verl: Flexible and Scalable Reinforcement Learning Library for LLM Reasoning and Tool-Calling

Presenter: Haibin Lin, ByteDance Seed

Motivation:
Why is Large-Scale RL Important?

Large-Scale RL for Reasoning and Agents

Learning to reason with large-scale RL greatly boosts the performance of LLMs

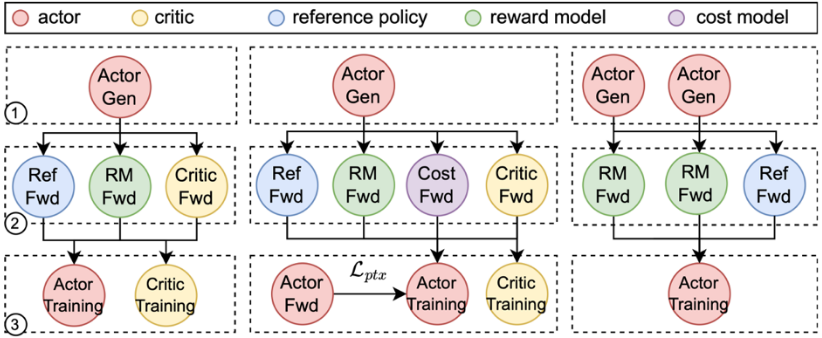
Model	Large-Scale RL?	AIME 2024	MATH 500	GPQA Diamond	Code Forces
GPT-4o (OpenAI 2024)	❌	44.6	60.3	50.6	>11.0%
o1 (OpenAI 2024)	✅	74.4	94.8	77.3	>89.0%

***Deep research** … was trained on **real-world tasks requiring browser and Python tool use**, using **the same reinforcement learning methods behind OpenAI o1**, our first reasoning model.*

– OpenAI Deep Research Blog, 2025

Challenge:
Why is Large-Scale RL Challenging?

RL as a Complex Dataflow



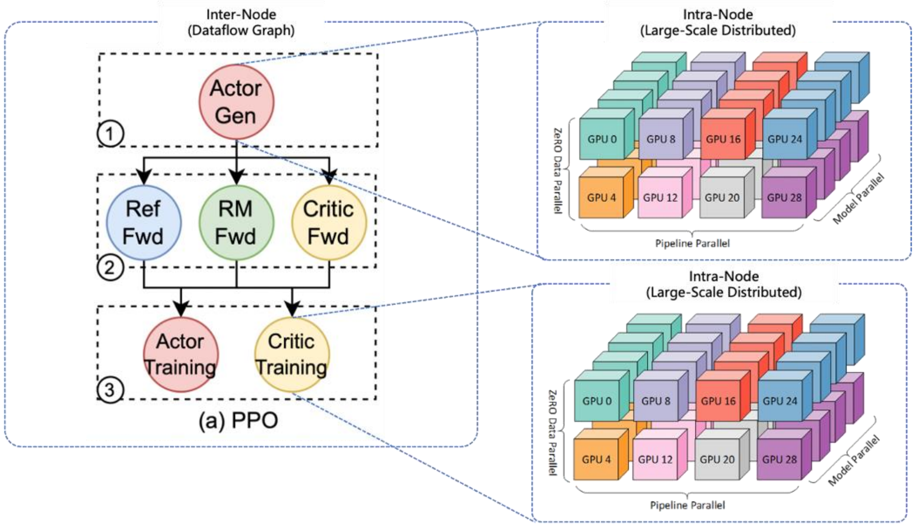
(a) PPO (b) Safe-RLHF (c) ReMax

Reinforcement Learning (RL) can be modelled as **complex dataflow graph** (Schaarschmidt et al. 2019; Liang et al. 2021; Sheng et al. 2025), consisting of:,

- **multiple models**: actor, critic, reference, reward model, etc.
- **multiple stages**: generating, preparing experiences, training
- **multiple workloads**: generation, inference, training

HybridFlow, Sheng et al., 2024

RL with LLMs is Large-Scale Distributed Dataflow



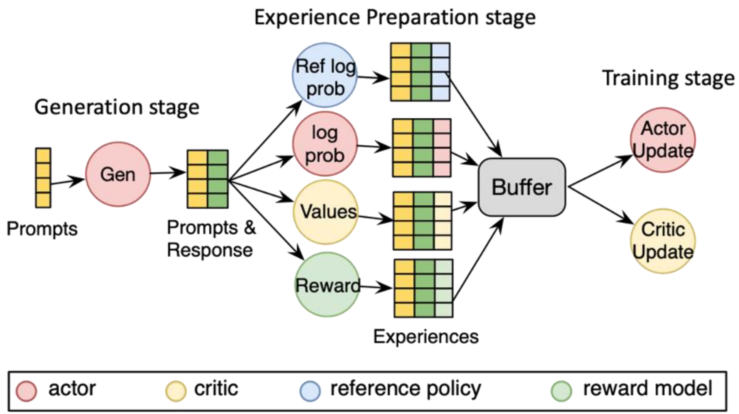
each **operator** in the RL dataflow = a large-scale **distributed** computing workload

constraints: computation dependency

Why verl for RL with LLMs?

Flexible and Efficient!

Flexibility in Programming: “Single-Controller”




```python
for prompts in dataloader:
    # Stage 1: Generation
    batch = actor.generate_sequences(prompts)
    # Stage 2: Experience Preparation
    batch = reward.compute_reward(batch)
    batch = reference.compute_log_prob(batch)
    batch = critic.compute_values(batch)
    batch = compute_advantage(batch, "gae")
    # Stage 3: Training
    critic.update_critic(batch)
    actor.update_actor(batch)
```

- Programming interface based on the “**single-controller**” paradigm
- RL algorithm core logic in **a few lines of code**!
- Diverse RL algorithms supported: PPO, GRPO, RLOO, GSPO, PRIME, DAPO, etc.

Efficiency: “Multi-Controller”

verl is efficient for intra-operator with the **“multi-controller”** paradigm and features like:

Training Backends:

- FSDP
- FSDP2
- Megatron

Generation Backends:

- vLLM
- SGLang
- …

Parallelism Algorithms:

- Data Parallelism
- Tensor Parallelism
- Pipeline Parallelism
- Context / Sequence Parallelism
- …

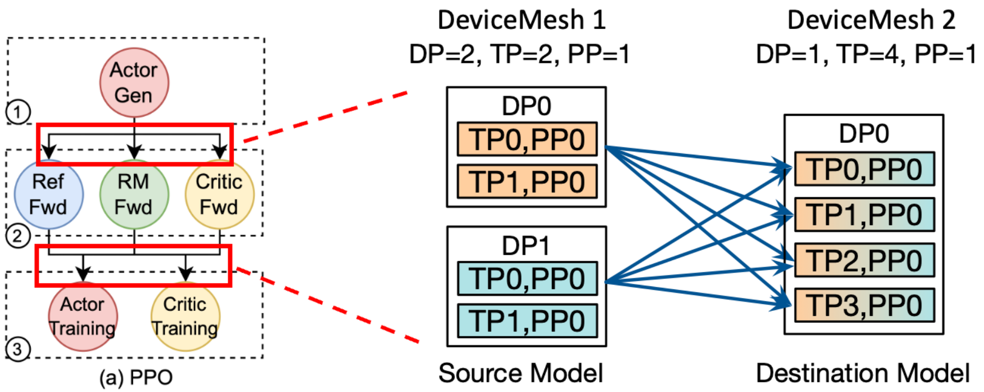
Efficient Kernels:

- Flash Attention 2
- Torch Compile
- Liger Kernel
- …

Efficiency: “Hybrid Engine” for synchronous RL

verl is efficient for inter-operator with the **“hybrid engine”** paradigm, utilizing:

- **offloading & reloading** enables fully utilizing the GPU memory
- **resharding** enables switching for the optimal parallelism strategy
- FSDP trainer & vLLM worker live in the **same process,** reducing mem fragmentation



HybridFlow, Sheng et al., 2024

Open-Source Community: Impactful and Inclusive

So far, verl has gained:

- 11.9k stars
- 2k forks
- 1.5k PRs
- 300+ contributors

Waiting for your participation!

Capabilities

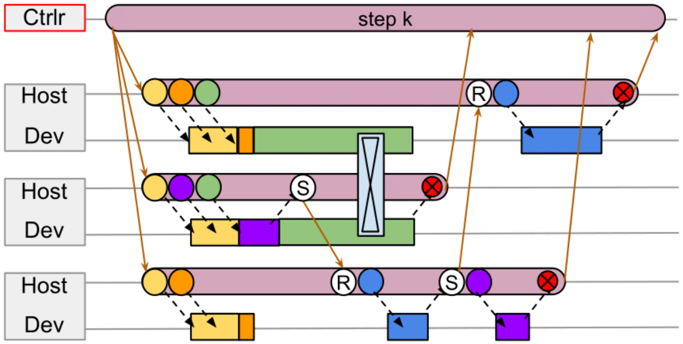
- Multi-modal (image/video)
- Large MoE RL
- Multi-GPU LoRA support
- Sandbox/search tools
- Recipes: DAPO, retool

Paradigm behind verl: HybridFlow

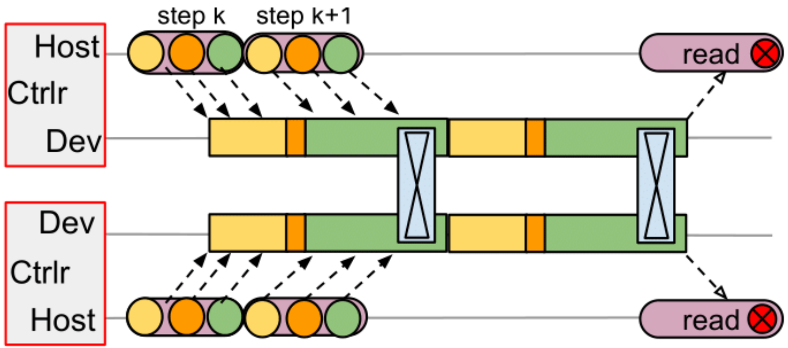
Sheng et al. 2024

Background: Single-Controller vs. Multi-Controller

Single-Controller (MPMD):
A centralized controller manages all the workers, running different programs

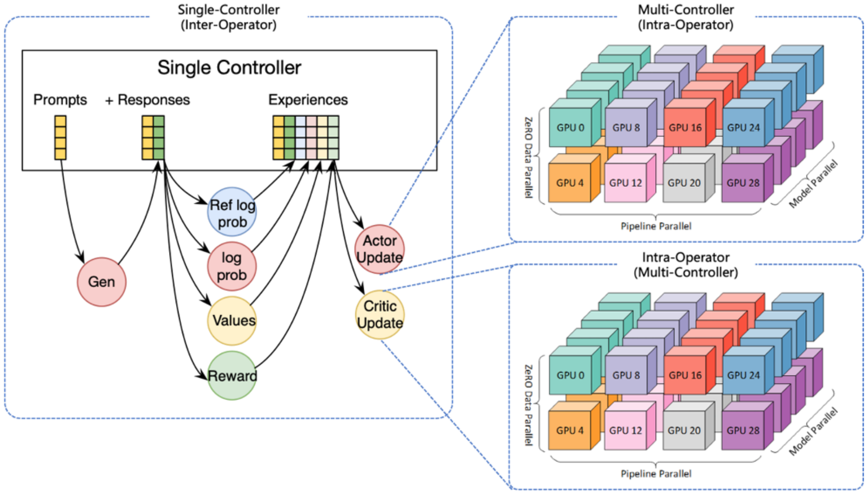


Multi-Controller (SPMD):
Each worker has its own controller, running the same program with different data



Pathways, Barham et al., 2022

New Paradigm: Hybrid-Controller!



- Hybrid-Controller = Single-Controller + N x Multi-Controller
- In the hybrid-controller, a single-controller manages multiple multi-controllers to process the dataflow

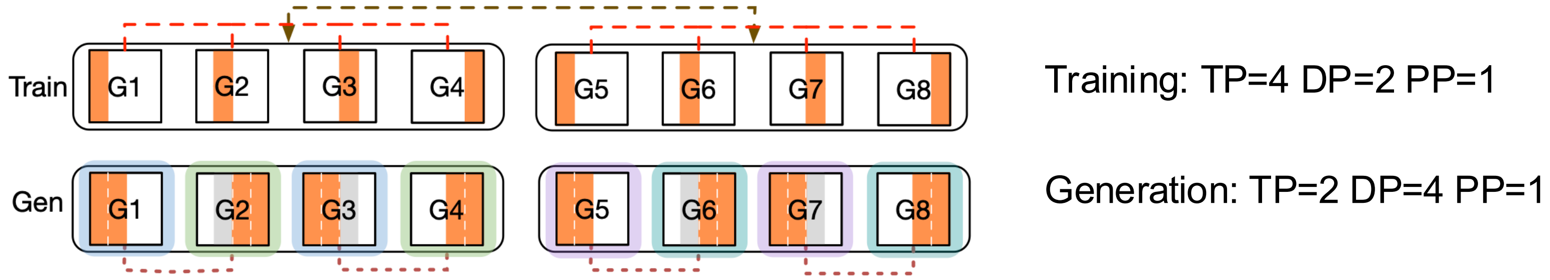
Controller Data Dispatch/Collection in verl

```
for prompts in dataloader:
    # Stage 1: Generation
    batch = actor.generate_sequences(prompts)
    # Stage 2: Experience Preparation
    batch = reward.compute_reward(batch)
    batch = reference.compute_log_prob(batch)
    batch = critic.compute_values(batch)
    batch = compute_advantage(batch, "gae")
    # Stage 3: Training
    critic.update_critic(batch)
    actor.update_actor(batch)
```

```
class CriticWorker(3DParallelWorker):
  @register(dispatch_mode=3D_PROTO)
  def compute_values(self, batch: DataProto):
      values = self.critic.forward(batch)
      batch.update(values=values)
# ...
class ActorWorker(3DParallelWorker):
  @register(dispatch_mode=3D_PROTO)
  def update_actor(self, batch: DataProto):
      loss = self.actor(batch)
      loss.backward()
```

- Each call in the single-controller (e.g. critic.compute_values, actor.update_actor) is an RPC to a multi-controller worker group
- The register decorator utility manages the distributed data transfer, which also makes multi-controller programming easier

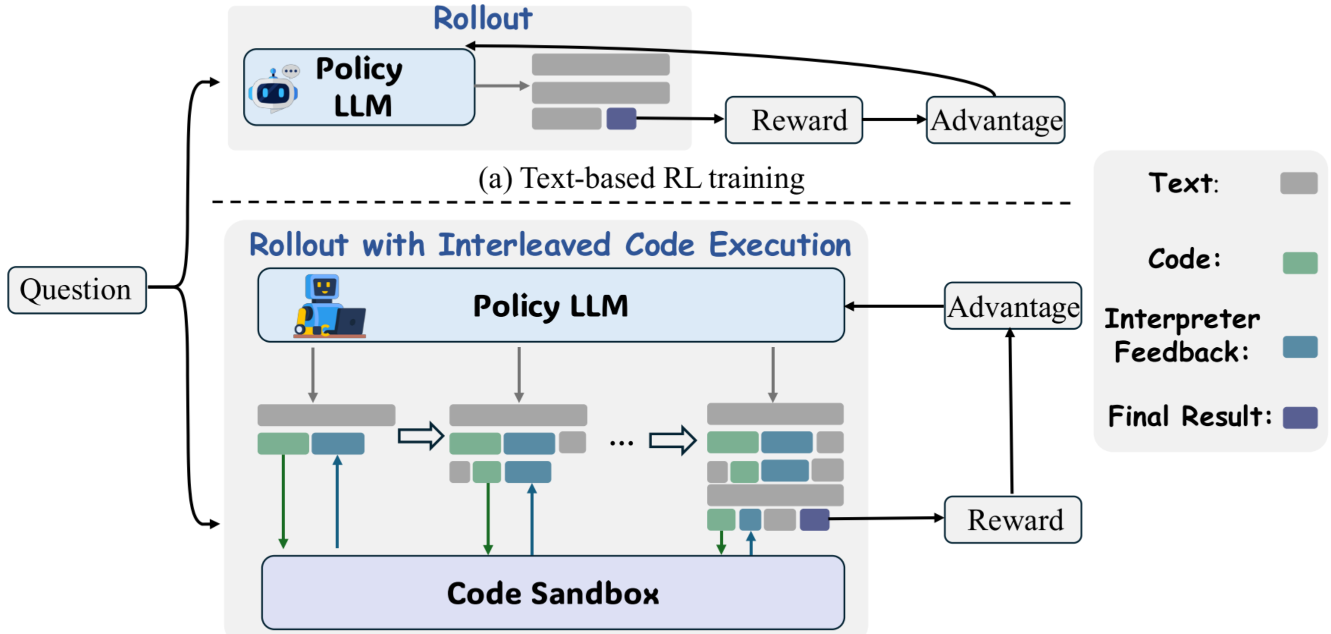
Train/Generation Weight Transfer in verl



- Optimal sharding for training/generation can be different
- Weight binding can be generalized as a transformation of DTensor between two device meshes under the same world via NCCL+RDMA
- Per-tensor/bucket transfer to avoid OOMs

```
for shared_training_param in model.parameters():
    train_full_param = shared_param.full_tensor()
    infer_sharded_param = redistribute(train_full_param, infer_device_mesh)
```

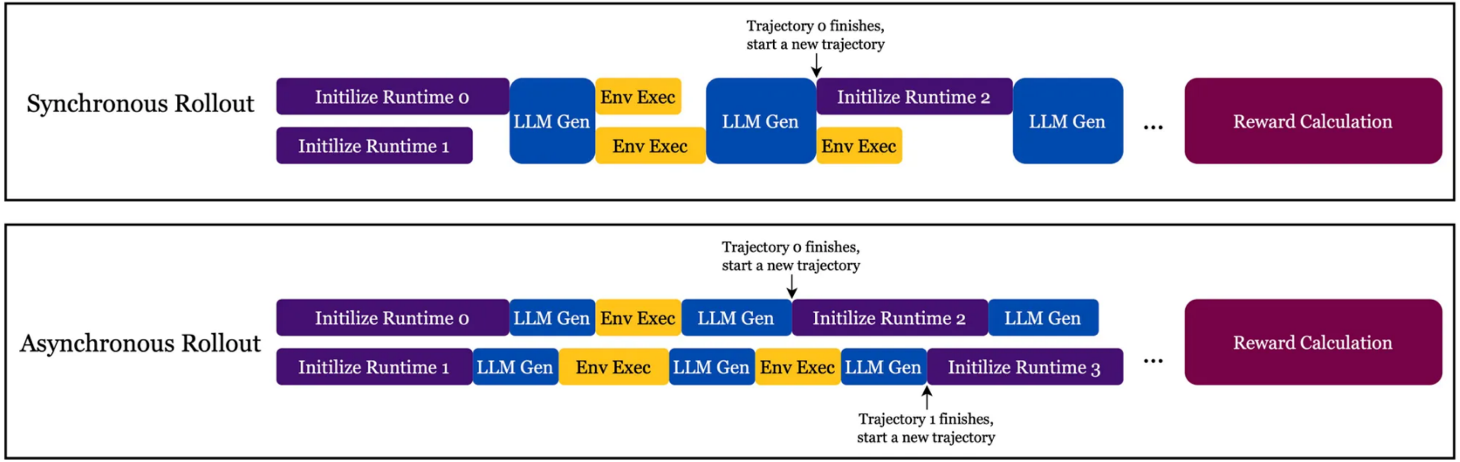
Approaching Agentic RL



(a) Text-based RL training

(b) ReTool RL training

Async Engine for Multi-Turn Rollout

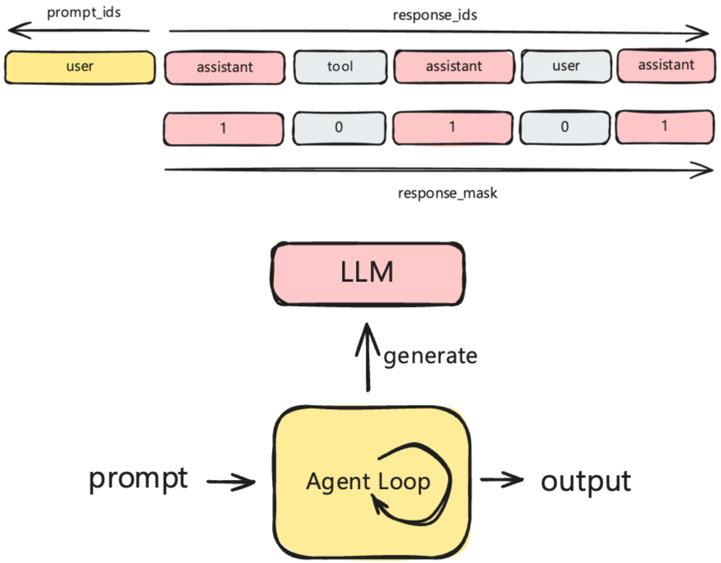


- Synchronous Engine: returns all the outputs in the batch at the same time
- Asynchronous Server: returns each output as soon as it is ready

https://novasky-ai.notion.site/skyrl-v0

Token-in-token-out Agent Loop Interface

Given one prompt, run a user defined loop with multi-turn/tool calling trajectories.
Token ids are used for server generation input / output.




```python
class DemoLoop(AgentLoopBase):

    async def run(self, messages: list[dict[str, Any]], ...) -> AgentLoopOutput:
        prompt_ids = await self.loop.run_in_executor(None,
            lambda: tokenizer.apply_chat_template(messages, ..., tokenize=True)
        )
        num_turns = 0
        while not is_done(prompt_ids, num_turns):
            response_ids = await self.rollout_server.generate(
                request_id=request_id, prompt_ids=prompt_ids, ...
            )
            prompt_ids += response_ids
            tool_response_ids = await call_tool(response_ids)
            prompt_ids += tool_response_ids
            num_turns += 1
            response_mask += ...

            output = AgentLoopOutput(
                prompt_ids=prompt_ids,
                response_ids=response_ids,
                response_mask=response_mask,
                num_turns=num_turns,
            )
            return output
```

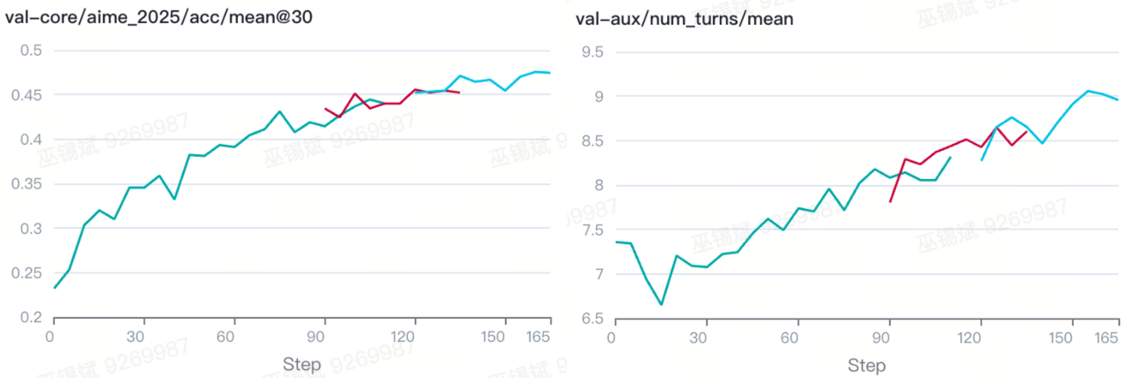
Agent Loop & Reproducible Recipes

- client-server mode: decoupled inference engine and agent loop
- parallel async loop running: request/prompt level async execution
- KV cache reuse: multi-turn requests sent to the same inference server

ReTool Recipe

AIME 0.6 with 32b model

verl/recipe/retool/



Recent Updates
& Roadmap

Efficient RL with Huge MoE like DeepSeek-V3-671B

verl is working on supporting efficient RL training for **giant MoE like DeepSeek-V3-671B**, based on the following features:

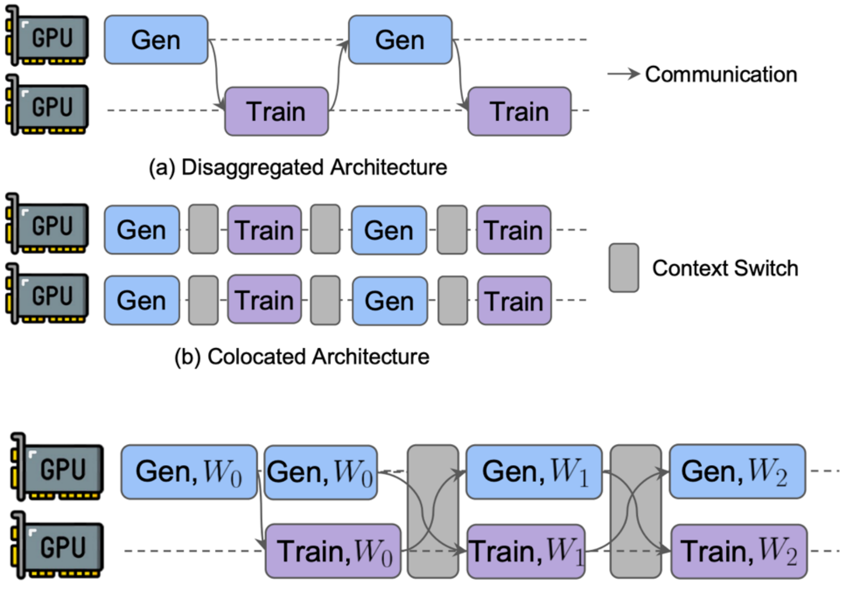
- Runnable with 256 H100 GPUs
- Training: **MoE models** based on Megatron, ~0.12 MFU
- Inference: **Multi-node** tensor parallel inference
- Verified reward curve on orz57k & proprietary datasets from community
- Planned: accelerate rollout performance (e.g. fp8)

For more details, please check issue tracker #1033.

Credit to NVIDIA, Amazon, RedNote, etc

Disaggregated Async Trainer

- Hybrid controller allows flexible device placement (collocate/disaggregated)
- One-step async pipeline to avoid onload/offload overhead & bubbles
- verl/recipe/one_step_off_policy
  ~1.2x speedup compared to sync trainer
- Long-tail generation problems
- Towards fully-async pipeline
  Interruptible generation PR 2200



AReaL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning
StreamRL: Scalable, Heterogeneous, and Elastic RL for LLMs with Disaggregated Stream Generation

Credit to Meituan

Q3 Roadmap

- Modular design: composable model engines with better abstraction
  - Algorithm agnostic engine abstraction: FSDP2, Megatron, and more
- Partial rollout & fully-async training pipeline (AReal, Kimi, 2025)
- Rollout performance optimizations (fp8)
- Agentic RL recipes (e.g. SWE-bench)

Github roadmap issue tracker #2388.