ROLL

like a Reinforcement Learning Algorithm Developer

ROLL: Reinforcement Learning Optimization for Large-scale Learning

Future Living Lab

智能引擎事业部

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Presenter: Wei Gao (ROLL Team)
November 18, 2025

Agenda

Part 1： ROLL

Part 2： ROLL-Flash

Part 3： RollPacker

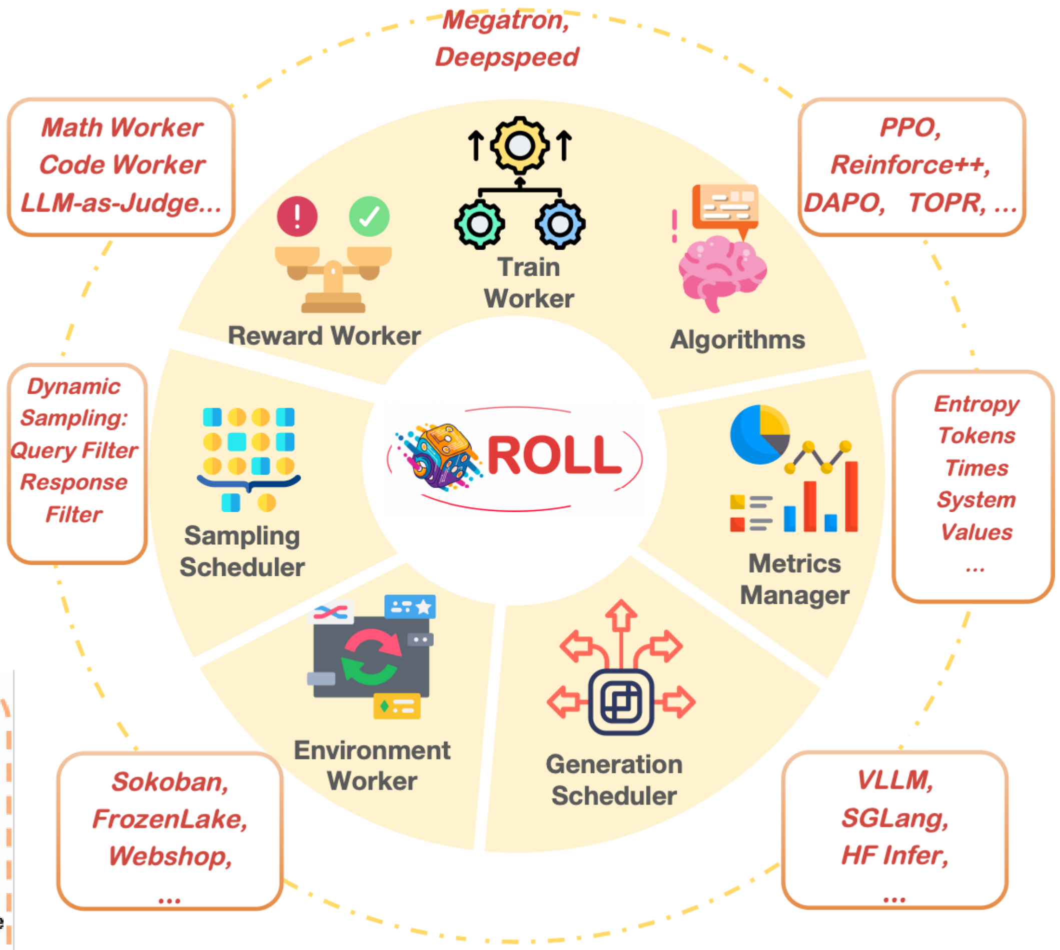
Part 1：ROLL

System Overview and Key Features

- **Objective**: An Efficient and User-Friendly Scaling Library for Reinforcement Learning with LLMs
- **Key Features**:
  - **Rich Training and Inference Engine:** Ray-based multi-role distributed architecture. Strategy abstraction unifies various backends.
  - **Modular Design:** Key modules (e.g., Rollout Scheduler, AutoDeviceMapping) streamline RL pipeline development.
  - **Sample-level Rollout Lifecycle Management:** dynamical sampling (DAPO), sampling ratio control.
  - **Observability:** Integrated with SwanLab / WandB / TensorBoard, tracking of performance for each domain and reward type.

🚀 *Why ROLL?*

- 📈 Scalable Training
- ⚙️ Agentic Support
- ⚒ Agile Experiment
- 📦 Modular & Pluggable
- ⚡ Fast & Effective
- ...

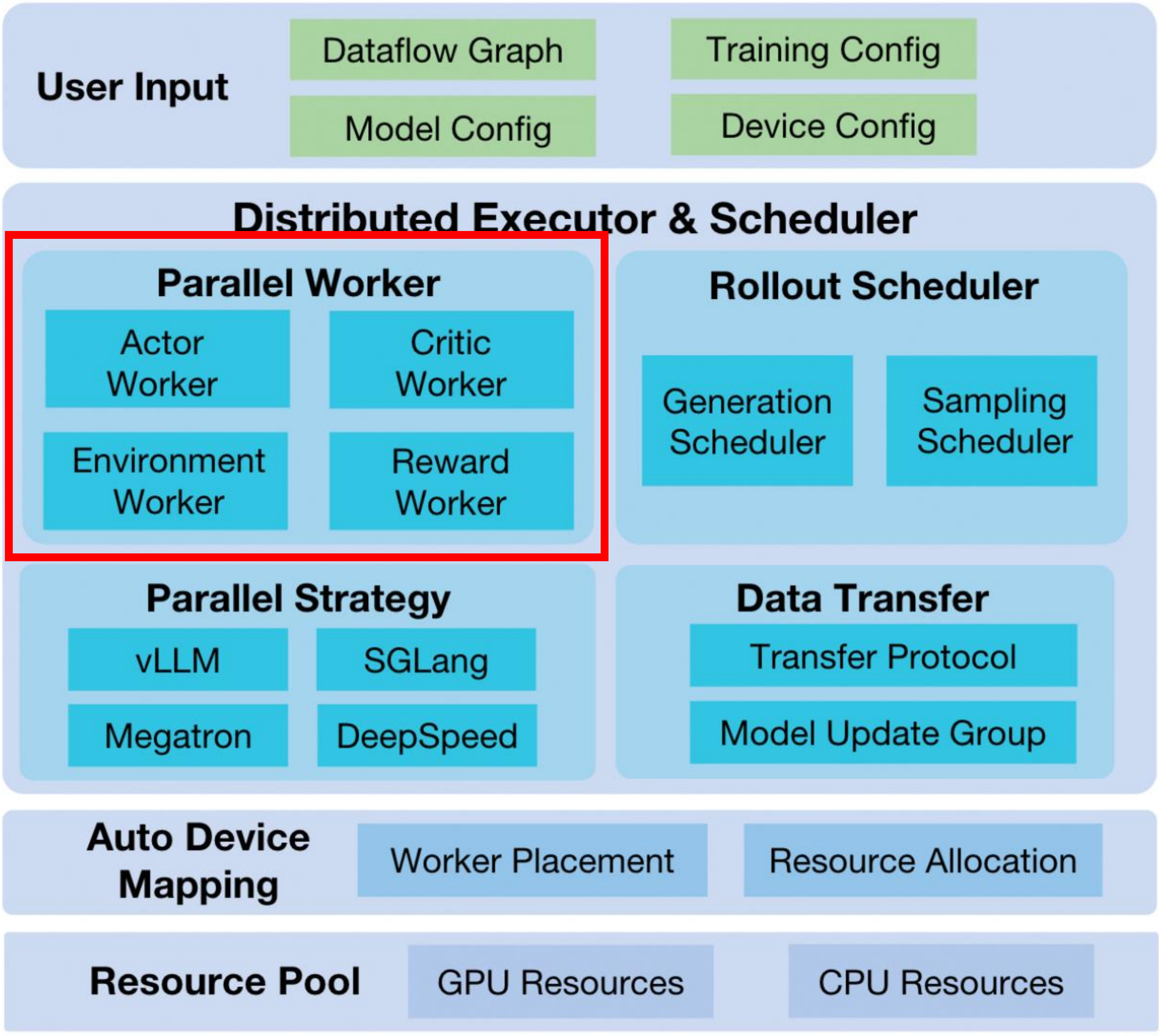


Framework Architecture

Distributed Architecture for RL Post-Training

Distributed Execution Unit：Parallel Worker

- The owner of a set of Ray PlacementGroup resources
  - Each worker is the fundamental execution unit.
  - Enables single-controller programming for distributed execution.
- **Cluster** Management and Orchestration
  - A **Cluster** serves as an abstraction for a collective of resources. Within this abstraction, workers fulfilling roles such as Actor, Critic, Environment, and Reward are centrally managed.
  - Users can define the data flow between Clusters, thereby managing the overall training pipeline.



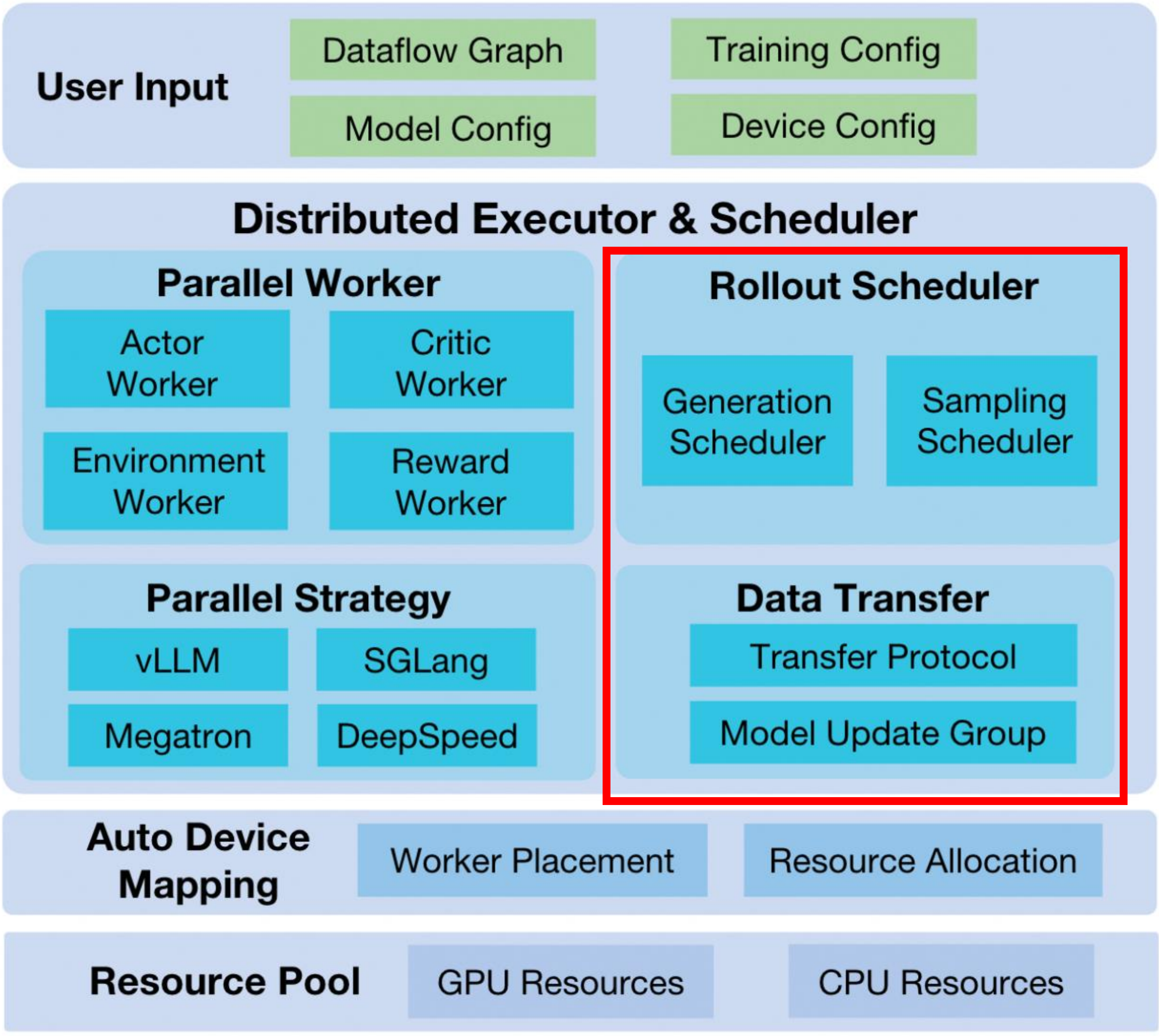
(a) Architecture

Framework Architecture

Distributed Architecture for RL Post-Training

- **Rollout Scheduler:** Enables sample-level rollouts, dynamic load balancing, and flexible prompt routing.

- **Data Transfer:** Ensures highly efficient parameter synchronization through the ModelUpdateGroup mechanism.

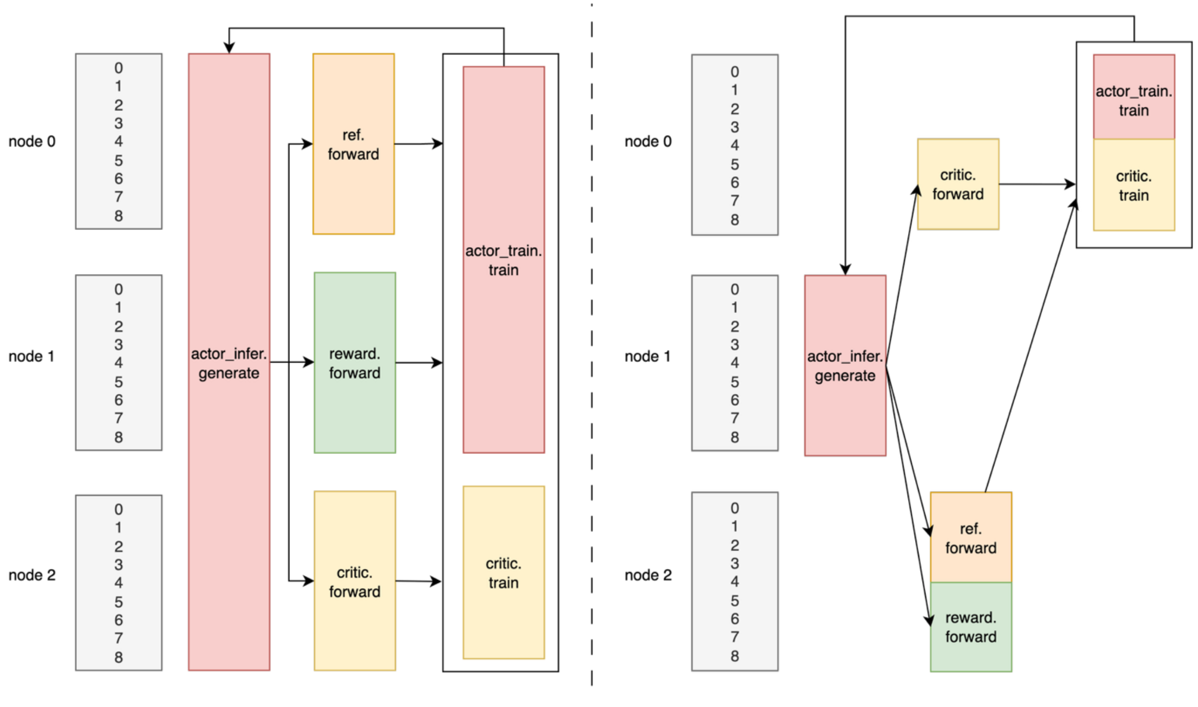


(a) Architecture

Framework Architecture

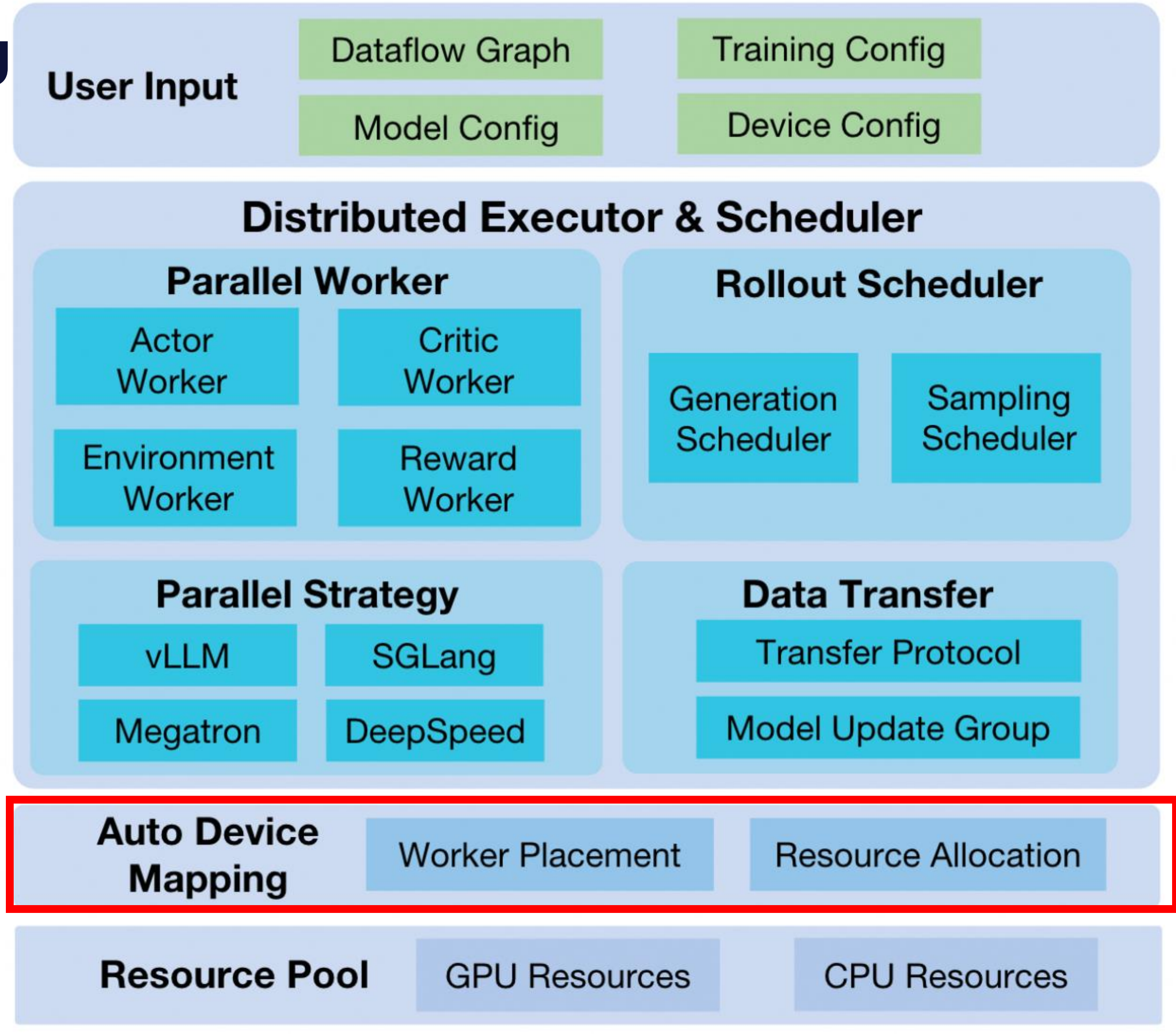
Distributed Architecture for RL Post-Training

- **AutoDeviceMapping:** Flexible resource allocation enabled by user-defined device mapping, supporting both **colocated** and **disaggregated** training paradigm.



colocated device_mapping

disaggregated device_mapping



(a) Architecture

System Workflow

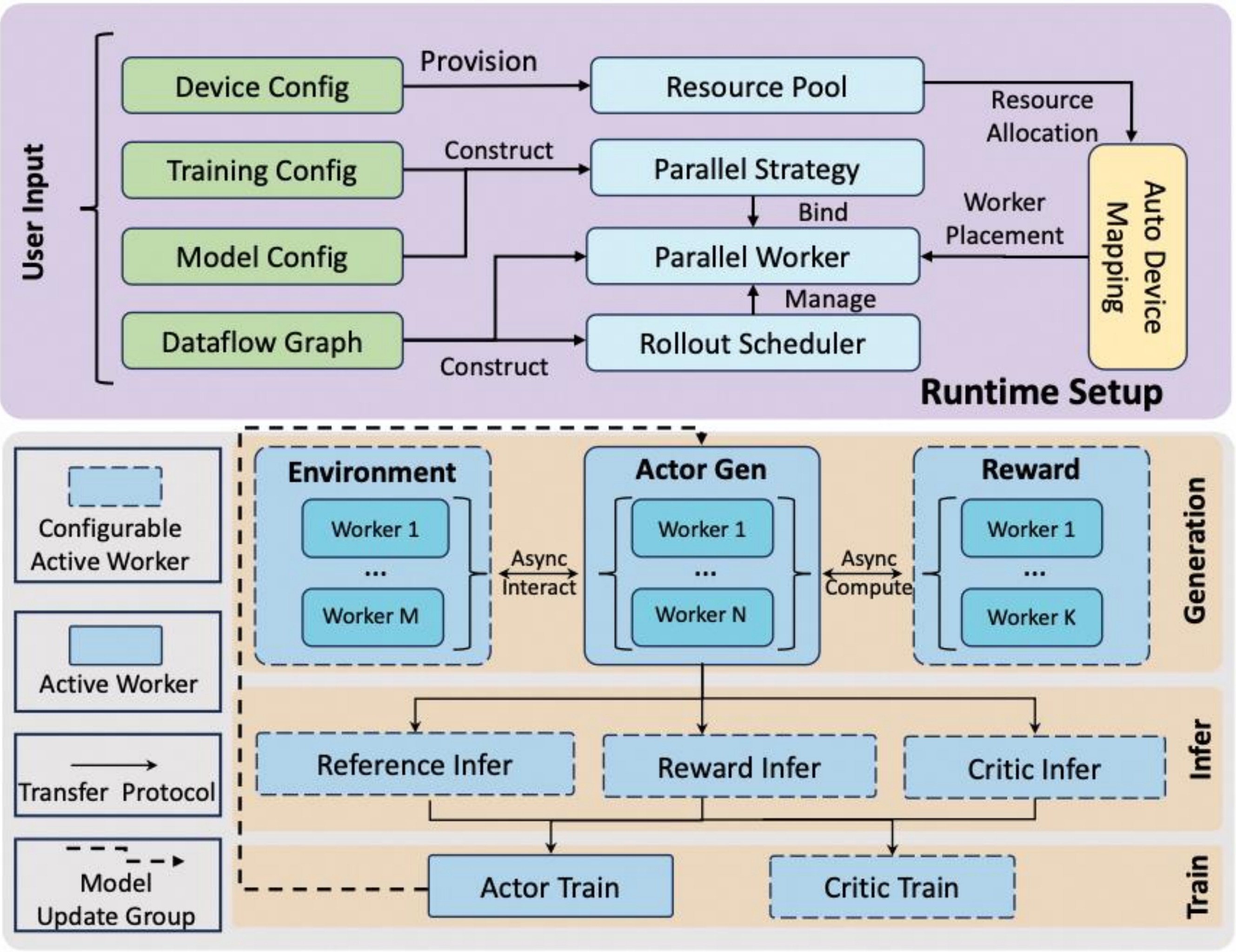
Initialization and Online Training

- **Runtime Setup**
  - Instantiate different workers and bind workers with resources based on user-defined configs.
- **Training Workflow**
  - Rollout Generation

    The Actor, Environment, and Reward components execute concurrently and asynchronously to facilitate the generation of rollouts.
  - Inference

    Inference logits to serve as the supervision signal
  - Train

    Model Update: Actor and Critic update model weights, and synchronize the weights with the rollout stage.



(b) Workflow

System Evaluation : RLVR

- **RLVR**: optimize LLM with RL for verifiable tasks (e.g., mathematics, code)
- **Evaluation Setup**:
  - Dataset：DeepMath、KodCode, etc
  - Model：Qwen2.5-7B-Base, Qwen3-30B-A3B-Base
  - Algorithm：PPO Loss+REINFORCE，rule-based, sandbox execution, and LLM-as-Judge verficiation
- **Result** (Figure 3a, 4a):
  - Qwen2.5-7B-Base: The accuracy improves from 0.18 to 0.52. Math：0.20->0.53；Code：0.13->0.41
  - Qwen3-30B-A3B-Base: Accuracy improves from 0.27 to 0.62
- **Resilience**: Robust and continuous training w/o failures.
- **Multi-modality Support**: Superior performance for Qwen2.5-VL-7B-Instruct on GEOQA_R1V_Train_8K (Figure 4c)

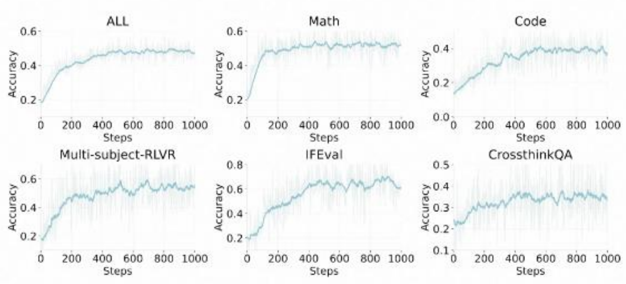


Figure 3: Accuracy Trends Across Different Tasks on Qwen2.5-7B-Base.

Figure3 a. Dense: Qwen2.5-7B-Base

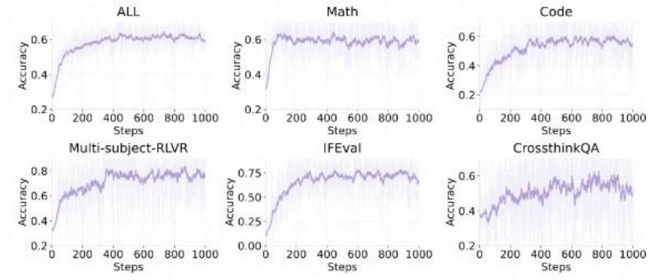


Figure 4: Accuracy Trends Across Different Tasks on Qwen3-30B-A3B-Base.

Figure 4 a. MOE: Qwen3-30B-A3B-Base

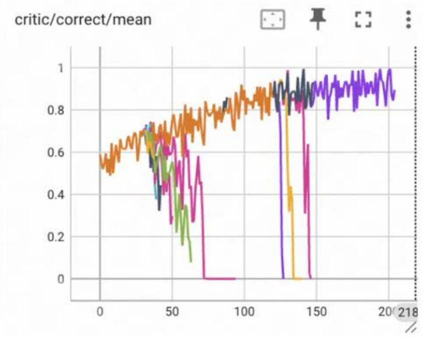


Figure 3b.MOE 200+B
Crash and resume

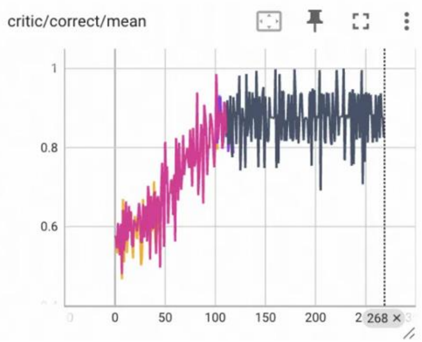


Figure 4b.MOE 200+B
Stable training

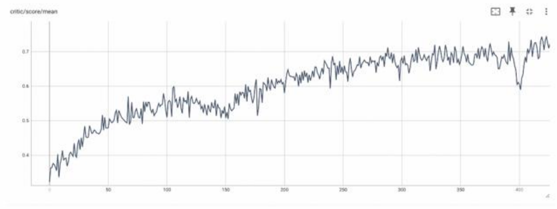


Figure 4 c Qwen2.5-VL-7B-Instruct 训练score曲线

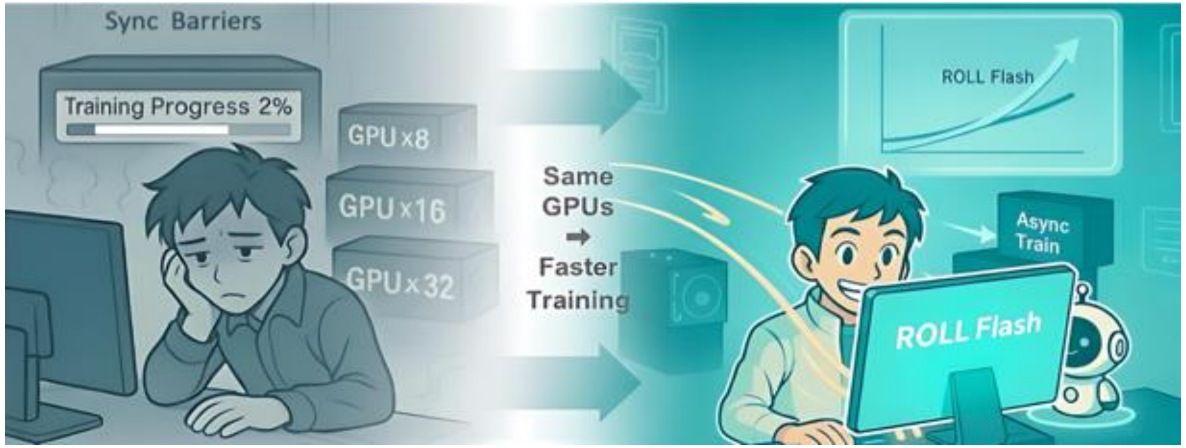
Part 2：ROLL-Flash

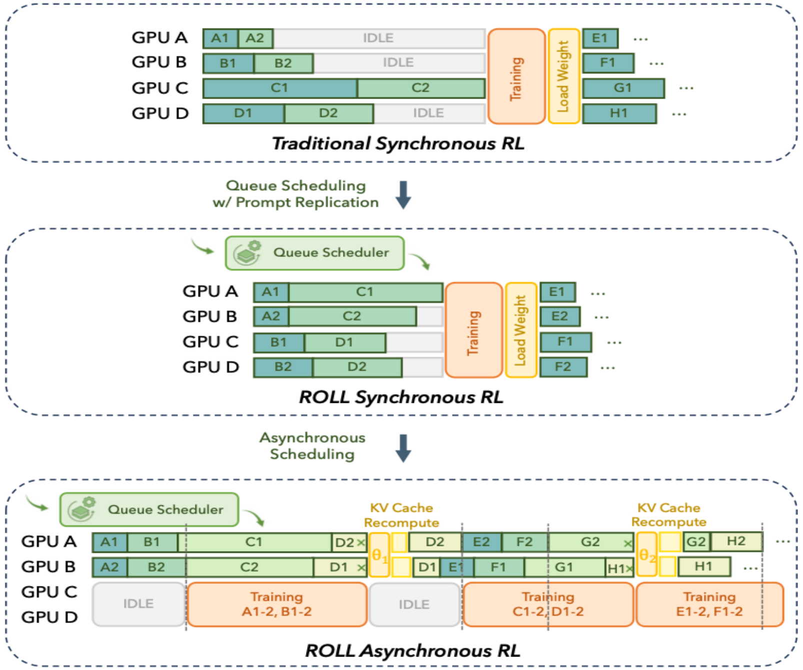
Accelerating RLVR and Agentic Training with Asynchrony

ROLL-Flash: ROLL with Asynchrony

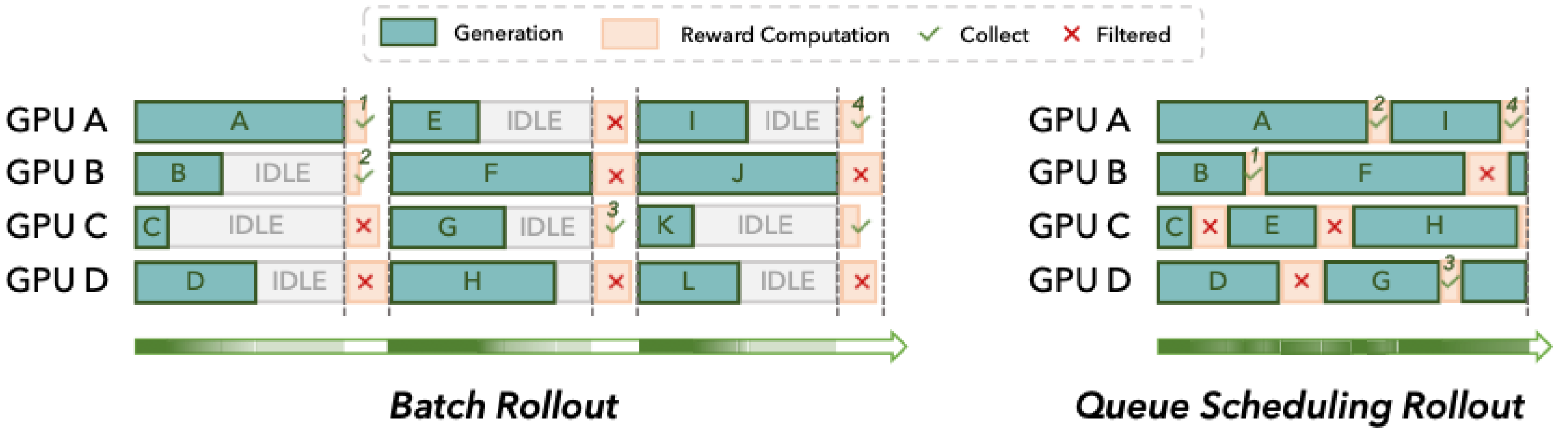
- **Objective**: Accelerate ROLL with native support for asynchronous RL post-training.
- **Design Principles**:
  - **Fine-grained Parallelism:** Sample-level lifecycle control during the rollout stage, enabling overlap among LLM generation, environment interaction, and reward computation, thereby reducing idle time and improving GPU utilization.

  - **Rollout-Train Decoupling:** places the rollout and training stages on separate resources and executes them in parallel. Consequently, the rollout stage does not wait for training to complete, and training can optimize the LLM using responses generated under stale policy.





Fine-grained Parallelism – Queue Scheduling



- **Batch Rollout:** The rollout generation, reward computation, and filtering are performed sequentially.
- **Queue Scheduling Rollout**: Once a prompt is ready, it immediately dispatches the corresponding reward worker to asynchronously compute the reward, thus substantially reduces the rollout overhead.

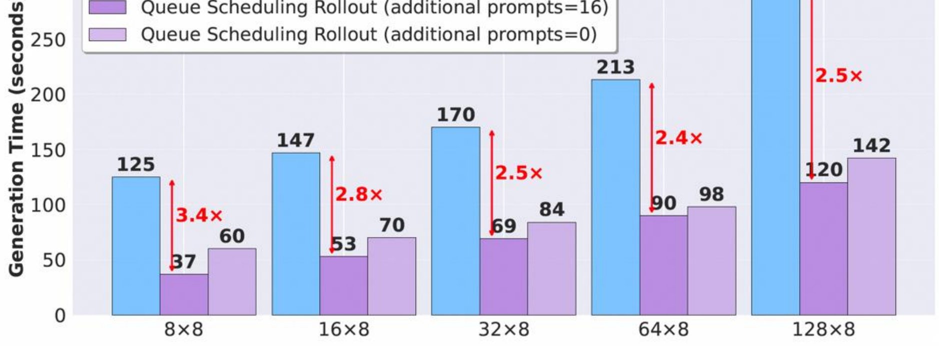


Figure 7: Efficiency comparison of generation time across different batch size×8 configurations.

Fine-grained Parallelism – Prompt Replication & Redundant Sampling

- **Prompt Replication:** expands each prompt into n independent rollout tasks, each running on separate GPUs, facilitating load balancing among rollout workers.
- **Redundant Environment Rollout:** This mechanism can tune (1) num env groups to spawn more concurrent environment groups, and (2) group size to generate more candidate trajectories per group, preventing fail-slow and fail-stop environments.

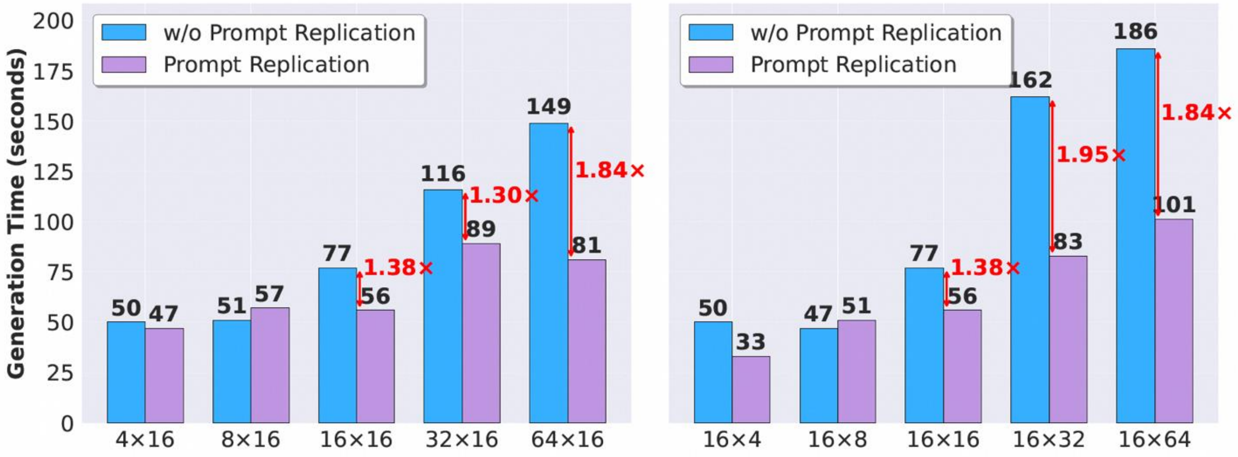
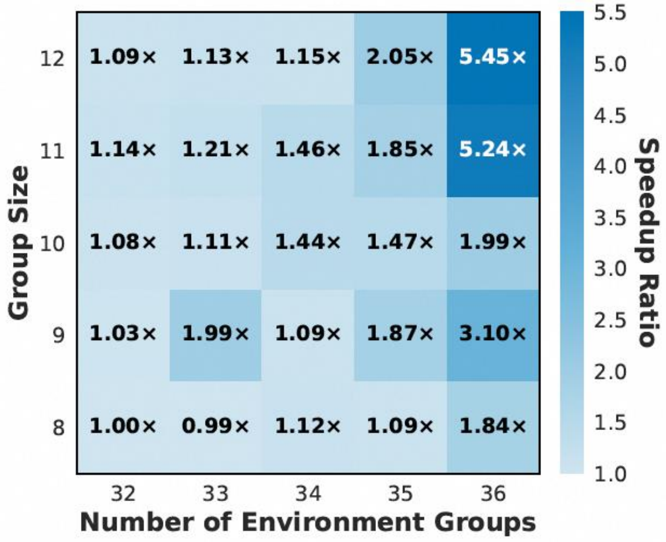


Figure 8: **Efficiency of using prompt replication across different rollout configurations. Left:** Vary-



Performance-Preserving Asynchronous Acceleration

- **Diverse Off-policy Algorithm Support**: We provide many off-the-shelf off-policy loss objectives for RL developers.

Loss Objective for Off-policy Algorithms

$$\textbf{PPO (Standard)}: \quad \mathcal{J}^{\mathrm{PPO}}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_{\mathrm{old}}}\left[\min\left(R(\tau)\frac{\pi_\theta(\tau)}{\pi_{\mathrm{old}}(\tau)}, R(\tau)\left(\frac{\pi_\theta(\tau)}{\pi_{\mathrm{old}}(\tau)}\right)_{1-\epsilon}^{1+\epsilon}\right)\right]$$

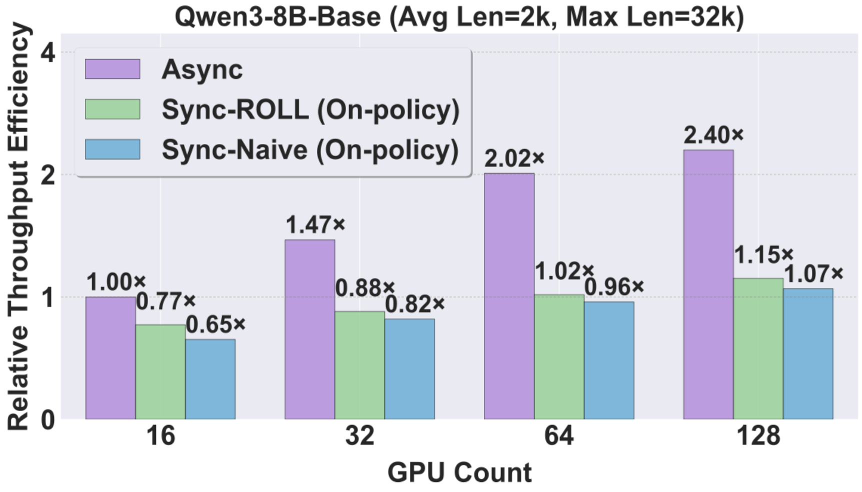
$$\textbf{Decoupled PPO}: \quad \mathcal{J}^{\mathrm{DPPO}}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_{\mathrm{old}}}\left[\min\left(R(\tau)\frac{\pi_\theta(\tau)}{\pi_{\mathrm{old}}(\tau)}, R(\tau)\frac{\pi_{\mathrm{prox}}(\tau)}{\pi_{\mathrm{old}}(\tau)}\left(\frac{\pi_\theta(\tau)}{\pi_{\mathrm{prox}}(\tau)}\right)_{1-\epsilon}^{1+\epsilon}\right)\right]$$

$$\textbf{Truncated IS}: \quad \mathcal{J}^{\mathrm{TIS}}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_{\mathrm{old}}}\left[\mathbf{sg}\left(\frac{\pi_\theta(\tau)}{\pi_{\mathrm{old}}(\tau)}\right)_0^c R(\tau)\log\pi_\theta(\tau)\right]$$

$$\textbf{CISPO}: \quad \mathcal{J}^{\mathrm{CISPO}}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_{\mathrm{old}}}\left[\mathbf{sg}\left(\frac{\pi_\theta(\tau)}{\pi_{\mathrm{old}}(\tau)}\right)_{1-\epsilon_{\mathrm{low}}^{\mathrm{IS}}}^{1+\epsilon_{\mathrm{high}}^{\mathrm{IS}}} R(\tau)\log\pi_\theta(\tau)\right]$$

$$\textbf{TOPR}: \quad \mathcal{J}^{\mathrm{TOPR}}(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_{\mathrm{old}}}\left[\left(\mathbf{1}_{\{\tau\in T^+\}} + \mathbf{1}_{\{\tau\in T^-\}}\,\mathbf{sg}\left(\frac{\pi_\theta(\tau)}{\pi_{\mathrm{old}}(\tau)}\right)_0^c\right)R(\tau)\log\pi_\theta(\tau)\right]$$

- **Empirical Analysis of Asynchronous Training**
  - **Takeaway 1: Async Achieves Superior Throughput Scalability.**
    - Increasing GPU resources causes Sync to suffer more from the impact of long-tail samples, whereas Async exhibits better scaling behavior and achieves higher resource utilization.
  - **Takeaway 2: Async Accelerates Training in Almost All Cases.**
    - Async effectively mitigates training stalls caused by long-tail generation overhead, delivering substantial speedups when the allocation of training and inference resources is well balanced.



Performance-Preserving Asynchronous Acceleration

- **Diverse Off-policy Algorithm Support**: Accelerate ROLL with native support for asynchronous RL post-training.

Table 1: **Async Ratio Required in various Configuration**

Model Size	0.6B	1.7B	4B	8B
Async Ratio	2	2	2	2
Length	4K	8K	16K	32K
Async Ratio	1	1	1	2
Rollout Size	32	64	128	256
Async Ratio	4	2	2	2

- **Empirical Analysis of Asynchronous Training**
  - **Takeaway 3: Async Ratio Can Be Small Enough.**
    - In typical configurations, setting the Asynchronous Ratio to 2 achieves the highest throughput, effectively balancing learning efficiency and the degree of off-policy learning.
  - **Takeaway 4: Async Training Can Be Stable and Nearly Performance-Lossless.**
    - Under Async Ratio 2 and 8 settings, various off-policy methods, as well as widely used GRPO algorithm, can consistently deliver performance gains on par with synchronous training.

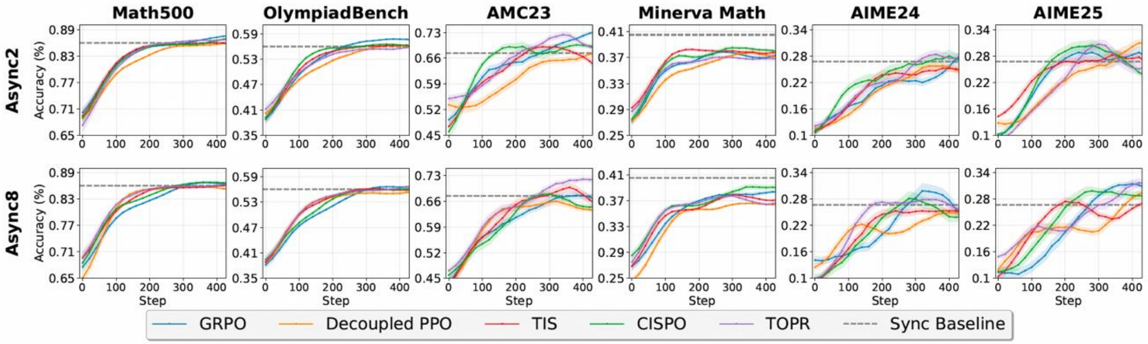
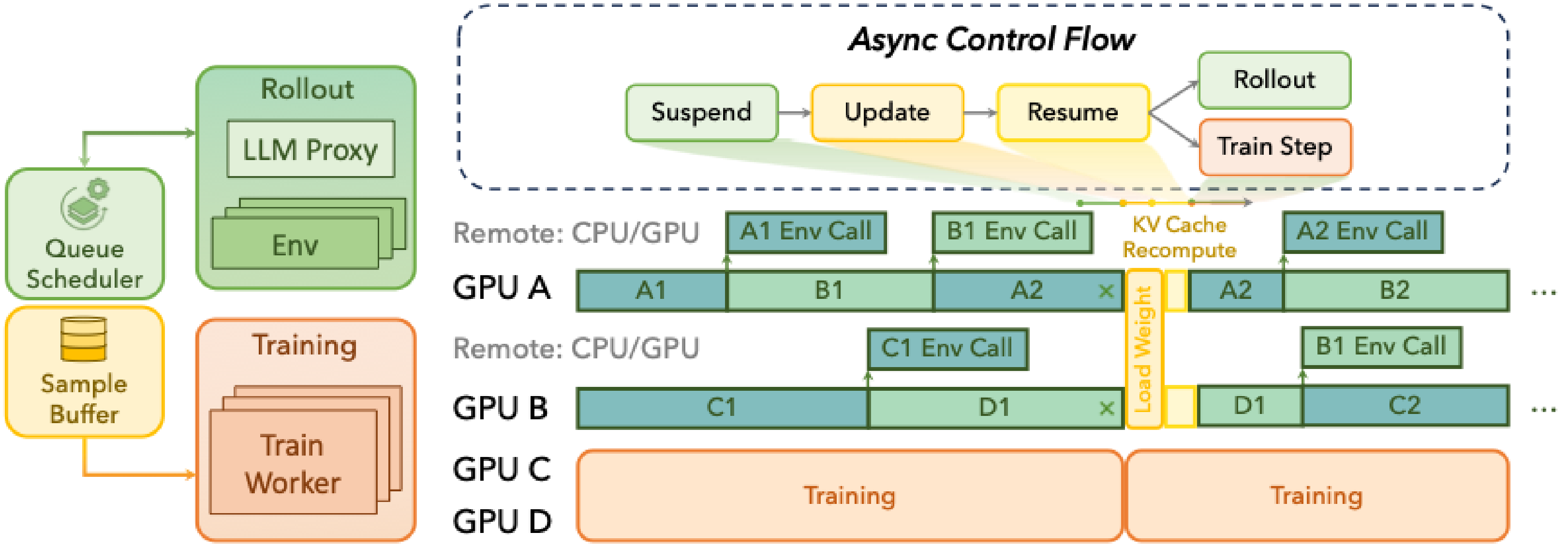


Figure 4: **Off-Policy Algorithm Performance Comparison under Async Ratio 2 and 8.** To ensure

Rollout-Train Decoupling

- **Execution workers for the rollout and training stages are run in a disaggregated and pipelined manner**
- **To manage staleness, we introduce AsyncController and a shared sample buffer.**
  - A pool of environment workers act as independent producers: they generate trajectories and enqueue them into **SampleBuffer**. The training worker fetches completed trajectory from buffer to perform gradient computation.
  - The **AsyncController** operates in three phases: it issues **suspend** to pause trajectory collection, executes **model update** by fetching and broadcasting the latest weights to all LLM serving workers, and then sends resume so the rollout stage continue collecting trajectories with the updated model.



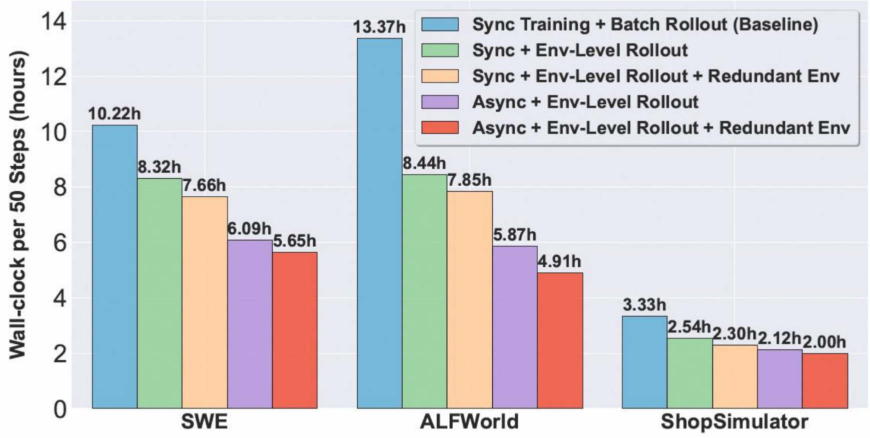


Figure 11: **Real-environment evaluation of environment-level asynchronous and redundant environment rollout.**

Part 3： RollPacker

Mitigating Long-Tail Rollouts for Fast, Synchronous RL Post-Training

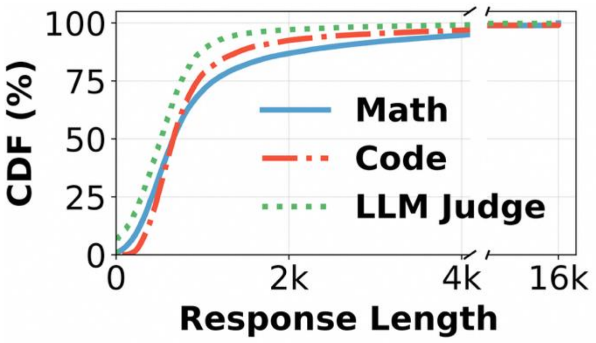
Synchronous Systems Suffers from GPU Underutilization

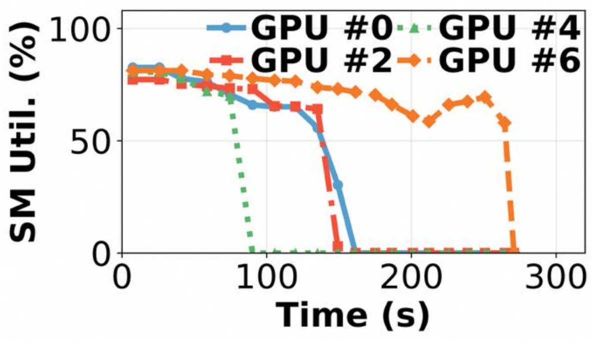
- **The rollout stage dominates runtime, accounting for approximately 70% of each training step.**

Table 1: Time breakdown of RL post-training. We train 14B models with a maximum length of 16k using veRL [45] and GRPO [43] with real-world datasets [20, 54] in three tasks.

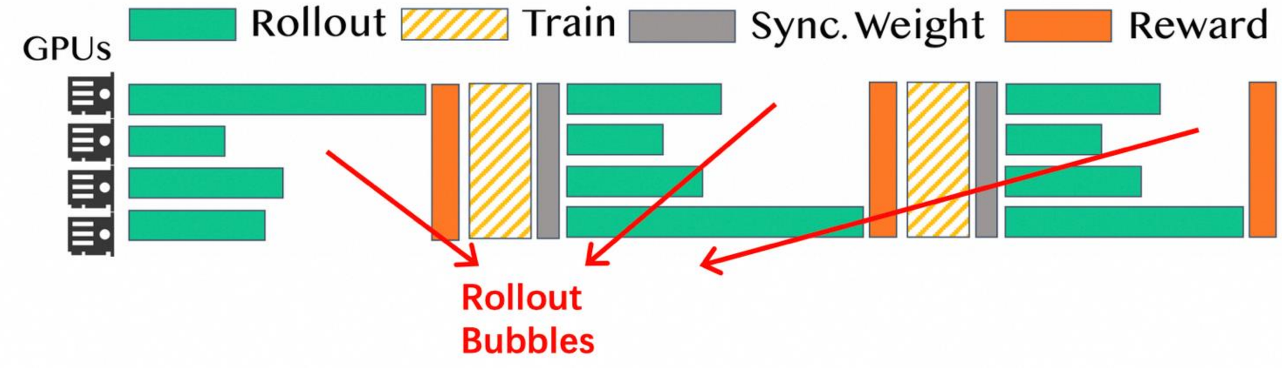
Task	Rollout	Reward	Training
Math	72%	5%	23%
Code	66%	13%	21%
LLM-as-a-Judge	71%	7%	22%

- **Long-tail responses lead to GPU bubbles, as most GPUs remain idle while awaiting their completion.**



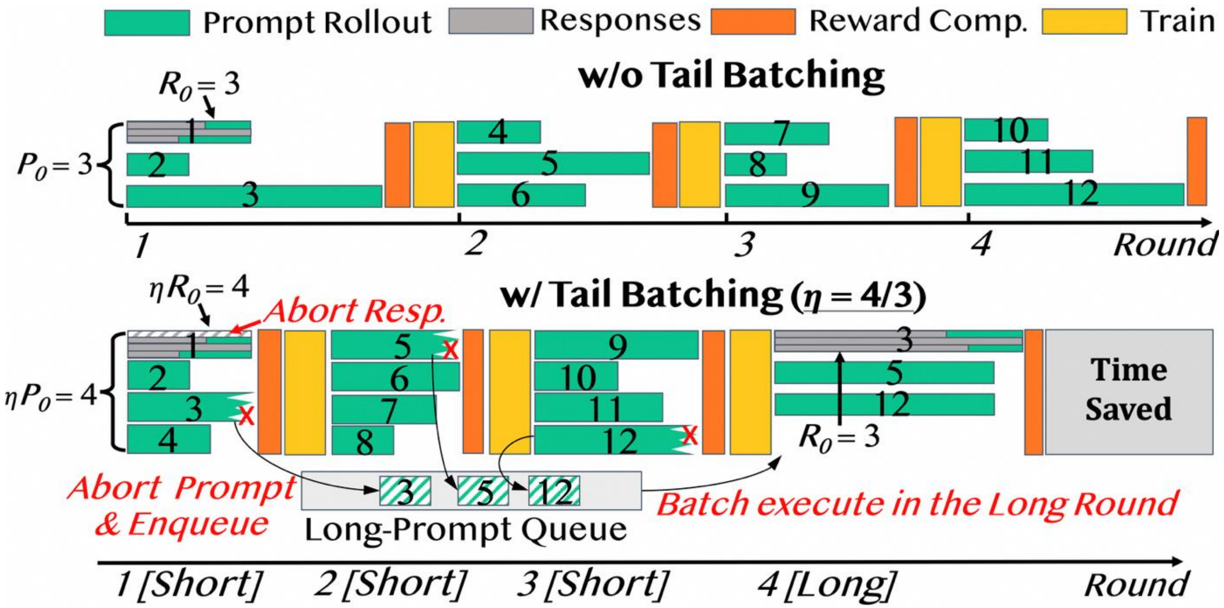


- **Synchronous training suffers from this rollout bubbles in each step, result in low utilization and increased**

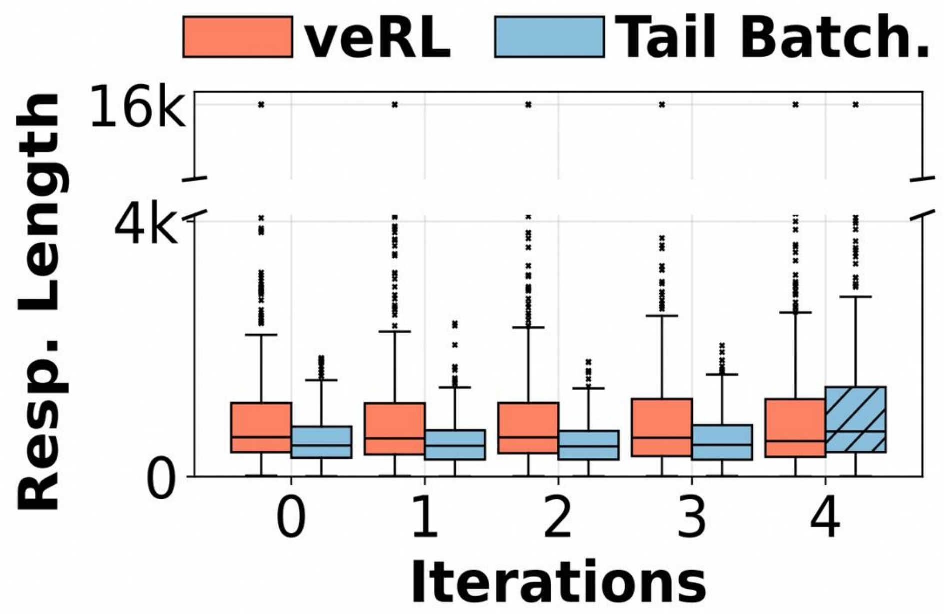


We propose tail batching to yield more balanced rollouts

- **The illustration of tail batching**
  - We perform redundant sampling to launch more requests during rollout stage in short rounds.
  - Long-prompt queue stores the aborted prompts.
  - We batch-execute rollouts for prompts in the queue during long rounds.

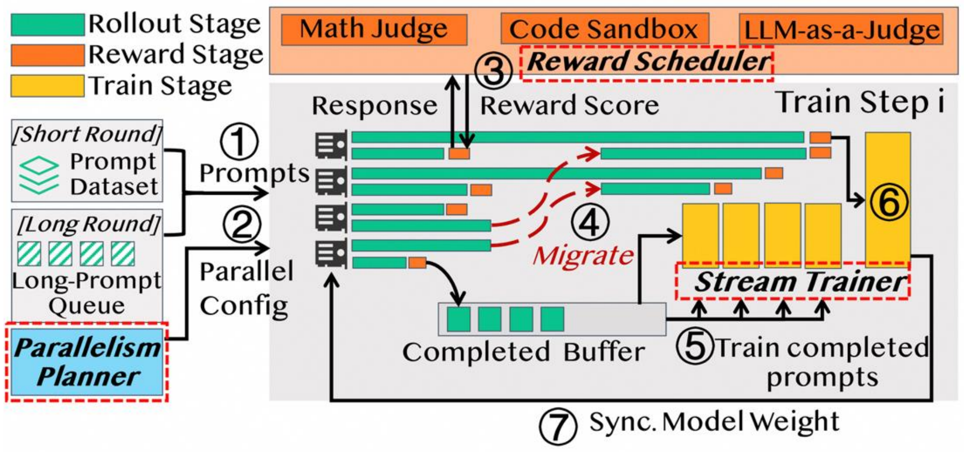


- **The effectiveness of tail batching**
  - Tail batching can yield shorter and more balanced responses in short rounds.

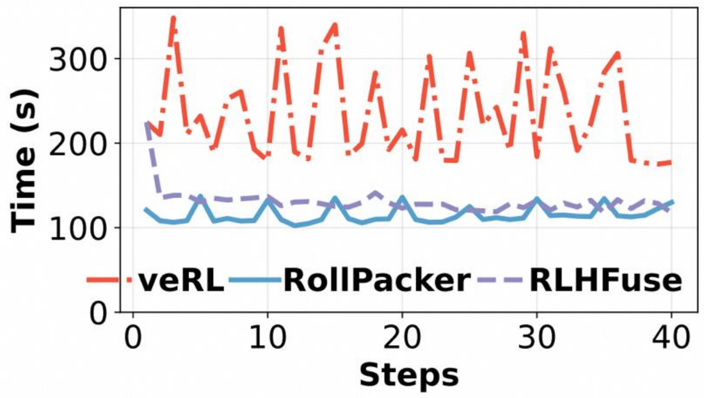


RollPacker introduces three system designs to unlock the potential of tail batching

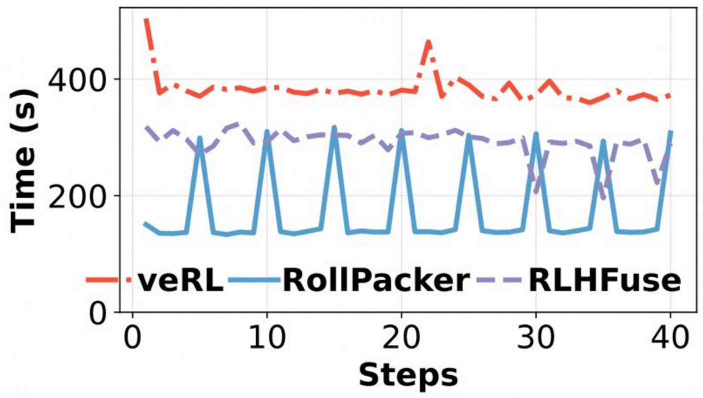
- **Key System Components**
  - **Parallelism Planner:** selects optimal TP size for short and long rounds.
  - **Reward Scheduler:** adaptively adjusts the resource budget to reduce the reward computation overhead.
  - **Stream Trainer:** elastically allocates the resources for rollout and training and overlaps the rollout and training stage.



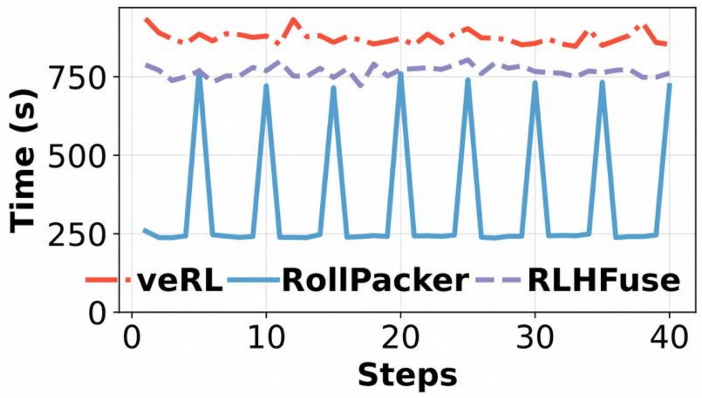
- **End-to-end Evaluation:** RollPacker significantly reduce end-to-end training overhead.



Qwen2.5-7B.



Qwen2.5-14B.



Qwen2.5-32B.

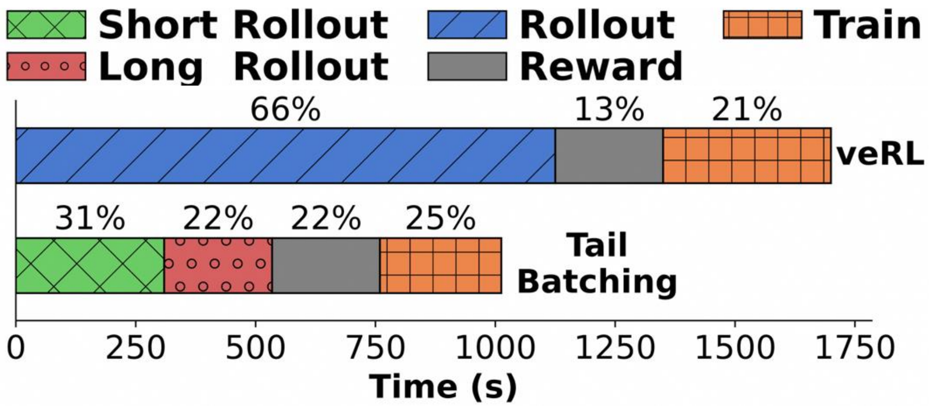
Reward Scheduler Alleviates Reward Computation Overhead

➢ Reward Computation Overhead Analysis

- **Observation: Reward overhead is non-negligible.**
  - Reward compute consists of up to 13% step time.
  - Tail batching enlarges this impact: rollout time is reduced, and reward compute time grows from 13% to 22%.

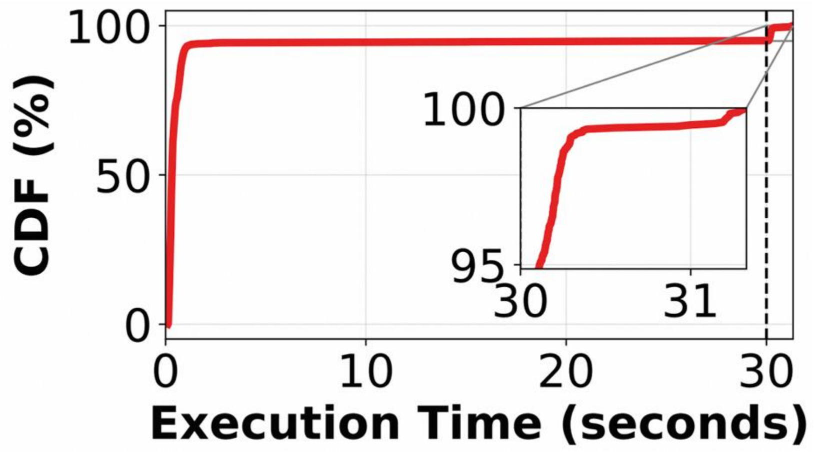
Task	Rollout	Reward	Training
Math	72%	5%	23%
Code	66%	13%	21%
LLM-as-a-Judge	71%	7%	22%

- **Insight**: Reward computation becomes a non-trivial contributor to the end-to-end latency.



Tailored Reward Computation Optimization: Code Sandbox

- **Code Sandbox Execution**
  - **Observation:** Some generated codes hit the fixed execution limit, resulting in zero rewards and wasted computational resources.
  - **Insight**: Terminating code executions that are likely to produce timeouts earlier to improve efficiency.



5% of code sandbox execution reach a 30s timeout.

- **Adaptive Timeout**
  - **Insight**: Correct responses usually terminate earlier than the fixed timeout.
  - **Solution**: Adaptive timeout for code execution. Track the max execution time of correct responses T_{anchor} and adjust timeout accordingly. Terminate executions which exceed the T_{anchor} and assign zero reward.

$$T_{\text{timeout}} = \min\left(\max(T_{\min}, \lambda T_{\text{anchor}}), T_{\max}\right)$$

The equation of adaptive timeout.

Easter Egg: ROCK – Reinforcement Open Construction Kit For Agentic RL

- **Robust Sandbox Isolation:** Independent, secure, fault-tolerant environments execution for each agent.
- **24/7 Health Monitoring:** Real-time diagnostics and proactive alerts, anticipating issues before they affect experiments.
- **Intelligent Load Balancing:** Automatic resource allocation ensures fair distribution and optimal utilization across all agents.
- **Automatic Fault Recovery**: Seconds-level restarts and seamless training continuation minimize downtime

QA Session

Join Roll Team: Internal referral code

WeChat Group

Scan and Star us: https://github.com/alibaba/ROLL