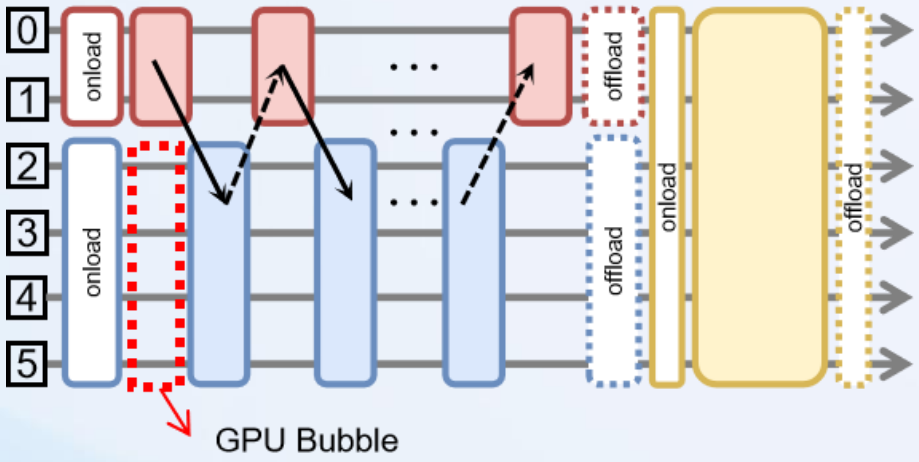
GPU Process

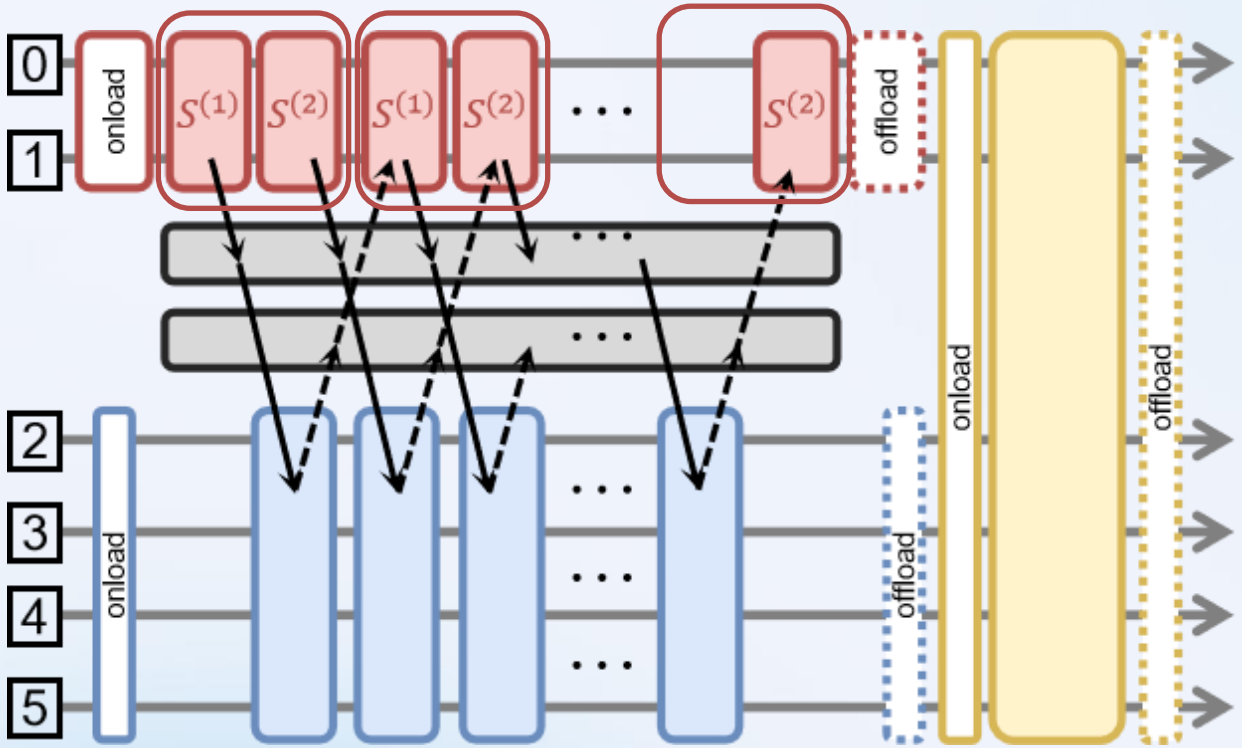
Simulator

Training

Generation

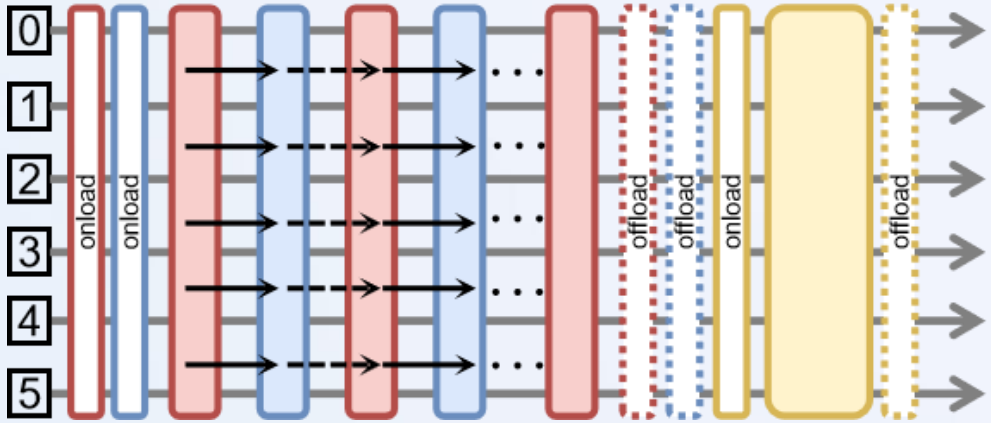
Data Queue



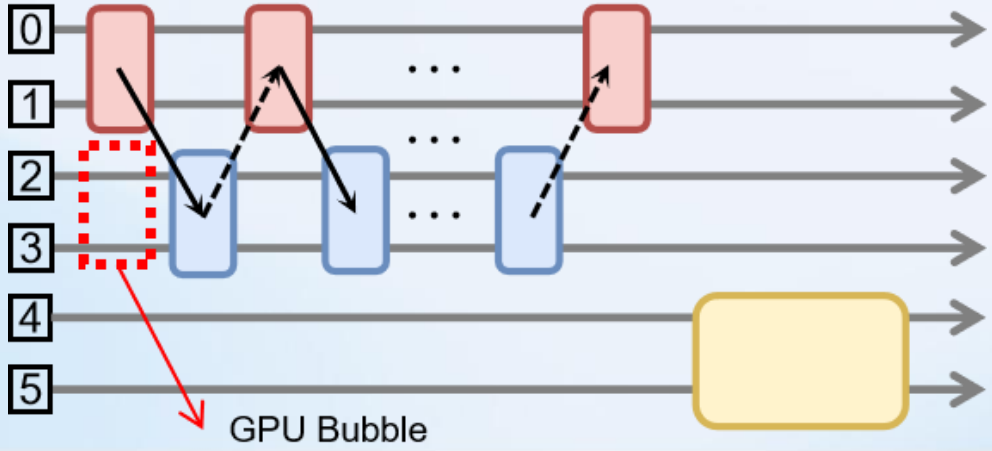



# Execution Layer: Various GPU allocation modes

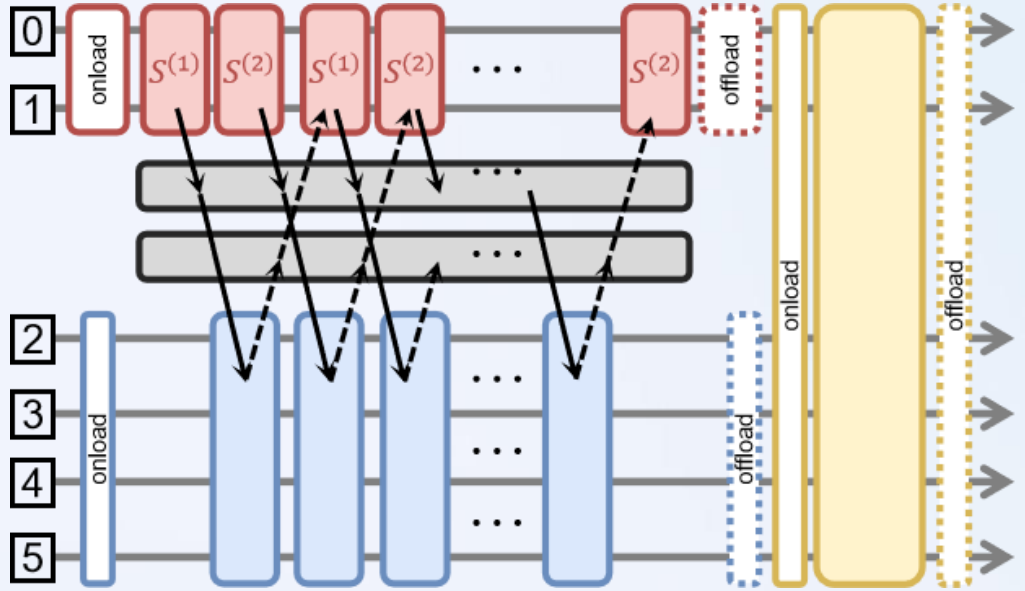
## ➢ Colocated Mode




## ➢ Disaggregated Mode




## ➢ Hybrid Mode




## ➢ Simple Configuration

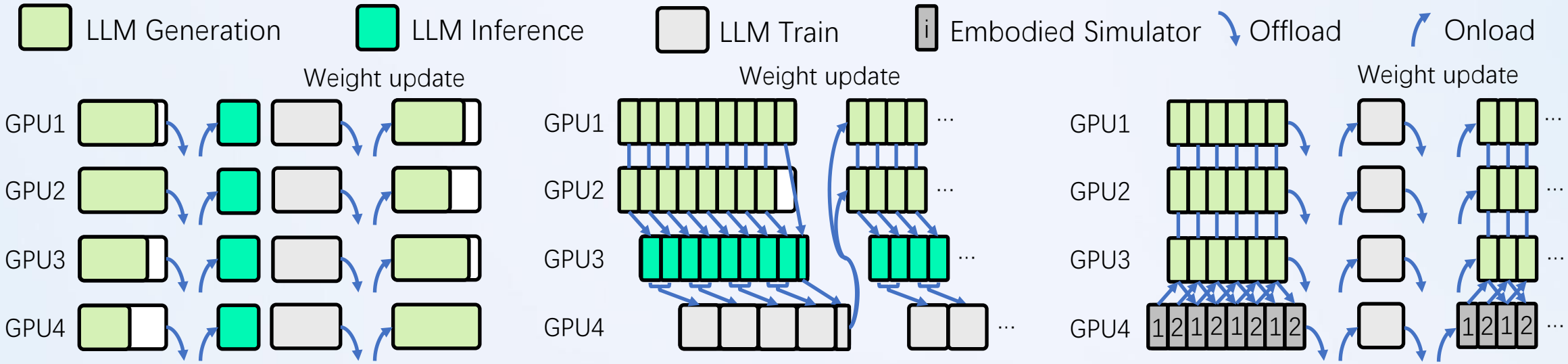
```
enable_offload: False
```

```
pipeline_stage_num: 2
```

```
cluster:
  num_nodes: 1
  component_placement:
    actor: 0-7
    env: 0-3
    rollout: 4-7
```

# Execution Layer: M2Flow transformation

❑ How to achieve fine-grained scheduling with ***the same workflow code***?




Temporal Scheduling
via **Auto Context Switching**

Temporal multiplexing by putting workers on the same devices

Spatial Scheduling
via **Elastic Pipelining**

Spatially disjoint workers form pipelines with varying data granularity

**Hybrid** Spatial and Temporal Scheduling

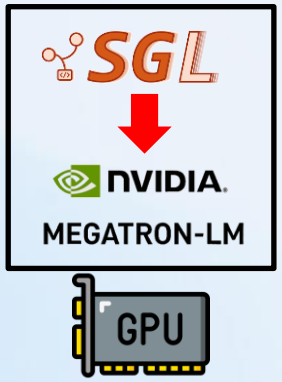
Arbitrary combination of spatial and temporal scheduling of workers

# M2Flow transformation: automatic context switching

❑ **Global device lock** for managing concurrent accelerator resource accesses

```
class RolloutWorker(Worker):
    def generate(self, in_channel: Channel, out_channel: Channel):
        with out_channel.device_lock:
            batch_size = 0
            while batch_size < self.total_batch_size:
```

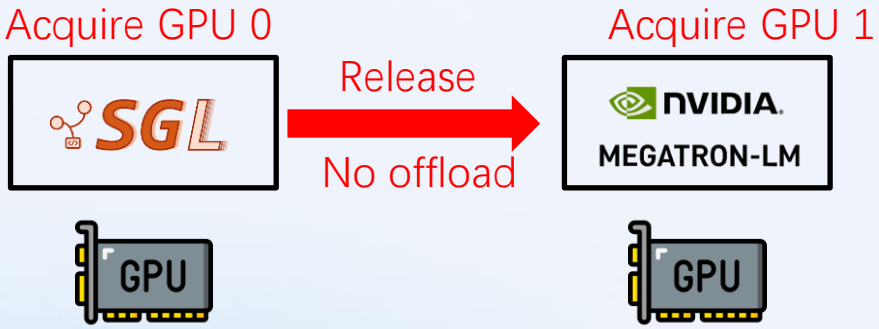
- A distributed lock primitive for acquiring and releasing accelerators
- Workers should implement their **onload** and **offload** interfaces
- A **priority lock** based on the data flow direction of the associated channel




Acquire GPU 0 with priority

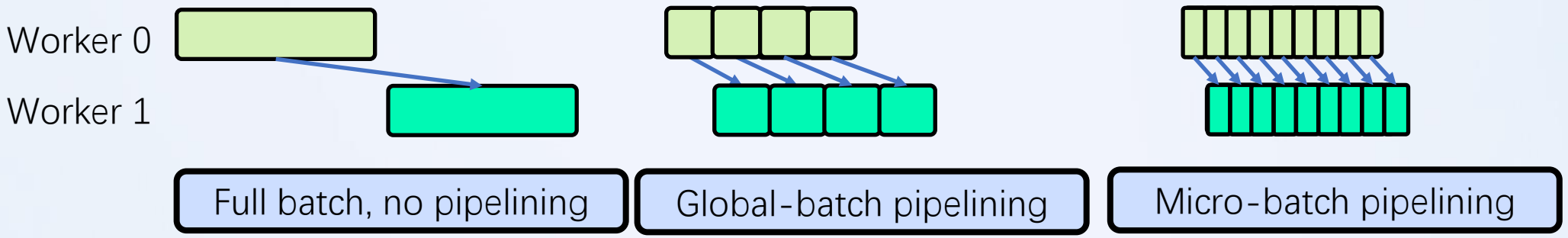
Release & offload

Acquire GPU 0



# M2Flow transformation: elastic pipelining

- Data processing granularity as the pipelining granularity




- The flow-like interface naturally enables pipelining at varying data granularity

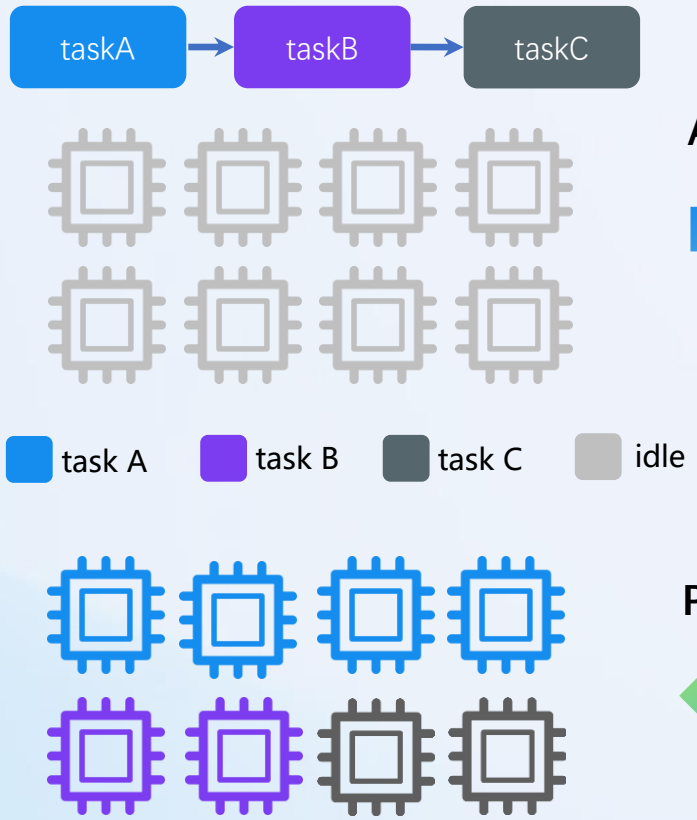
```
prompts = [
    "Write a short, neutral self-introduction for a fictional character. Hello, my name is",
    "Provide a concise factual statement about France’s capital city. The capital of France is",
    "Explain possible future trends in artificial intelligence. The future of AI is",
]
llm.async_generate(prompts, sampling_params)
```

SGLang’s support for batch generation

- **Performance factor**: larger batch can better utilize accelerators, but leads to more bubbles
- **Semantic factor**: algorithm semantic like batch-wise normalization requires larger batch

# Scheduling Layer

## Offline Automated Scheduling




**Analyzer**

performance analysis:

(Second / iter)

task A: 60

task B: 30

task C: 30

**Scheduler**

Single-node, 8GPUs:

task A: 0-3

task B: 4-5

task C: 6-7

**Physical Placement**

### Analyzer

Automated performance analysis of usage efficiency and characteristics

Small–scale GPU cluster

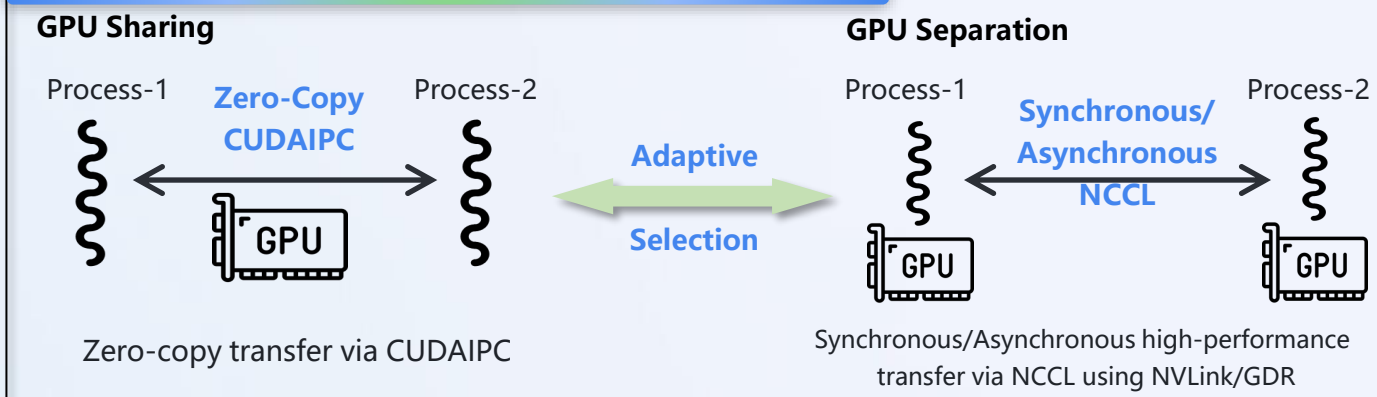
### Scheduler

Offline calculation of the optimal placement strategy using dynamic programming

GPU cluster of any scale

# Communication Layer

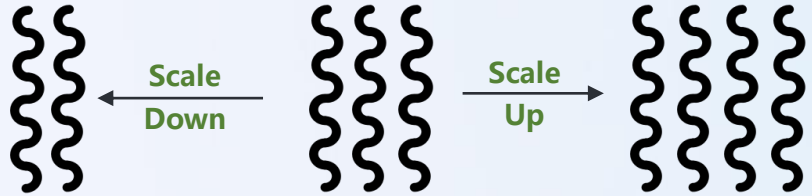
## Adaptive CUDAIPC/NCCL Communication




Zero-copy transfer via CUDAIPC

Synchronous/Asynchronous high-performance transfer via NCCL using NVLink/GDR

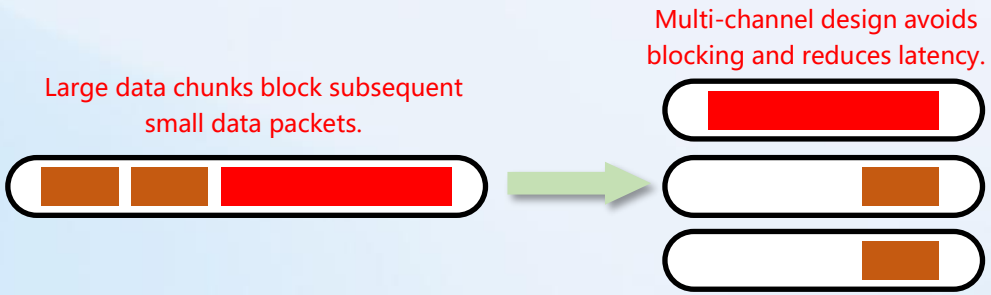
## Fast Communication Reconfiguration



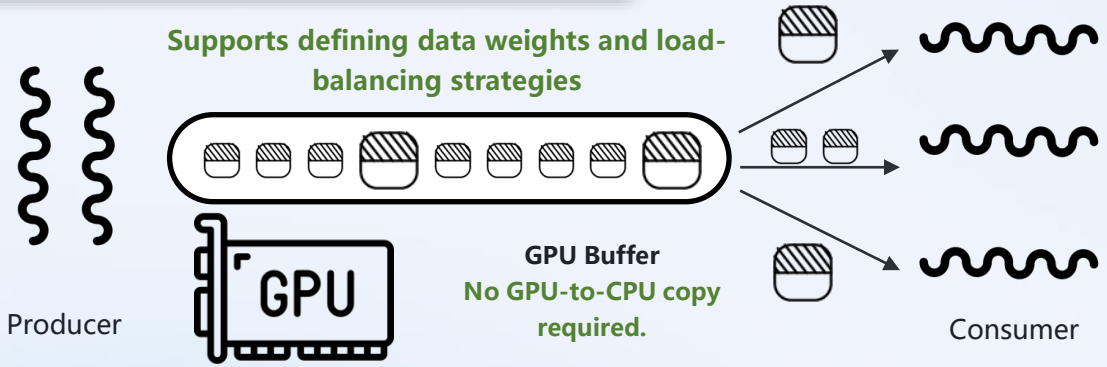

Fast scaling based on Lazy Initialization

## Multi-channel Communication

Concurrent communication using multiple CUDA streams and network streams; small data concurrency avoids Head-of-Line (HOL) blocking and reduces latency.




## Load-balanced Transfer Queue

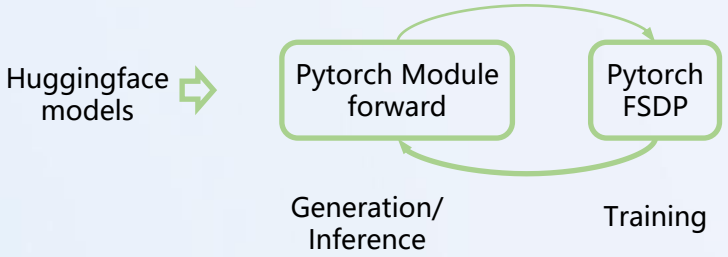



Achieves efficient and convenient pipelined data exchange between multi-components.

# Other: Multi-backend Integration

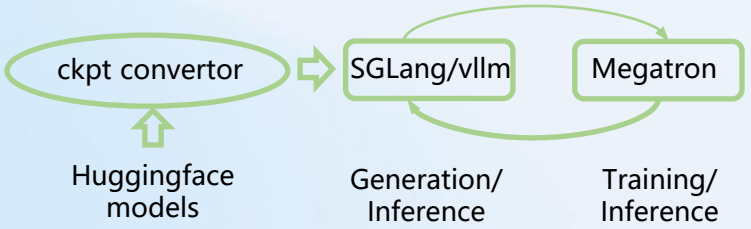
## Huggingface+FSDP

- Out-of-the-box support for Hugging Face models, no adaptation required.
- The preferred mode for rapid, small-scale validation.

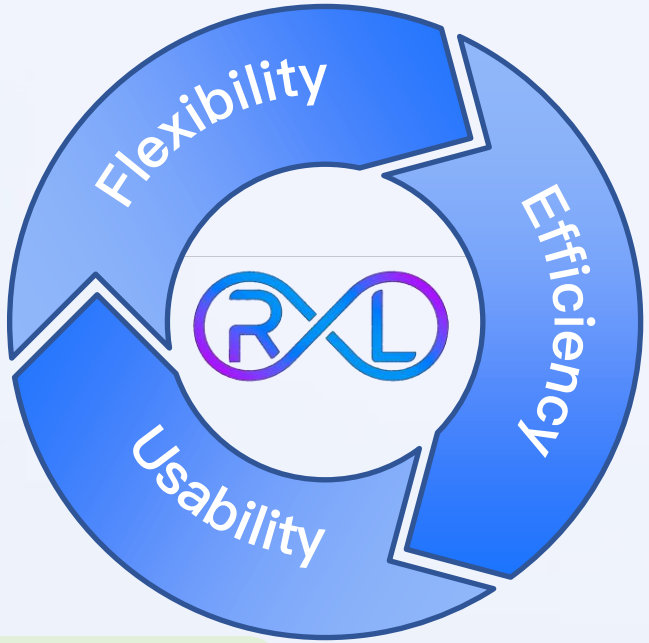



## Megatron+SGLang/vllm

- Rapid adaptation for models already supported by Megatron and SGLang/vLLM.
- Rapid adaptation for models already supported by Megatron and SGLang/vLLM.

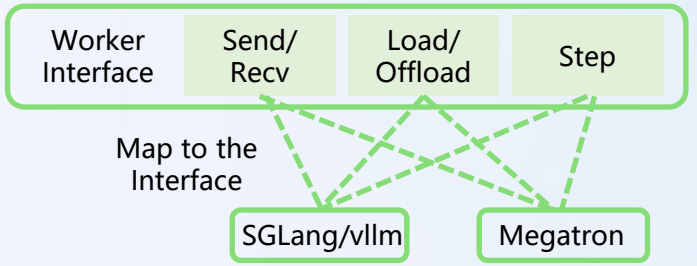



Embodied "Cerebellum" (VLA)




Embodied "Cerebrum" (VLM)

**A novel, non-intrusive framework for integration solution**




## Supports all optimization capabilities of Megatron.

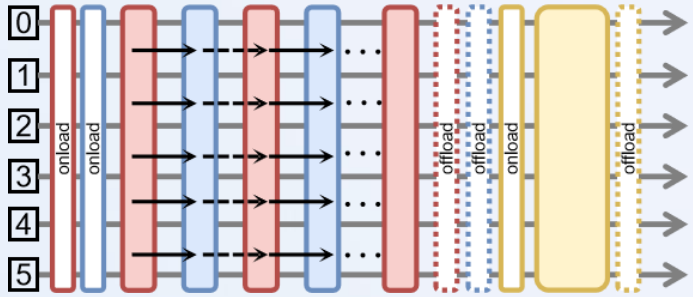
- Supports all types of parallelism, including data, tensor, pipeline (1F1B, VPP, ZeroBubble), sequence, context, and expert parallelism.

## Supports all optimization capabilities of SGLang/vllm.

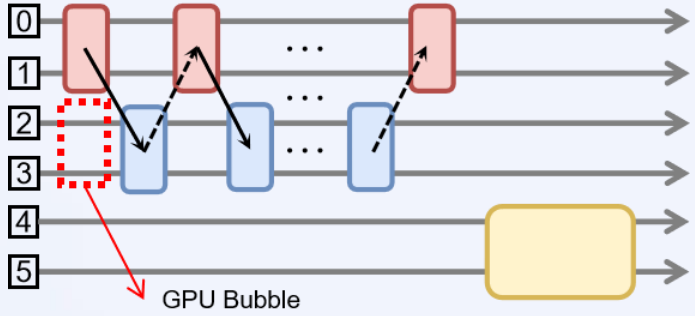
- Supports batched rollouts.
- Supports rollouts based on online serving.

# System Experiments for GPU Allocation

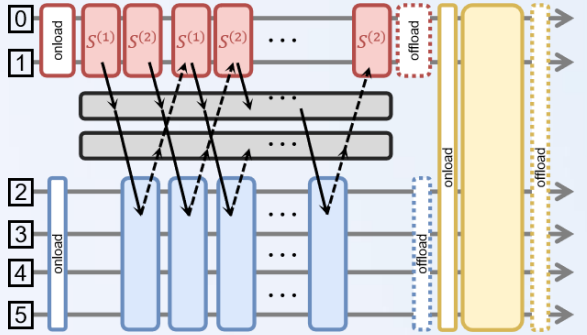
## ➢ Colocated Mode



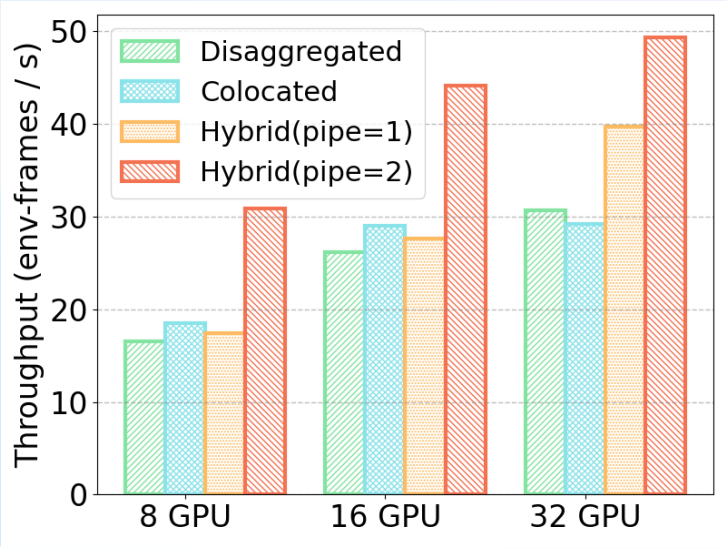

## ➢ Disaggregated Mode



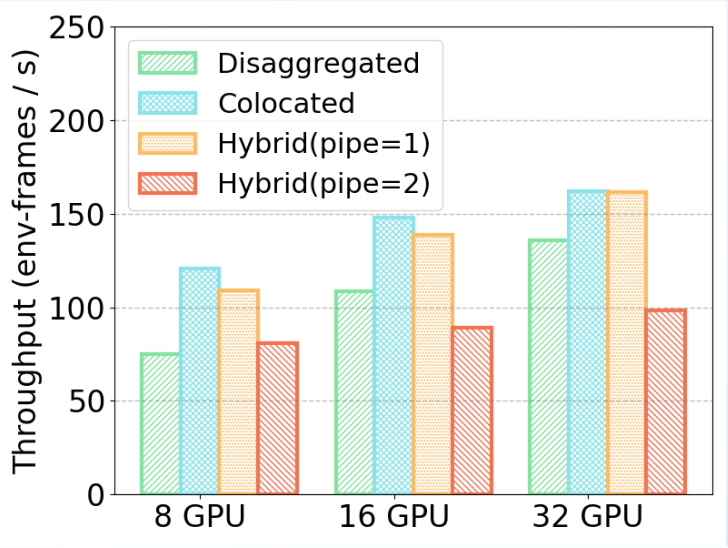

## ➢ Hybrid Mode



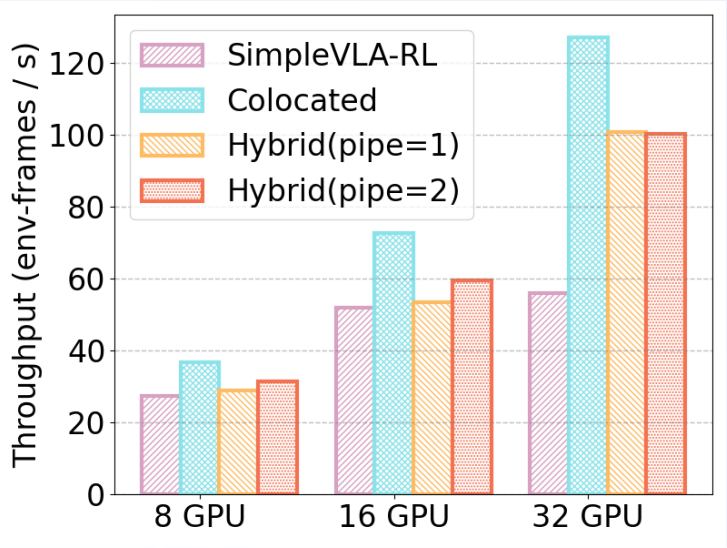

OpenVLA, ManiSkill




OpenVLA-OFT, ManiSkill




OpenVLA-OFT, LIBERO



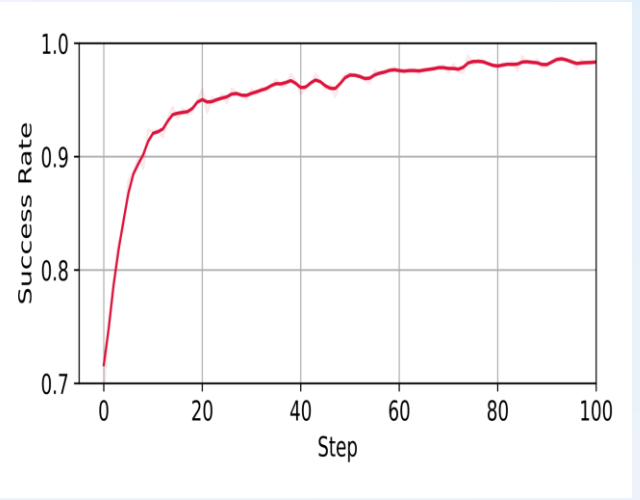
# 03

# Algorithm-level Design

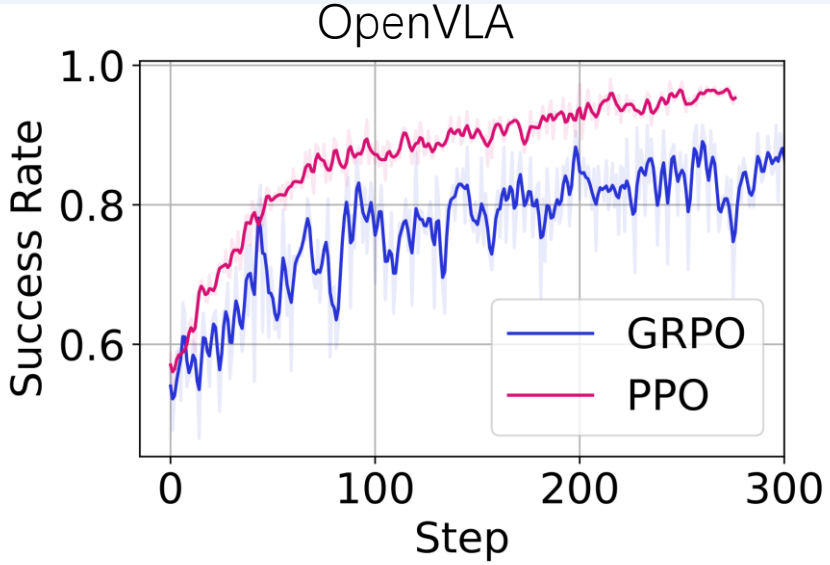
# High Performance: LIBERO

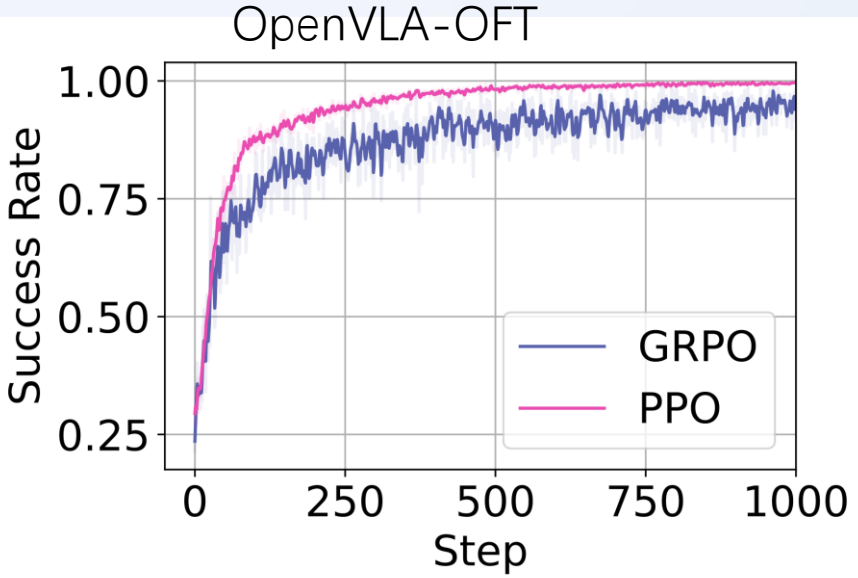
➢Note: A single model trains and evaluates on all 130 tasks on LIBERO.

| | OpenVLA-OFT (Base) | OpenVLA-OFT (RLinf-GRPO) | Δ Improvement |
|---|---|---|---|
| Spatial | 72.18% | 99.40% | **+27.22%** |
| Object | 71.48% | 99.80% | **+28.32%** |
| Goal | 64.06% | 98.79% | **+34.73%** |
| 10 | 48.44% | 93.95% | **+45.51%** |
| 90 | 70.97% | 98.59% | **+27.62%** |
| Average | 65.43% | 98.11% | **+32.68%** |



# High Performance: ManiSkill



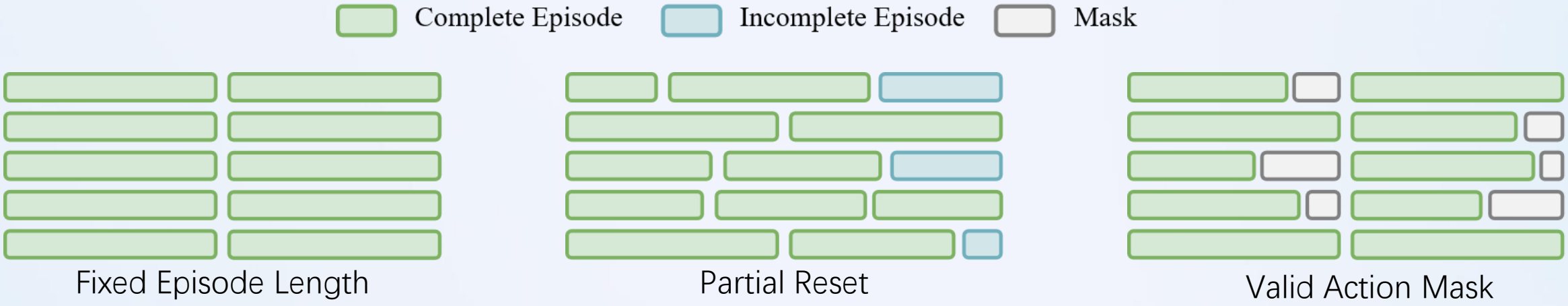




| | In-Distribution | Out-Of-Distribution | | | |
|---|---|---|---|---|---|
| | | Vision | Semantic | Execution | Avg. |
| OpenVLA (Base) | 53.91% | 38.75% | 35.94% | 42.11% | 39.10% |
| RL4VLA (PPO) | 93.75% | 80.47% | 75.00% | 81.77% | 79.15% |
| OpenVLA (RLinf-GRPO) | 84.38% | 74.69% | 72.99% | 77.86% | 75.15% |
| OpenVLA (RLinf-PPO) | 96.09% | 82.03% | **78.35%** | **85.42%** | **81.93%** |
| OpenVLA-OFT (Base) | 28.13% | 27.73% | 12.95% | 11.72% | 18.29% |
| OpenVLA-OFT (RLinf-GRPO) | 94.14% | 84.69% | 45.54% | 44.66% | 60.64% |
| OpenVLA-OFT (RLinf-PPO) | **97.66%** | **92.11%** | 64.84% | 73.57% | 77.05% |

# Overview

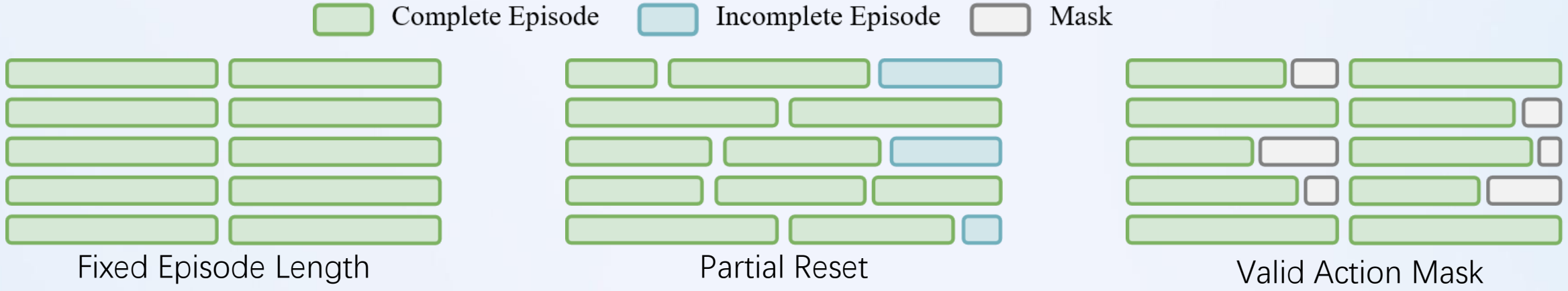
How to achieve these results?
See implementation details!
(Main Idea: Combining ideas from robotics RL and LLM RL.)

# Rollout Strategy



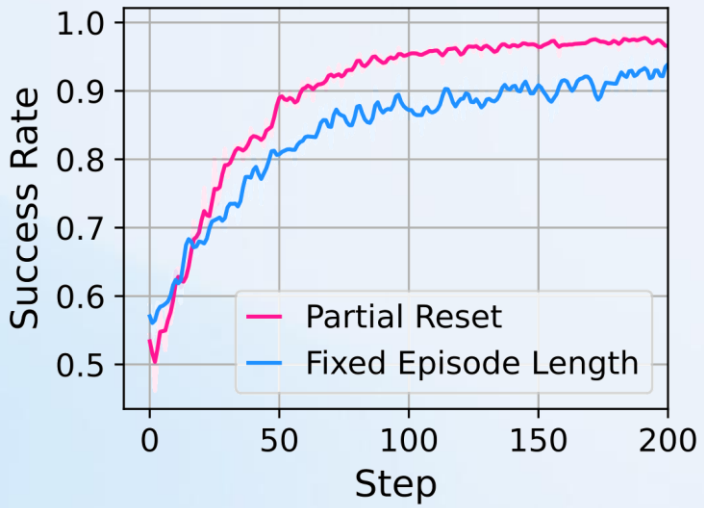

| | Reset Criterion | Train Data | Optimization Objective | Current Support Algos |
|---|---|---|---|---|
| Fixed Episode Length | Reach the max-episode-steps | All rollout data | The ability to maintain the success state. (success_at_end) | PPO, GRPO |
| Partial Reset | When the episode succeeds | All rollout data | The ability to achieve success at least once in one episode (success_once) | PPO |
| Valid Action Mask | Reach the max-episode-steps | Data before the first success | The ability to achieve success at least once in one episode (success_once) | GRPO |

# Rollout Strategy

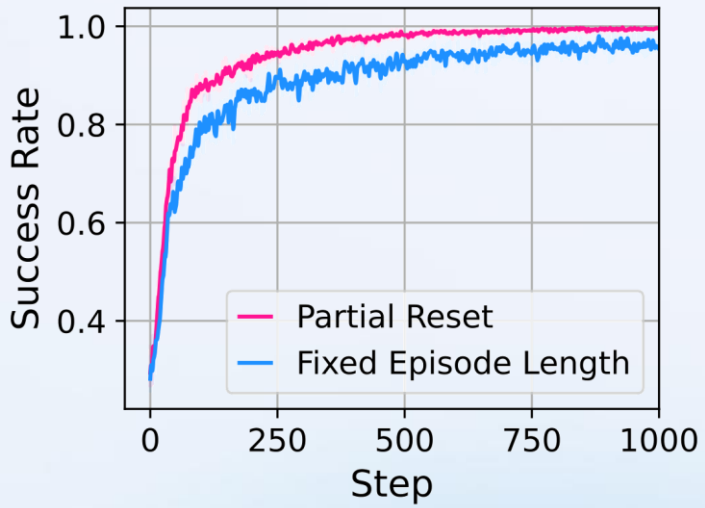



**PPO**

OpenVLA, ManiSkill

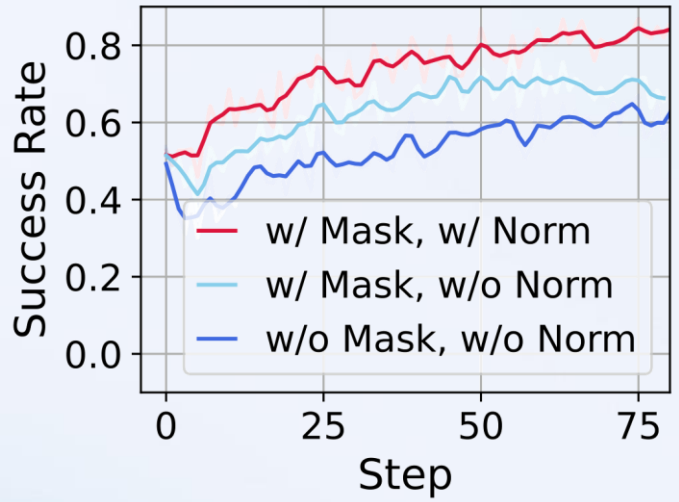



OpenVLA-OFT, ManiSkill



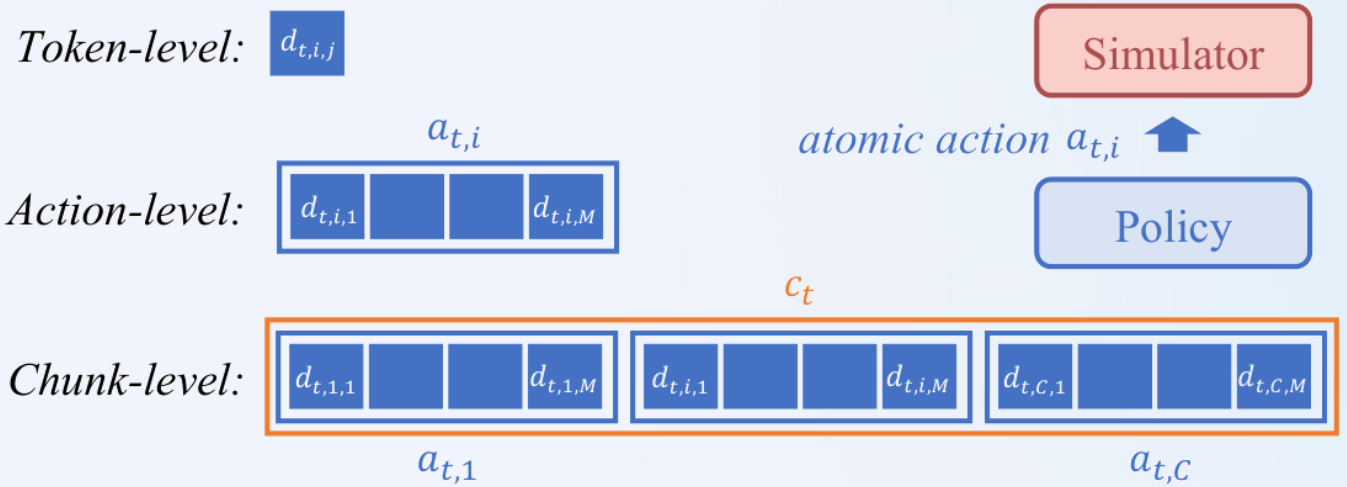

**GRPO**

OpenVLA-OFT, LIBERO



# Advantage and Log-probability

- Action Granularity
  - LLM: token
  - Robotics: action / chunk
- Supported Granularity
  - advantage: action, chunk
  - logprob: token, action, chunk

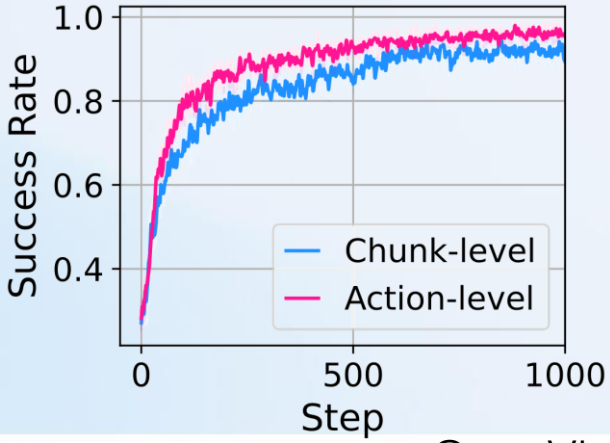


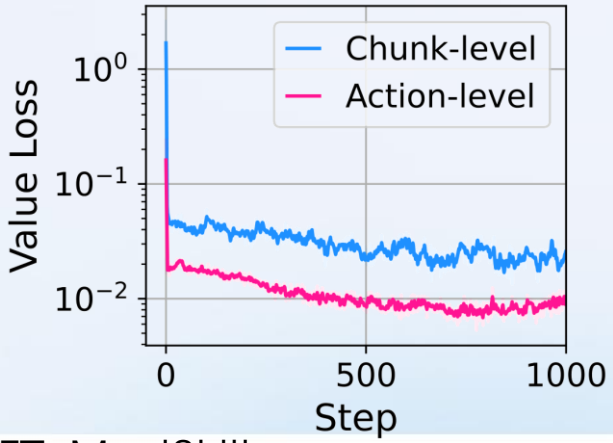

- **Chunk-level**: $\pi(c_t|o_t) = \prod_{i=1}^{C} \pi(a_{t,i}|o_t, a_{t,:i-1})$,
- **Action-level**: $\pi(a_{t,i}|o_t, a_{t,:i-1}) = \prod_{j=1}^{M} \pi(d_{t,i,j}|o_t, d_{t,i,:j-1})$,
- **Token-level**: $\pi(d_{t,i,j}|o_t, d_{t,i,:j-1})$.

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

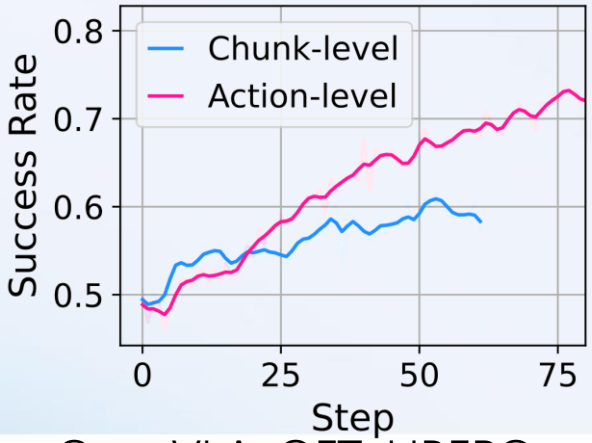
# PPO Details: Critic Design

- Use ValueHead as critic
  - A three-layer MLP
  - Attached to the last hidden-state of LLM for observation
- Value-type
  - Corresponds to the advantage granularity.
  - action-level is better.




OpenVLA-OFT, ManiSkill




OpenVLA-OFT, LIBERO

# GRPO Details: Normalization

- GRPO: every **trajectory** has equal contribution

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

$$\hat{A}^{(i)} = \hat{A}_t^{(i)} = \frac{\mathcal{R}^{(i)} - \text{mean}(\{\mathcal{R}^{(j)}\}_{j=1}^{G})}{\text{std}(\{\mathcal{R}^{(j)}\}_{j=1}^{G})}, \quad \forall t \in \{0, 1, \ldots, |\tau^{(i)}|\}, \qquad \rho_t^{(i)}(\theta) = \frac{\pi_\theta(a_t^{(i)} \mid o_t^{(i)})}{\pi_{\theta_{\text{old}}}(a_t^{(i)} \mid o_t^{(i)})},$$

- DAPO: every **token** has equal contribution

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[ \frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G} \sum_{t=1}^{|o_i|} \min\left( r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}}\right)\hat{A}_{i,t} \right) \right],$$

$$\text{s.t.} \quad 0 < \left| \{o_i \mid \texttt{is\_equivalent}(a, o_i)\} \right| < G.$$

# GRPO Details: Normalization

| | Better Trajectories | Normalization |
|---|---|---|
| LLM | Longer | token |
| Robotics | Shorter | trajectory |

- GRPO: every **trajectory** has equal contribution

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

$$\hat{A}^{(i)} = \hat{A}_t^{(i)} = \frac{\mathcal{R}^{(i)} - \text{mean}(\{\mathcal{R}^{(j)}\}_{j=1}^{G})}{\text{std}(\{\mathcal{R}^{(j)}\}_{j=1}^{G})}, \quad \forall t \in \{0, 1, \ldots, |\tau^{(i)}|\}, \qquad \rho_t^{(i)}(\theta) = \frac{\pi_\theta(a_t^{(i)} \mid o_t^{(i)})}{\pi_{\theta_{\text{old}}}(a_t^{(i)} \mid o_t^{(i)})},$$

- DAPO: every **token** has equal contribution

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[ \frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G} \sum_{t=1}^{|o_i|} \min\left( r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}}\right)\hat{A}_{i,t} \right) \right],$$

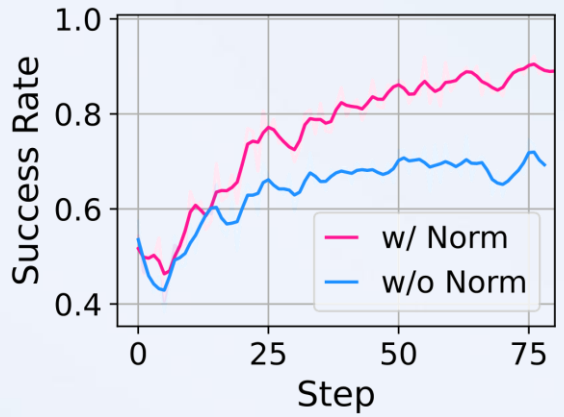
$$\text{s.t.} \quad 0 < \left| \{o_i \mid \texttt{is\_equivalent}(a, o_i)\} \right| < G.$$

# GRPO Details: Normalization

- Loss in the code with trajectory length normalization

```
policy_loss = (
    torch.max(policy_loss, policy_loss2) / loss_mask_ratio
) * loss_mask
policy_loss = policy_loss.mean()
clip_fraction = torch.gt(policy_loss2, policy_loss).float() * loss_mask
clip_fraction = clip_fraction.mean()
ppo_kl = (-logratio * loss_mask).mean()
```

| | Code | Loss |
|---|---|---|
| w/ Norm | divide loss_mask_ratio | GRPO (trajectory-level) |
| w/o Norm | no loss_mask_ratio | ~DAPO (token-level) |



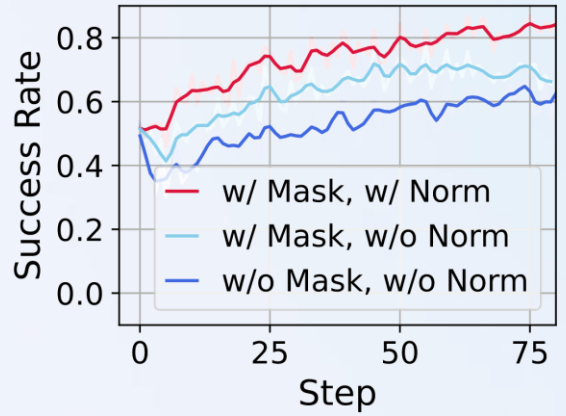
# GRPO Details: Normalization

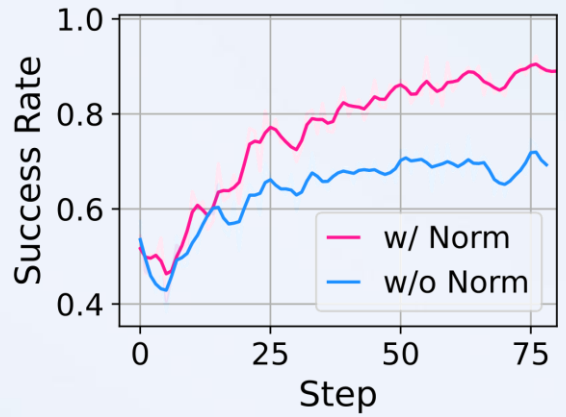
- Loss in the code

```
policy_loss = (
    torch.max(policy_loss, policy_loss2) / loss_mask_ratio
) * loss_mask
policy_loss = policy_loss.mean()
clip_fraction = torch.gt(policy_loss2, policy_loss).float() * loss_mask
clip_fraction = clip_fraction.mean()
ppo_kl = (-logratio * loss_mask).mean()
```

| | Code | Loss |
|---|---|---|
| w/ Norm | divide loss_mask_ratio | GRPO (trajectory-level) |
| w/o Norm | no loss_mask_ratio | ~DAPO (token-level) |

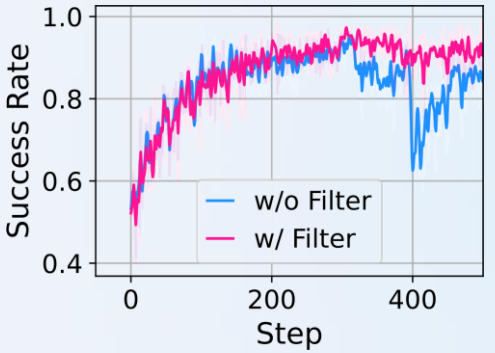
Valid Action Mask



# GRPO Details: Filter

- Adopted from DAPO idea
- Omit groups where all trajectories have the same return value.
- Effects:
  - Stablize the training process and thus achieves better performance

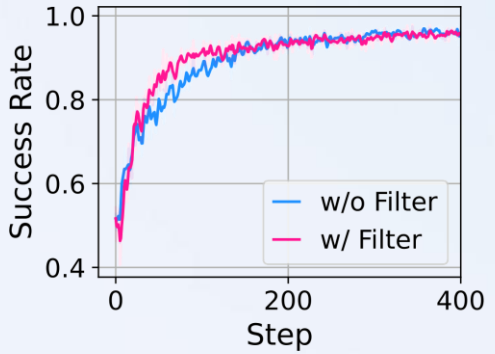


OpenVLA, ManiSkill

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\{R_i\}_{i=1}^{G})}{\text{std}(\{R_i\}_{i=1}^{G})}.$$

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[ \frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G}\sum_{t=1}^{|o_i|} \min\left(r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}}\right)\hat{A}_{i,t}\right) \right]$$

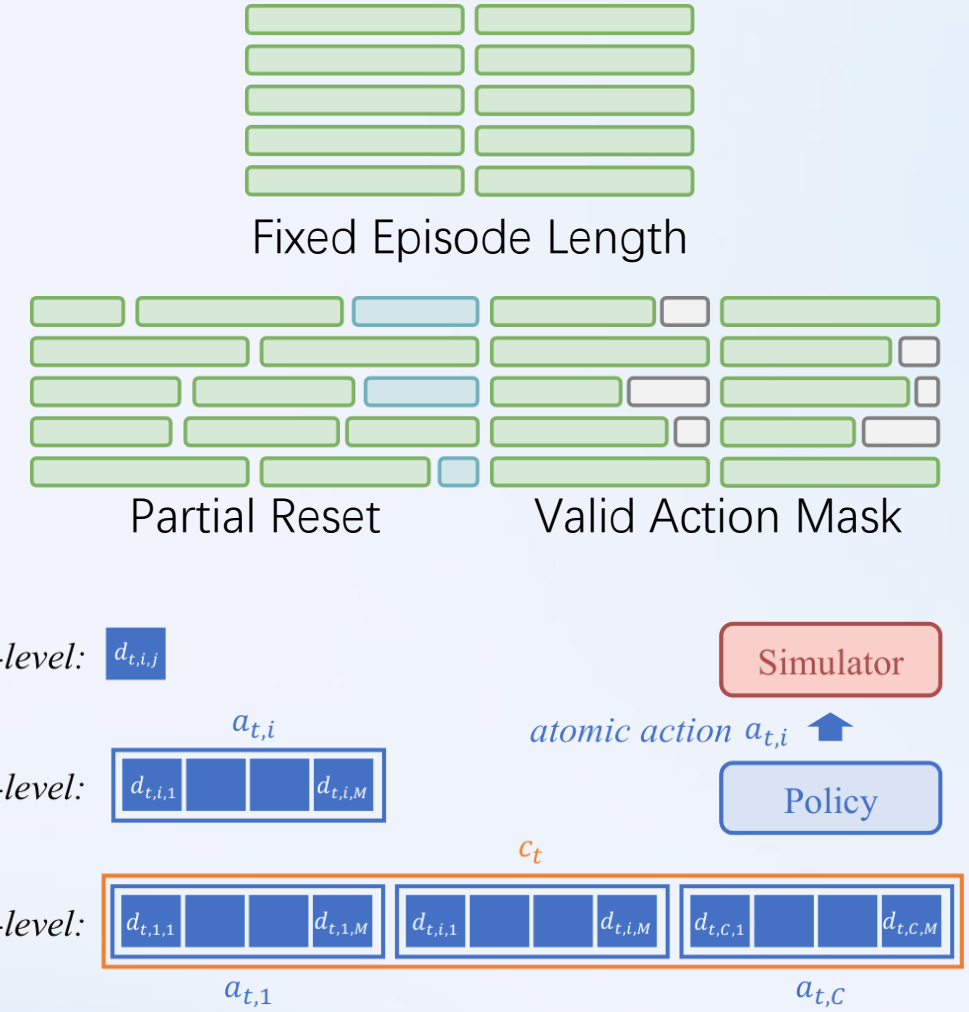
$$\text{s.t.} \quad 0 < \left| \{o_i \mid \texttt{is\_equivalent}(a, o_i)\} \right| < G.$$



OpenVLA-OFT, ManiSkill

# Summary for Implementation Details

- General
  - Support various rollout strategies.
  - Support various advantages and logprob granularity.
- PPO
  - Use a simple MLP critic sharing backbone with the actor.
  - Use action-level value.
- GRPO
  - Normalize loss by trajectory length.
  - Use filter.



# Real World Deployment

| Object Type | OpenVLA (SFT) Pick | OpenVLA (SFT) Success | OpenVLA (RLinf-PPO) Pick | OpenVLA (RLinf-PPO) Success |
|---|---|---|---|---|
| Pepper | 1/5 | 0/5 | 2/5 | 1/5 |
| Cucumber | 1/5 | 0/5 | 1/5 | 1/5 |
| Banana | 0/5 | 0/5 | 3/5 | 3/5 |
| Kiwi | 0/5 | 0/5 | 2/5 | 1/5 |
| Mangosteen | 1/5 | 0/5 | 1/5 | 1/5 |
| Sponge | 0/5 | 0/5 | 4/5 | 1/5 |
| Total | 3/30 | 0/30 | 13/30 | 8/30 |

SFT Model

RL Model

# Star and Start!


**Welcome to RLinf!**

https://github.com/RLinf/RLinf
https://rlinf.readthedocs.io/en/latest/

InfoQ 极客传媒

北京

QCon 全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

北京

AiCon 全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

上海

QCon 全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月 — 5月 — 6月 — 8月 — 10月 — 12月

上海

AiCon 全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

深圳

AiCon 全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

北京

AiCon 全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

# 极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播

参会咨询

查看会议

QCon
全球软件开发大会

InfoQ 极客传媒

# THANKS

Chao Yu

zoeyuchao@gmail.com