# verl: Hybrid Controller-based RLHF System

Bytedance verl team

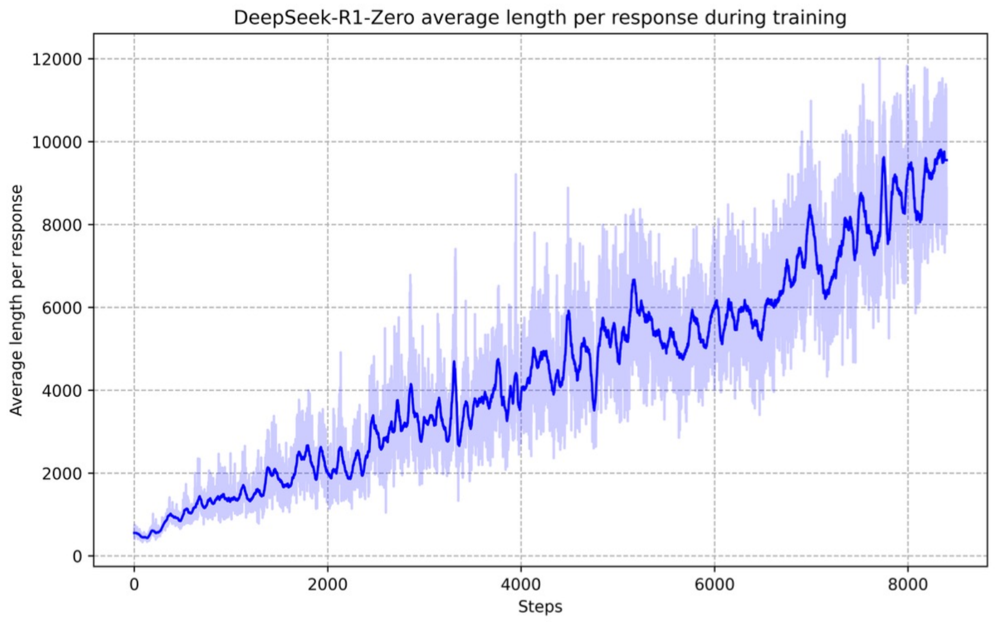
Mar 16, 2025

Beijing

# RL-LLM Infra Challenge

Megatron-LM

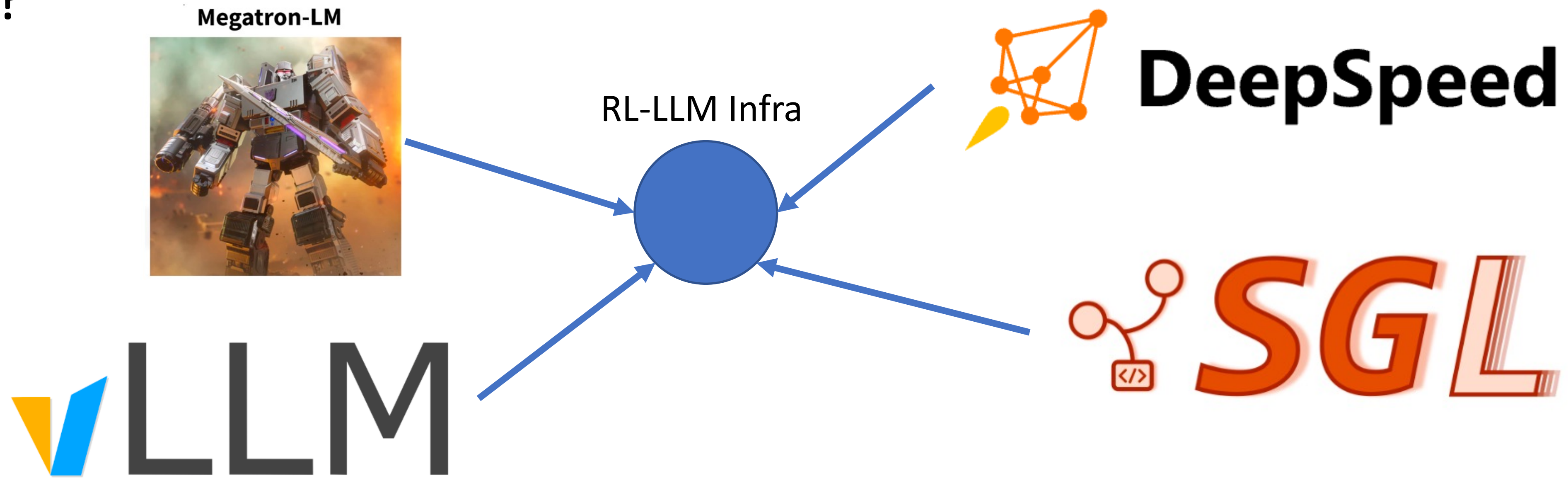
- The abstraction of data parallel programming abstraction (DeepSpeed/FSDP Zero3) is not sufficient (e.g., accelerate)
- Model sizes are growing
  - DeepSeek-r1 671B
  - Llama 405b
  - Qwen 72b
- Sequence length are growing
  - From 10k ~ 1M
- nD parallelism is necessary
  - TP/PP/SP/CP/EP
  - Megatron-LM



Megatron-LM: https://github.com/NVIDIA/Megatron-LM
DeepSeek-R1: https://github.com/deepseek-ai/DeepSeek-R1

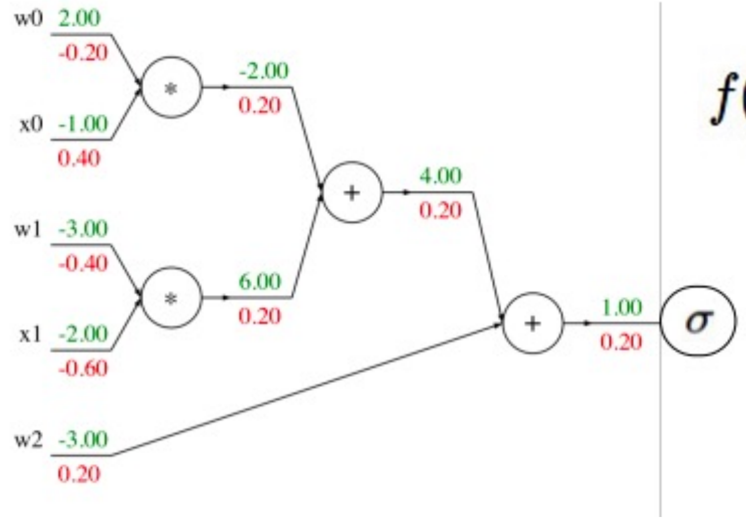
# RL-LLM Infra Challenge (Cont')

- RL consists of training, inference, autoregressive generation and tool calling
- Efficiently combining them into one system is challenging
- How to provide researchers correct abstractions to implement new ideas?

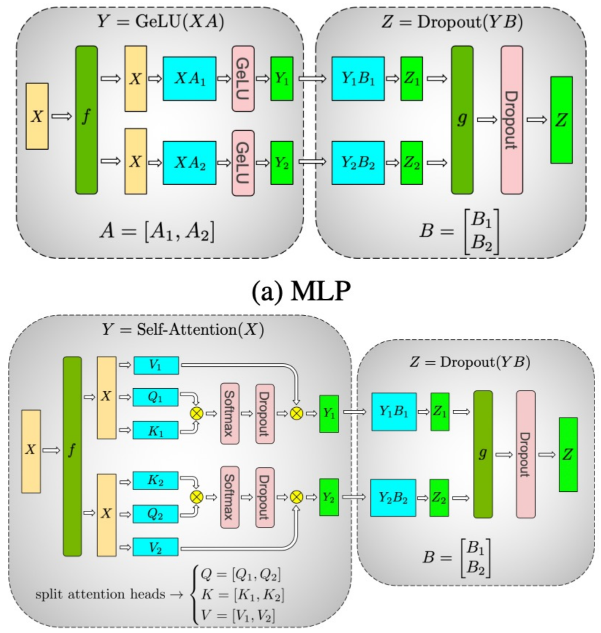


# LLM training is Data Flow

- LLM training is just distributed neural network
  - Computation graph including numerical ops and communication ops
- SPMD for high performance
  - Each process runs the same program with different data



$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



(a) MLP

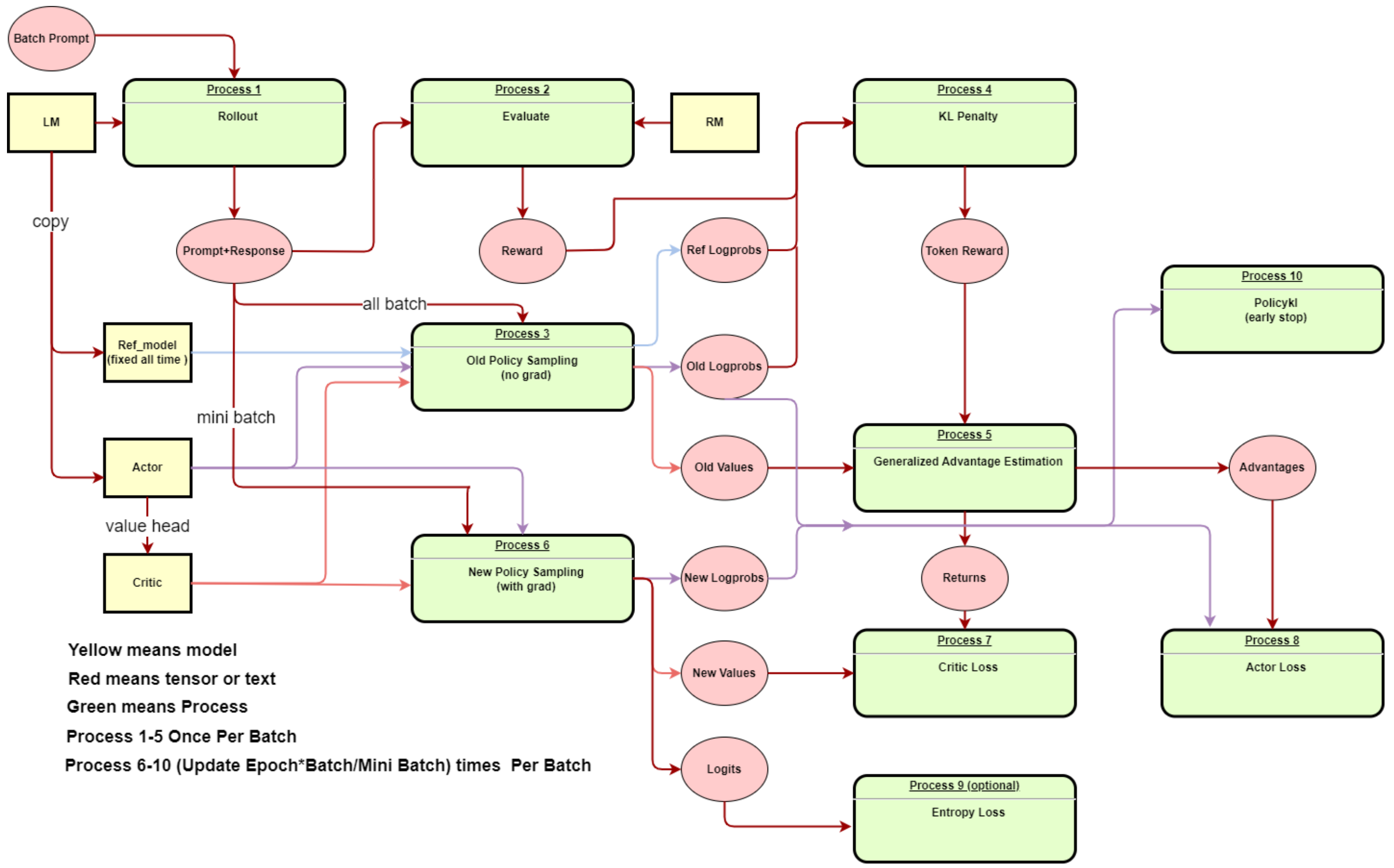
(b) Self-Attention

- https://cs231n.stanford.edu/slides/2024/lecture_4.pdf
- https://arxiv.org/pdf/1909.08053

https://zhuanlan.zhihu.com/p/635757674

# RL is Data Flow

- RL data flow is a DAG
- RL data flow is single process



Yellow means model
Red means tensor or text
Green means Process
Process 1-5 Once Per Batch
Process 6-10 (Update Epoch*Batch/Mini Batch) times Per Batch

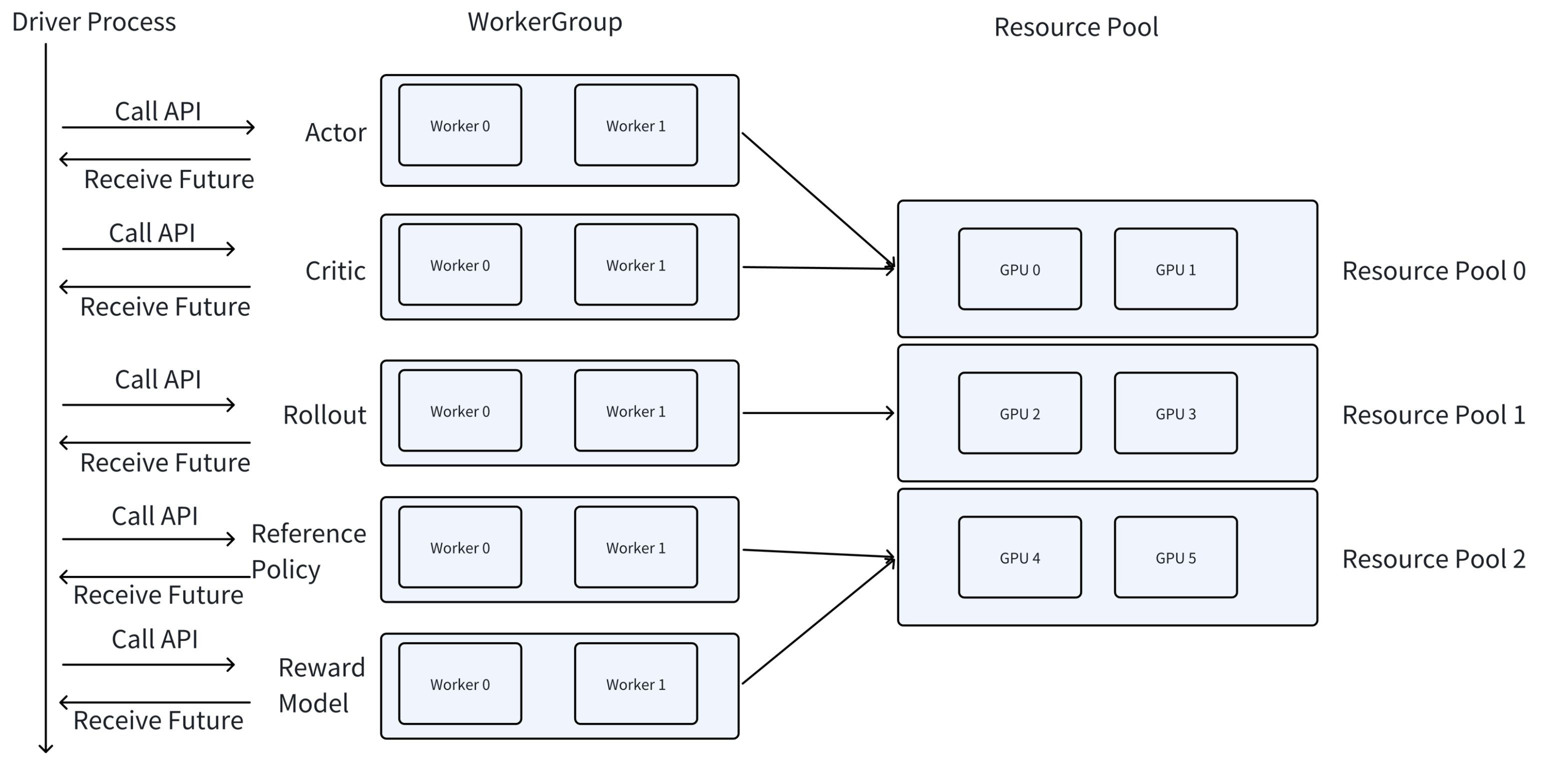
# HybridFlow Programming Abstraction

- RL-LLM infra is a two-level data flow problem
  - Algorithm-level **control** flow
  - Low-level **computation** flow
- Design Choice - Existing works
  - Integrate control flow with computation flow
  - Control flow becomes multi-process
    - Control flow has to be aware of distributed information
  - Inflexible when the control flow changes

# HybridFlow Programming Abstraction

- Separate control flow and computation flow
  - Single controller for control flow, SPMD for computation flow
  - Driver process runs the control flow. WorkerGroup runs SPMD computation flow

- Worker Groups mapped onto the same resource pool share GPU

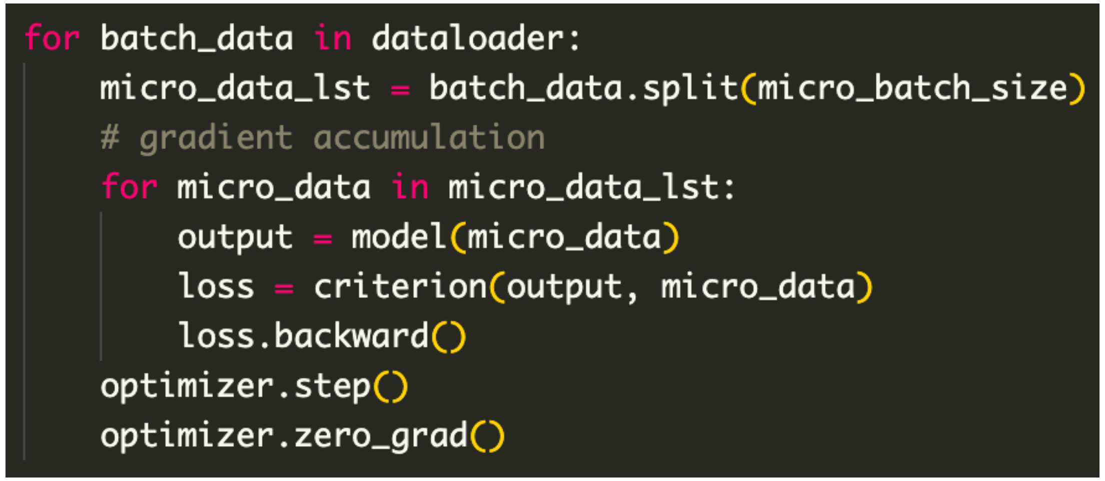


## Demo - PPO

- How to program using HybridFlow
  - Step 1: Implement the computation engine via Worker
  - Step 2: Instantiate Resource Pool and WorkerGroup using Worker
  - Step 3: Implement driver using WorkerGroup
- Components
  - Actor
  - Rollout
  - Reference Policy
  - Critic
  - Reward Model

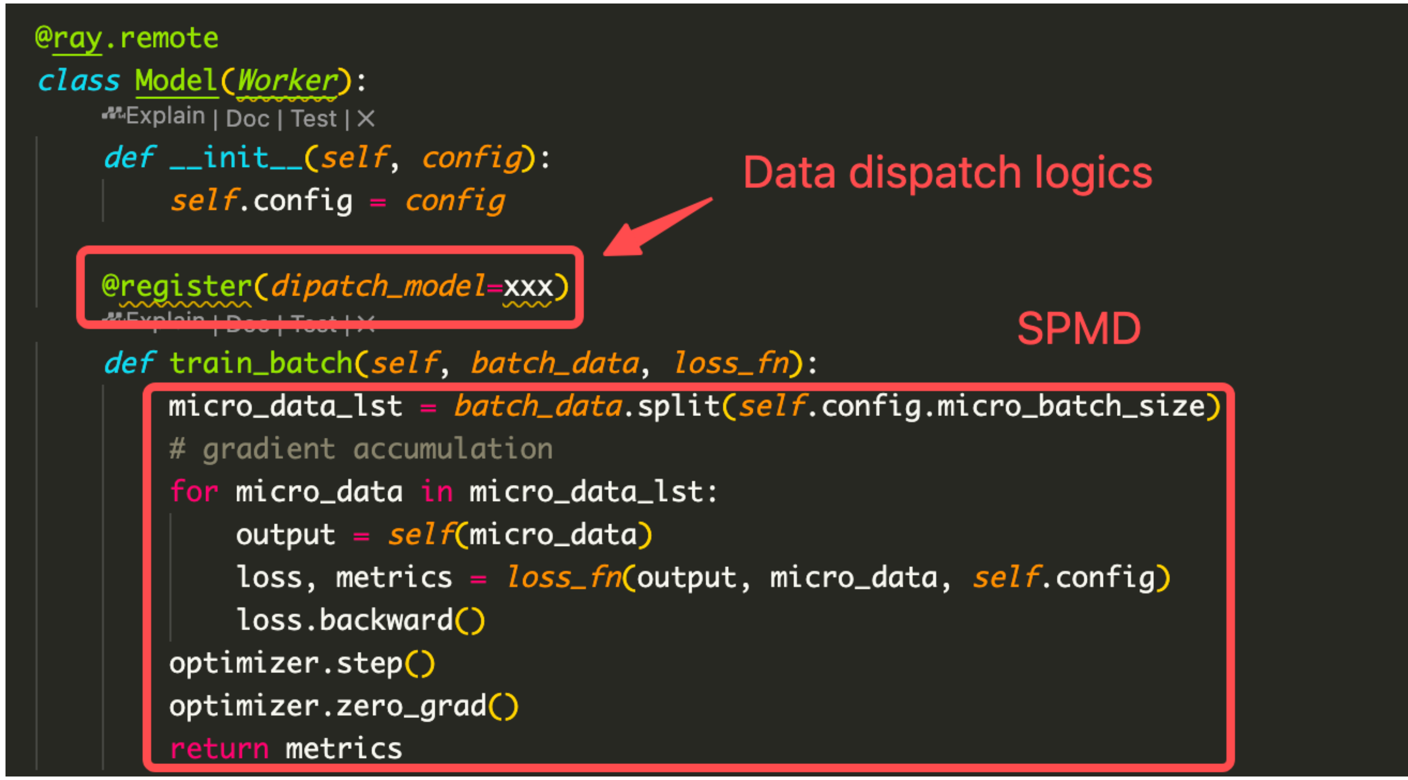
# HybridFlow Programming Step 1

- Implement SPMD computation inside a method of Worker
- Decorate the method with appropriate dispatch method



```
for batch_data in dataloader:
    micro_data_lst = batch_data.split(micro_batch_size)
    # gradient accumulation
    for micro_data in micro_data_lst:
        output = model(micro_data)
        loss = criterion(output, micro_data)
        loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Classic training loops

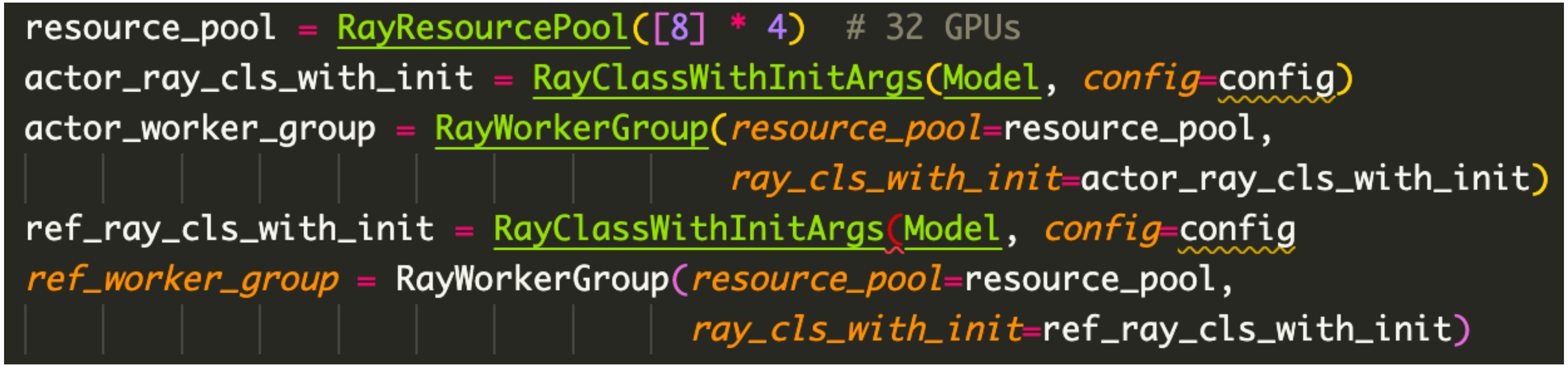


```
@ray.remote
class Model(Worker):
    def __init__(self, config):
        self.config = config

    @register(dipatch_model=xxx)
    def train_batch(self, batch_data, loss_fn):
        micro_data_lst = batch_data.split(self.config.micro_batch_size)
        # gradient accumulation
        for micro_data in micro_data_lst:
            output = self(micro_data)
            loss, metrics = loss_fn(output, micro_data, self.config)
            loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        return metrics
```

# HybridFlow Programming Step 2

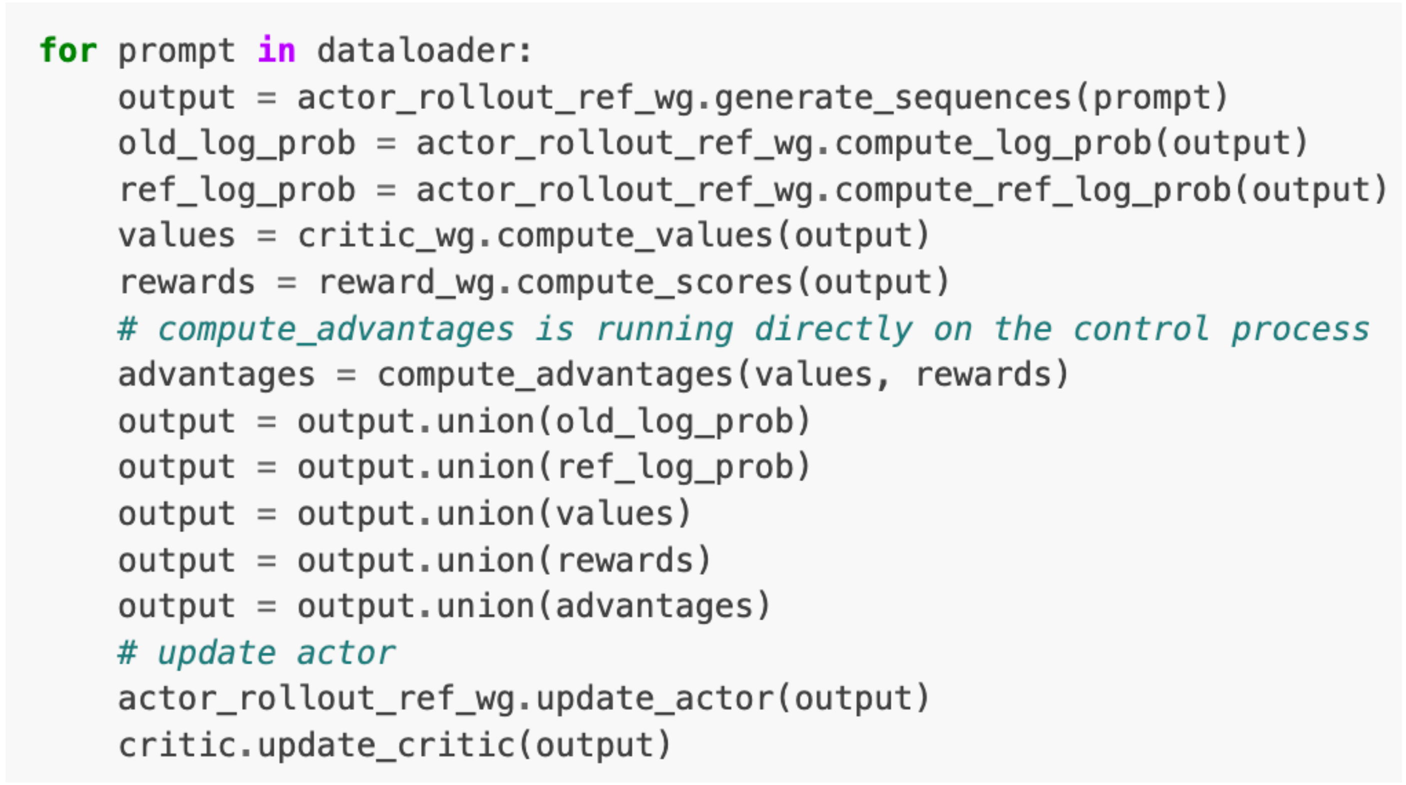
- Instantiate Resource Pool and map WorkerGroups
- For example:
  - 32 GPUs
  - Actor and Reference policy colocate



```
resource_pool = RayResourcePool([8] * 4)  # 32 GPUs
actor_ray_cls_with_init = RayClassWithInitArgs(Model, config=config)
actor_worker_group = RayWorkerGroup(resource_pool=resource_pool,
                                    ray_cls_with_init=actor_ray_cls_with_init)
ref_ray_cls_with_init = RayClassWithInitArgs(Model, config=config
ref_worker_group = RayWorkerGroup(resource_pool=resource_pool,
                                  ray_cls_with_init=ref_ray_cls_with_init)
```

## HybridFlow Programming Step 3

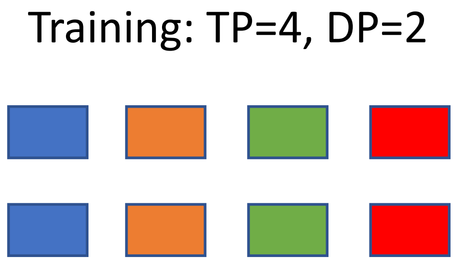
- Using the worker groups to construct the algorithm
- With hybridflow programming abstraction, the controller is just a single process program

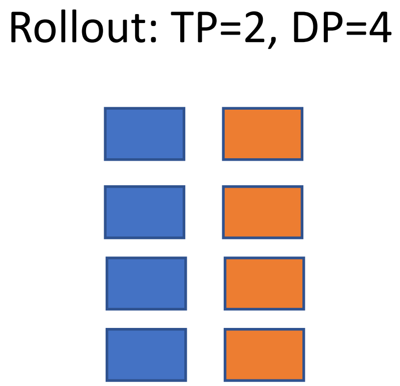


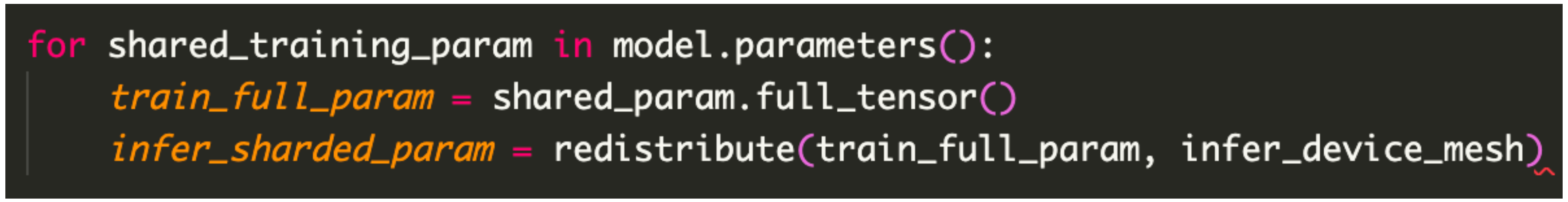
```
for prompt in dataloader:
    output = actor_rollout_ref_wg.generate_sequences(prompt)
    old_log_prob = actor_rollout_ref_wg.compute_log_prob(output)
    ref_log_prob = actor_rollout_ref_wg.compute_ref_log_prob(output)
    values = critic_wg.compute_values(output)
    rewards = reward_wg.compute_scores(output)
    # compute_advantages is running directly on the control process
    advantages = compute_advantages(values, rewards)
    output = output.union(old_log_prob)
    output = output.union(ref_log_prob)
    output = output.union(values)
    output = output.union(rewards)
    output = output.union(advantages)
    # update actor
    actor_rollout_ref_wg.update_actor(output)
    critic.update_critic(output)
```

# HybridEngine

- The optimal sharding of training and rollout can be different
- Weight binding can be generalized as a transformation of **DTensor** between two device meshes under the same world







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

# Awesome works using verl

- TinyZero: a reproduction of **DeepSeek R1 Zero** recipe for reasoning tasks
- PRIME: Process reinforcement through implicit rewards
- RAGEN: a general-purpose reasoning **agent** training framework
- Logic-RL: a reproduction of DeepSeek R1 Zero on 2K Tiny Logic Puzzle Dataset.
- SkyThought: RL training for Sky-T1-7B by NovaSky AI team.
- deepscaler: iterative context scaling with GRPO
- critic-rl: LLM critics for code generation
- Easy-R1: **Multi-modal** RL training framework
- self-rewarding-reasoning-LLM: self-rewarding and correction with **generative reward models**
- Search-R1: RL with reasoning and **searching (tool-call)** interleaved LLMs
- Code-R1: Reproducing R1 for **Code** with Reliable Rewards
- DQO: Enhancing multi-Step reasoning abilities of language models through direct Q-function optimization
- FIRE: Flaming-hot initiation with regular execution sampling for large language models
- ReSearch: Learning to **Re**ason with **Search** for LLMs via Reinforcement Learning
- DeepRetrieval: Let LLMs learn to **search** and **retrieve** desirable docs with RL
- cognitive-behaviors: Cognitive Behaviors that Enable Self-Improving Reasoners, or, Four Habits of Highly Effective STaRs