verl 代码

Author

commit 5a66ed26497a

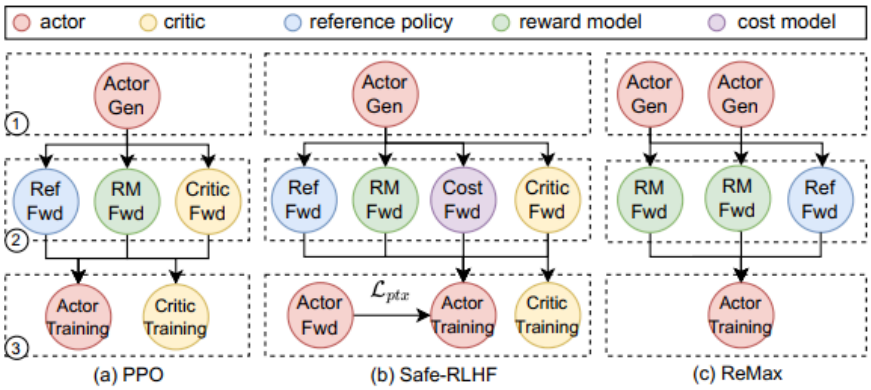


Figure 1. Dataflow graph of 3 RLHF algorithms [19, 43, 55]. Stage ①, ②, ③ represent Generation, Preparation, and Training, respectively.

1 verl/trainer/main_ppo.py

1) verl/trainer/main_ppo.py

```
70
71      role_worker_mapping = {
72          Role.ActorRollout: ray.remote(ActorRolloutRefWorker),
73          Role.Critic: ray.remote(CriticWorker),
74          Role.RefPolicy: ray.remote(ActorRolloutRefWorker)
75      }
76
77      global_pool_id = 'global_pool'
78      resource_pool_spec = {
79          global_pool_id: [config.trainer.n_gpus_per_node] * config.trainer.nnodes,
80      }
81      mapping = {
82          Role.ActorRollout: global_pool_id,
83          Role.Critic: global_pool_id,
84          Role.RefPolicy: global_pool_id,
85      }
86
87      # we should adopt a multi-source reward function here
88      # - for rule-based rm, we directly call a reward score
89      # - for model-based rm, we call a model
90      # - for code related prompt, we send to a sandbox if there are test cases
91      # - finally, we combine all the rewards together
92      # - The reward type depends on the tag of the data
93      if config.reward_model.enable:
94          if config.reward_model.strategy == 'fsdp':
95              from verl.workers.fsdp_workers import RewardModelWorker
96          elif config.reward_model.strategy == 'megatron':
97              from verl.workers.megatron_workers import RewardModelWorker
98          else:
99              raise NotImplementedError
100         role_worker_mapping[Role.RewardModel] = ray.remote(RewardModelWorker)
101         mapping[Role.RewardModel] = global_pool_id
```

这里初始化role_worker_mapping, 对应的worker是哪个

问题：ray.remote的作用是什么，ActorRollRefWorker都是class

`ray.remote(A)` 是 **Ray** 框架中的一个重要操作，它的作用是将类 `A` 标记为 **远程类**，使得 `A` 的实例可以在 Ray 的分布式环境中运行。具体来说，`ray.remote(A)` 会返回一个 **远程类** 的引用，你可以通过这个引用在 Ray 集群中创建 `A` 的实例，并调用其方法。

---

1. Ray 的基本概念

- **Ray** 是一个分布式计算框架，用于在集群中并行化和分布式化 Python 代码。
- **远程函数**：通过 `ray.remote` 装饰的函数可以在 Ray 集群中远程执行。
- **远程类**：通过 `ray.remote` 装饰的类可以在 Ray 集群中远程实例化，其方法可以远程调用。

---

2. `ray.remote(A)` 的作用

- **将类 `A` 标记为远程类**：
  - `ray.remote(A)` 返回一个远程类的引用，这个引用可以用来在 Ray 集群中创建 `A` 的实例。
  - 例如：
    这里
    `RemoteA` 是一个远程类的引用。

```
RemoteA = ray.remote(A)
```

- **创建远程实例**：
  - 通过 `RemoteA.remote()` 可以在 Ray 集群中创建 `A` 的实例。
  - 例如：
    这里
    `a` 是一个远程对象的引用，实际的对象在 Ray 集群中运行。

    ```
    a = RemoteA.remote()
    ```

- **调用远程方法**：
  - 通过 `a.method_name.remote()` 可以调用远程对象的方法。
  - 例如：
    这里
    `result` 是一个 `ObjectRef`，表示远程调用的结果。

    ```
    result = a.some_method.remote(args)
    ```

---

3. 代码示例

以下是一个完整的示例，展示如何使用 `ray.remote(A)`：

```
import ray

# 初始化 Ray
ray.init()

# 定义一个普通类
class A:
    def __init__(self, value):
        self.value = value

    def add(self, x):
        return self.value + x

# 将类 A 标记为远程类
RemoteA = ray.remote(A)

# 在 Ray 集群中创建 A 的实例
a = RemoteA.remote(value=10)

# 调用远程方法
result = a.add.remote(x=5)
```

```
# 获取结果
print(ray.get(result))  # 输出: 15
```

4. `ray.remote(A)` 的底层原理

- **分布式对象存储**：
  - Ray 会将远程类的实例存储在分布式对象存储中，每个实例有一个唯一的 `ObjectRef`。
- **任务调度**：
  - 当调用远程方法时，Ray 会将任务调度到合适的节点上执行。
- **异步执行**：
  - 远程方法的调用是异步的，返回一个 `ObjectRef`，可以通过 `ray.get` 获取结果。

5. 使用场景

- **分布式计算**：
  - 如果 `A` 是一个计算密集型类，可以通过 `ray.remote(A)` 将其分布到多个节点上运行。
- **状态共享**：
  - 如果 `A` 是一个状态类，可以通过 `ray.remote(A)` 在多个任务之间共享状态。
- **并行化**：
  - 如果 `A` 的方法可以并行化，可以通过 `ray.remote(A)` 实现并行执行。

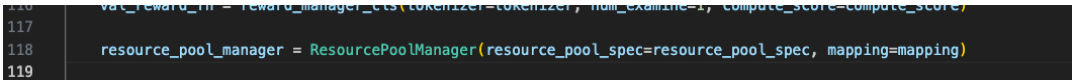
6. 注意事项

- **类的可序列化**：
  - 类 `A` 及其方法必须是可序列化的，因为 Ray 需要将它们传输到远程节点。
- **性能开销**：
  - 远程调用会有一定的性能开销，适合计算密集型任务，而不适合频繁的小任务。
- **资源管理**：
  - 需要合理管理 Ray 集群的资源，避免资源竞争或浪费。

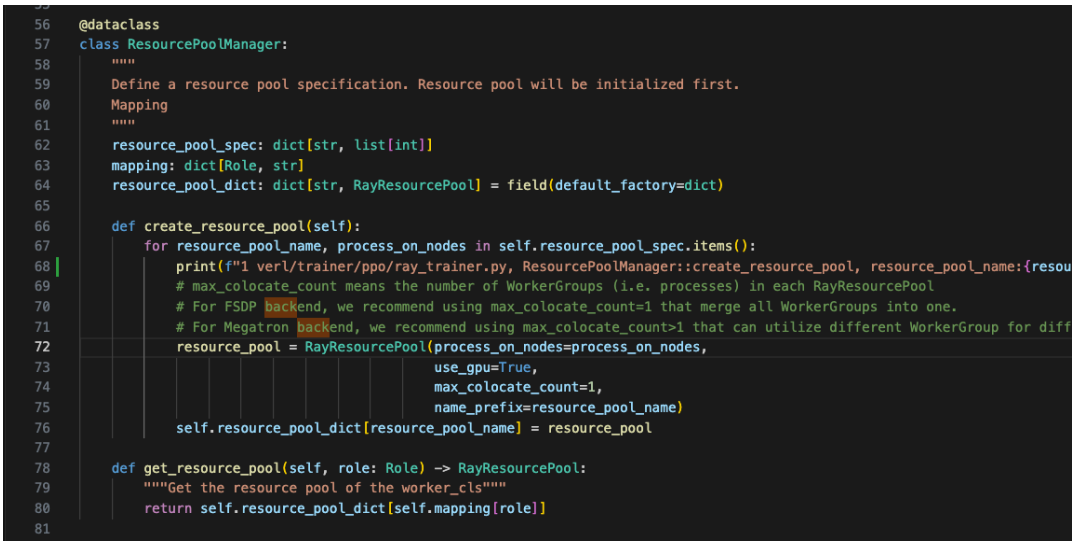
总结

`ray.remote(A)` 的作用是将类 `A` 标记为远程类，使得 `A` 的实例可以在 Ray 的分布式环境中运行。通过 `ray.remote(A)`，你可以轻松地将类的实例和方法分布到多个节点上执行，从而实现分布式计算和并行化。

2) verl/trainer/main_ppo.py

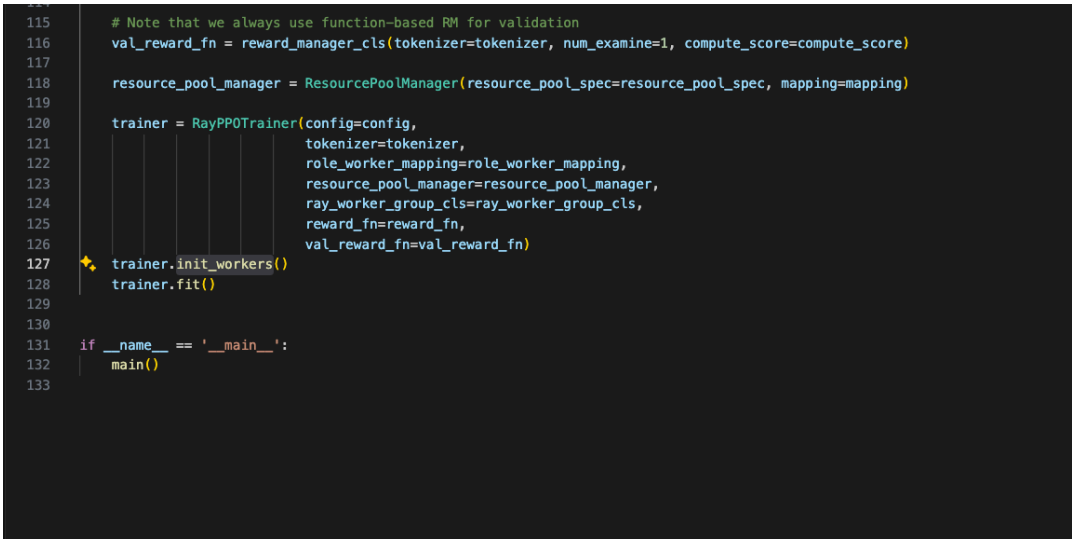


I class ResourcePoolManager(verl/trainer/ppo/ray_trainer.py)



```python
@dataclass
class ResourcePoolManager:
    """
    Define a resource pool specification. Resource pool will be initialized first.
    Mapping
    """
    resource_pool_spec: dict[str, list[int]]
    mapping: dict[Role, str]
    resource_pool_dict: dict[str, RayResourcePool] = field(default_factory=dict)

    def create_resource_pool(self):
        for resource_pool_name, process_on_nodes in self.resource_pool_spec.items():
            print(f"1 verl/trainer/ppo/ray_trainer.py, ResourcePoolManager::create_resource_pool, resource_pool_name:{resou
            # max_colocate_count means the number of WorkerGroups (i.e. processes) in each RayResourcePool
            # For FSDP backend, we recommend using max_colocate_count=1 that merge all WorkerGroups into one.
            # For Megatron backend, we recommend using max_colocate_count>1 that can utilize different WorkerGroup for diff
            resource_pool = RayResourcePool(process_on_nodes=process_on_nodes,
                                            use_gpu=True,
                                            max_colocate_count=1,
                                            name_prefix=resource_pool_name)
            self.resource_pool_dict[resource_pool_name] = resource_pool

    def get_resource_pool(self, role: Role) -> RayResourcePool:
        """Get the resource pool of the worker_cls"""
        return self.resource_pool_dict[self.mapping[role]]

```

3)RayPROTrainer init(verl/trainer/main_ppo.py)



```python
    # Note that we always use function-based RM for validation
    val_reward_fn = reward_manager_cls(tokenizer=tokenizer, num_examine=1, compute_score=compute_score)

    resource_pool_manager = ResourcePoolManager(resource_pool_spec=resource_pool_spec, mapping=mapping)

    trainer = RayPPOTrainer(config=config,
                            tokenizer=tokenizer,
                            role_worker_mapping=role_worker_mapping,
                            resource_pool_manager=resource_pool_manager,
                            ray_worker_group_cls=ray_worker_group_cls,
                            reward_fn=reward_fn,
                            val_reward_fn=val_reward_fn)
    trainer.init_workers()
    trainer.fit()


if __name__ == '__main__':
    main()

```

I class RayPPOTrainer(verl/trainer/ppo/ray_trainer.py)

问题1:self.kl_ctrl干什么的

```python
        self.role_worker_mapping = role_worker_mapping
        self.resource_pool_manager = resource_pool_manager
        self.use_reference_policy = Role.RefPolicy in role_worker_mapping
        self.use_rm = Role.RewardModel in role_worker_mapping
        self.ray_worker_group_cls = ray_worker_group_cls
        print(f"1 verl/trainer/ppo/ray_trainer.py, RayPPOTrainer:: __init__, self.use_reference_policy:{self.use_reference_
        # define KL control
        if self.use_reference_policy:
            if config.algorithm.kl_ctrl.type == 'fixed':
                self.kl_ctrl = core_algos.FixedKLController(kl_coef=config.algorithm.kl_ctrl.kl_coef)
            elif config.algorithm.kl_ctrl.type == 'adaptive':
                assert config.algorithm.kl_ctrl.horizon > 0, f'horizon must be larger than 0. Got {config.critic.kl_ctrl.ho
                self.kl_ctrl = core_algos.AdaptiveKLController(init_kl_coef=config.algorithm.kl_ctrl.kl_coef,
                                                               target_kl=config.algorithm.kl_ctrl.target_kl,
                                                               horizon=config.algorithm.kl_ctrl.horizon)
            else:
                raise NotImplementedError
        else:
            self.kl_ctrl = core_algos.FixedKLController(kl_coef=0.)
        print(f"2 verl/trainer/ppo/ray_trainer.py, RayPPOTrainer:: __init__, self.config.algorithm.adv_estimator: {self.con
        if self.config.algorithm.adv_estimator == 'gae':
            self.use_critic = True
        elif self.config.algorithm.adv_estimator == 'grpo':
            self.use_critic = False
        elif self.config.algorithm.adv_estimator == 'reinforce_plus_plus':
            self.use_critic = False
```

II: sef._validate_config

III:self._create_dataloader

iv:

4) verl/trainer/main_ppo.py trainer.init_workers() → verl/trainer/ppo/ray_trainer.py

i: 初始化Actor, Critic, Ref, RM worker

```python
    def init_workers(self):
        """Init resource pool and worker group"""
        self.resource_pool_manager.create_resource_pool()

        self.resource_pool_to_cls = {pool: {} for pool in self.resource_pool_manager.resource_pool_dict.values()}

        # create actor and rollout
        if self.hybrid_engine:
            resource_pool = self.resource_pool_manager.get_resource_pool(Role.ActorRollout)
            actor_rollout_cls = RayClassWithInitArgs(cls=self.role_worker_mapping[Role.ActorRollout],
                                                     config=self.config.actor_rollout_ref,
                                                     role='actor_rollout')
            self.resource_pool_to_cls[resource_pool]['actor_rollout'] = actor_rollout_cls
        else:
            raise NotImplementedError

        # create critic
        if self.use_critic:
            resource_pool = self.resource_pool_manager.get_resource_pool(Role.Critic)
            critic_cls = RayClassWithInitArgs(cls=self.role_worker_mapping[Role.Critic], config=self.config.critic)
            self.resource_pool_to_cls[resource_pool]['critic'] = critic_cls

        # create reference policy if needed
        if self.use_reference_policy:
            resource_pool = self.resource_pool_manager.get_resource_pool(Role.RefPolicy)
            ref_policy_cls = RayClassWithInitArgs(self.role_worker_mapping[Role.RefPolicy],
                                                  config=self.config.actor_rollout_ref,
                                                  role='ref')
            self.resource_pool_to_cls[resource_pool]['ref'] = ref_policy_cls

        # create a reward model if reward_fn is None
        if self.use_rm:
            # we create a RM here
            resource_pool = self.resource_pool_manager.get_resource_pool(Role.RewardModel)
            rm_cls = RayClassWithInitArgs(self.role_worker_mapping[Role.RewardModel], config=self.config.reward_model)
            self.resource_pool_to_cls[resource_pool]['rm'] = rm_cls
```

```
666     # initialize WorkerGroup
667     # NOTE: if you want to use a different resource pool for each role, which can support different parallel size,
668     # you should not use `create_colocated_worker_cls`. Instead, directly pass different resource pool to different worker groups.
669     # See https://github.com/volcengine/verl/blob/master/examples/ray/tutorial.ipynb for more information.
670     all_wg = {}
671     self.wg_dicts = []
672     for resource_pool, class_dict in self.resource_pool_to_cls.items():
673         worker_dict_cls = create_colocated_worker_cls(class_dict=class_dict)
674         wg_dict = self.ray_worker_group_cls(resource_pool=resource_pool, ray_cls_with_init=worker_dict_cls)
675         spawn_wg = wg_dict.spawn(prefix_set=class_dict.keys())
676         all_wg.update(spawn_wg)
677         # keep the referece of WorkerDict to support ray >= 2.31. Ref: https://github.com/ray-project/ray/pull/45699
678         self.wg_dicts.append(wg_dict)
679
680     if self.use_critic:
681         self.critic_wg = all_wg['critic']
682         self.critic_wg.init_model()
683
684     if self.use_reference_policy:
685         self.ref_policy_wg = all_wg['ref']
686         self.ref_policy_wg.init_model()
687
688     if self.use_rm:
689         self.rm_wg = all_wg['rm']
690         self.rm_wg.init_model()
691
692     # we should create rollout at the end so that vllm can have a better estimation of kv cache memory
693     self.actor_rollout_wg = all_wg['actor_rollout']
694     print(f"1 verl/trainer/ppo/ray_trainer.py, RayPPOTrainer::init_workers, self.actor_rollout_wg:{self.actor_rollout_wg} and type(self.
695     self.actor_rollout_wg.init_model()
```

line 672的self.respource_pool_to_cls是RayWorkerGroup(from verl.single_controller.ray import RayWorkerGroup， config.actor_rollout_ref.actor.strategy == 'fsdp')

问题2 :line 673～line 675 worker_dict_cls = create_colocated_worker_cls(class_dict=class_dict),⇒以后来看(verl/single_controller/ray/base.py)

```
176 class RayWorkerGroup(WorkerGroup):
177
178     def __init__(self,
179                  resource_pool: RayResourcePool = None,
180                  ray_cls_with_init: RayClassWithInitArgs = None,
181                  bin_pack: bool = True,
182                  name_prefix: str = None,
183                  detached=False,
184                  worker_names=None,
185                  **kwargs) -> None:
186         super().__init__(resource_pool=resource_pool, **kwargs)
187         self.ray_cls_with_init = ray_cls_with_init
188         self.name_prefix = get_random_string(length=6) if name_prefix is None else name_prefix
189
190         if worker_names is not None:
191             assert self._is_init_with_detached_workers
192             self._worker_names = worker_names
193
194         if self._is_init_with_detached_workers:
195             self._init_with_detached_workers(worker_names=worker_names)
196         else:
197             self._init_with_resource_pool(resource_pool=resource_pool,
198                                           ray_cls_with_init=ray_cls_with_init,
199                                           bin_pack=bin_pack,
200                                           detached=detached)
201
202         if ray_cls_with_init is not None:
203             self._bind_worker_method(self.ray_cls_with_init.cls, func_generator)
```

==========

ii: ActorRolloutRefWorker, CriticWorker，RewardModelWorker的init_model

a)先了解fsdp

PyTorch Fully Sharded Data Parallel (FSDP) 是 PyTorch 提供的一种分布式训练技术，旨在解决大规模模型训练中的显存瓶颈问题。FSDP 通过 **分片模型参数、梯度和优化器状态**，将它们分布到多个 GPU 上，从而显著减少单个 GPU 的显存占用。以下是对 FSDP 的详细解释：

1. FSDP 的核心思想

FSDP 的核心思想是 **完全分片（Fully Sharded）**，即：

- **模型参数分片**：将模型的每一层参数分片到多个 GPU 上，而不是在每个 GPU 上保存完整的模型副本。
- **梯度分片**：在反向传播时，每个 GPU 只计算自己分片部分的梯度。
- **优化器状态分片**：优化器的状态（如动量、Adam 的二阶矩估计等）也分片到多个 GPU 上。

通过这种方式，FSDP 可以显著减少单个 GPU 的显存占用，从而支持更大规模的模型训练。

2. FSDP 的工作原理

(1) 前向传播

- 每个 GPU 只加载自己分片部分的模型参数。
- 在前向传播过程中，FSDP 会自动从其他 GPU 获取所需的分片参数（通过通信）。
- 计算完成后，释放不需要的参数以节省显存。

(2) 反向传播

- 每个 GPU 只计算自己分片部分的梯度。
- 在反向传播过程中，FSDP 会通过通信获取其他 GPU 的分片梯度。
- 计算完成后，释放不需要的梯度以节省显存。

(3) 优化器更新

- 每个 GPU 只更新自己分片部分的参数和优化器状态。
- 在更新完成后，FSDP 会通过通信同步所有 GPU 的参数。

3. FSDP 的优势

- **显存效率高**：
  - 通过分片参数、梯度和优化器状态，显存占用显著减少。
  - 可以训练比传统数据并行（如 `torch.nn.DataParallel` 或 `torch.nn.parallel.DistributedDataParallel`）更大的模型。
- **扩展性强**：
  - 支持大规模分布式训练，适用于数百甚至数千个 GPU。
- **灵活性高**：
  - 支持自定义分片策略，可以根据模型结构优化分片方式。

4. FSDP 的使用方法

以下是一个简单的 FSDP 使用示例：

```
import torch
import torch.distributed as dist
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP
```

```
from torch.distributed.fsdp import ShardingStrategy
from torch.nn import Linear

# 初始化分布式环境
dist.init_process_group(backend="nccl")

# 定义一个简单的模型
class MyModel(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = Linear(1024, 1024)
        self.layer2 = Linear(1024, 1024)

    def forward(self, x):
        return self.layer2(self.layer1(x))

# 创建模型
model = MyModel().cuda()

# 使用 FSDP 包装模型
sharding_strategy = ShardingStrategy.FULL_SHARD  # 完全分片策略
model = FSDP(model, sharding_strategy=sharding_strategy)

# 定义优化器
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# 训练步骤
inputs = torch.randn(32, 1024).cuda()
outputs = model(inputs)
loss = outputs.sum()
loss.backward()
optimizer.step()
```

5. FSDP 的分片策略

FSDP 支持多种分片策略，可以通过 `ShardingStrategy` 指定：

- **FULL_SHARD**：
  - 完全分片，参数、梯度和优化器状态都分片到多个 GPU 上。
  - 显存占用最小，但通信开销最大。
- **SHARD_GRAD_OP**：
  - 只分片梯度和优化器状态，参数在每个 GPU 上保存完整副本。
  - 显存占用较高，但通信开销较小。
- **NO_SHARD**：
  - 不分片，相当于传统的 `DistributedDataParallel`。

- 显存占用最高，但通信开销最小。

---

6. FSDP 的通信优化

FSDP 通过以下方式优化通信：

- **通信重叠**：
  - 在前向传播和反向传播过程中，FSDP 会异步通信，减少通信开销。
- **分片通信**：
  - 只通信需要的分片数据，而不是整个参数或梯度。

---

7. FSDP 的适用场景

- **大规模模型训练**：
  - 当模型参数过多，单个 GPU 无法容纳时，FSDP 是一个理想的选择。
- **显存受限的环境**：
  - 在显存有限的 GPU 集群上，FSDP 可以显著提高训练效率。
- **分布式训练**：
  - 支持数百甚至数千个 GPU 的分布式训练。

---

8. FSDP 的局限性

- **通信开销**：
  - 分片策略会增加通信开销，尤其是在 `FULL_SHARD` 模式下。
- **实现复杂度**：
  - 需要手动配置分片策略和分布式环境，对用户的要求较高。
- **兼容性**：
  - 某些自定义模型或操作可能不支持 FSDP。

9. FSDP 与传统数据并行的对比

特性	FSDP	传统数据并行（DDP）
显存占用	低（分片参数、梯度和优化器状态）	高（每个 GPU 保存完整模型副本）
通信开销	较高（分片通信）	较低（全量通信）
适用模型规模	大规模模型	中小规模模型
实现复杂度	较高	较低

总结

PyTorch FSDP 是一种高效的分布式训练技术，通过分片模型参数、梯度和优化器状态，显著减少显存占用，支持大规模模型训练。它的核心优势在于显存效率高、扩展性强，但需要权衡通信开销和实现复杂度。如果你需要训练超大规模模型，FSDP 是一个非常强大的工具。

ii)ActorRolloutRefWorker::init

问题3:self._is_offload_param, self._is_offload_grad, self._is_offload_optimizer还有self.config.actor.ppo_mini_batch_size，self.config.actor.ppo_mini_batch_size，self.config.actor.ppo_micro_batch_size_per_gpu， self.config.rollout.log_prob_micro_batch_size_per_gpu，self.config.ref.log_prob_micro_batch_size 这些是干什么用的

```
            self._is_offload_param = self.config.actor.fsdp_config.get('param_offload', False)
            self._is_offload_grad = self.config.actor.fsdp_config.get('grad_offload', False)
            self._is_offload_optimizer = self.config.actor.fsdp_config.get('optimizer_offload', False)
        elif self._is_ref:
            # TODO: it seems that manual offload is slowly than FSDP offload
            self._is_offload_param = self.config.ref.fsdp_config.get('param_offload', False)
        print(f"4 verl/workers/fsdp_workers.py ActorRolloutRefWorker.__init__ self._is_offload_param: {self._is_offload_param} and self._is_offload_
        # normalize config
        if self._is_actor:
            self.config.actor.ppo_mini_batch_size *= self.config.rollout.n #TODO(Xiao):why multiply by n?
            print(f"5 verl/workers/fsdp_workers.py ActorRolloutRefWorker.__init__ self.config.actor.ppo_mini_batch_size: {self.config.actor.ppo_mini
            print(f"6 verl/workers/fsdp_workers.py ActorRolloutRefWorker.__init__ self.device_mesh.shape[0]:{self.device_mesh.shape[0]} ")
            print(f"7 verl/workers/fsdp_workers.py ActorRolloutRefWorker.__init__ self.ulysses_sequence_parallel_size: {self.ulysses_sequence_parall
            self.config.actor.ppo_mini_batch_size //= (self.device_mesh.shape[0] // self.ulysses_sequence_parallel_size)#TODO(Xiao):self.device_mesh
            # micro bsz
            if self.config.actor.ppo_micro_batch_size is not None:
                self.config.actor.ppo_micro_batch_size //= (self.device_mesh.shape[0] //
                                                            self.ulysses_sequence_parallel_size)
                self.config.actor.ppo_micro_batch_size_per_gpu = self.config.actor.ppo_micro_batch_size
                assert self.config.actor.ppo_mini_batch_size % self.config.actor.ppo_micro_batch_size_per_gpu == 0
                print(f"8 verl/workers/fsdp_workers.py ActorRolloutRefWorker.__init__ self.config.actor.ppo_micro_batch_size: {self.config.actor.ppo
                print(f"9 verl/workers/fsdp_workers.py ActorRolloutRefWorker.__init__ self.config.actor.ppo_micro_batch_size_per_gpu: {self.config.a
        # normalize rollout config
        if self._is_rollout and self.config.rollout.log_prob_micro_batch_size is not None:
            self.config.rollout.log_prob_micro_batch_size //= (self.device_mesh.shape[0] //
                                                               self.ulysses_sequence_parallel_size)
            self.config.rollout.log_prob_micro_batch_size_per_gpu = self.config.rollout.log_prob_micro_batch_size
        # normalize ref config
        if self._is_ref and self.config.ref.log_prob_micro_batch_size is not None:
            self.config.ref.log_prob_micro_batch_size //= (self.device_mesh.shape[0] //
                                                           self.ulysses_sequence_parallel_size)
            self.config.ref.log_prob_micro_batch_size_per_gpu = self.config.ref.log_prob_micro_batch_size

```

iii)问题5:ActorRolloutRefWorker::_build_model_optimizer不是很懂(verl/workers/fsdp_workers.py)

iv)问题4:在ActorRolloutRefWorker::init_model中，self.actor_module_fsdp, self.actor,self.ref_module_fsdp 干什么用

```
def init_model(self):
        self.actor_module_fsdp, self.actor_optimizer, self.actor_lr_scheduler, self.actor_model_config = self._build_model_optimizer(
            model_path=self.config.model.path,
            fsdp_config=fsdp_config,
            optim_config=optim_config,
            override_model_config=override_model_config,
            use_remove_padding=use_remove_padding,
            enable_gradient_checkpointing=self.config.model.get('enable_gradient_checkpointing', False),
            trust_remote_code=self.config.model.get('trust_remote_code', False),
            use_liger=self.config.model.get('use_liger', False),
            role='actor')

        # get the original unwrapped module
        self.actor_module = self.actor_module_fsdp._fsdp_wrapped_module

        if self._is_offload_param:
            # param is require during state_dict in sharding manager
            offload_fsdp_grad(module=self.actor_module_fsdp)
            log_gpu_memory_usage('After offload actor grad during init', logger=logger)
        if self._is_offload_optimizer:
            offload_fsdp_optimizer(optimizer=self.actor_optimizer)
            log_gpu_memory_usage('After offload actor optimizer during init', logger=logger)
    # load from checkpoint
    if self._is_actor:
        OmegaConf.set_struct(self.config.actor, True)
        with open_dict(self.config.actor):
            self.config.actor.use_remove_padding = use_remove_padding
        self.actor = DataParallelPPOActor(config=self.config.actor,
                                          actor_module=self.actor_module_fsdp,
                                          actor_optimizer=self.actor_optimizer)

    if self._is_rollout:
        self.rollout, self.rollout_sharding_manager = self._build_rollout()

    if self._is_ref:
        self.ref_module_fsdp = self._build_model_optimizer(model_path=self.config.model.path,
                                                           fsdp_config=self.config.ref.fsdp_config,
                                                           optim_config=None,
                                                           override_model_config=override_model_config,
                                                           use_remove_padding=use_remove_padding,
                                                           trust_remote_code=self.config.model.get(
                                                               'trust_remote_code', False),
```

==========

4 CriticWorke::init和 CriticWorke::init_model

```
@register(dispatch_mode=Dispatch.ONE_TO_ALL)
def init_model(self):
    # This is used to import external_lib into the huggingface systems
    import_external_libs(self.config.model.get('external_lib', None))

    from verl.workers.critic import DataParallelPPOCritic
    self.critic_module, self.critic_optimizer, self.critic_lr_scheduler = self._build_critic_model_optimizer(
        self.config)

    if self._is_offload_param:
        offload_fsdp_param_and_grad(module=self.critic_module, offload_grad=self._is_offload_grad)
    if self._is_offload_optimizer:
        offload_fsdp_optimizer(optimizer=self.critic_optimizer)

    self.critic = DataParallelPPOCritic(config=self.config,
                                        critic_module=self.critic_module,
                                        critic_optimizer=self.critic_optimizer)

    self.flops_counter = FlopsCounter(self.critic_model_config)
    self.checkpoint_manager = FSDPCheckpointManager(model=self.critic_module,
                                                    optimizer=self.critic_optimizer,
                                                    lr_scheduler=self.critic_lr_scheduler,
                                                    tokenizer=self.tokenizer)

    torch.cuda.empty_cache()
```

问题6：self.critic， self.critic_module,干什么的

==========

5 RayRPOTrainer::fit(verl/trainer/ppo/ray_trainer.py)

1)line 828 batch: DataProto = DataProto.from_single_dict(batch_dict)#TODO(xiao):2025-02-12, need understand the

DataProto

line 831　　gen_batch = batch.pop(batch_keys=['input_ids', 'attention_mask', 'position_ids'])#Xiao:2025-02-12, why pop these keys

```
817
818         # we start from step 1
819         self.global_steps += 1
820         temp_i = 0
821         for epoch in range(self.config.trainer.total_epochs):
822             for batch_dict in self.train_dataloader:
823                 metrics = {}
824                 timing_raw = {}
825                 if temp_i <= 2:
826                     print(f"1 verl/trainer/ppo/ray_trainer.py, RayPPOTrainer::fit, batch_dict.keys(): {batch_dict.keys()}")
827                     temp_i += 1
828                 batch: DataProto = DataProto.from_single_dict(batch_dict)#TODO(xiao):2025-02-12, need understand the DataProto
829
830                 # pop those keys for generation
831                 gen_batch = batch.pop(batch_keys=['input_ids', 'attention_mask', 'position_ids'])#Xiao:2025-02-12, why pop these keys
832
833                 with _timer('step', timing_raw):
834                     # generate a batch
835                     with _timer('gen', timing_raw):
836                         gen_batch_output = self.actor_rollout_wg.generate_sequences(gen_batch)
837
838                     batch.non_tensor_batch['uid'] = np.array([str(uuid.uuid4()) for _ in range(len(batch.batch))],
839                                                              dtype=object)
840                     # repeat to align with repeated responses in rollout
841                     batch = batch.repeat(repeat_times=self.config.actor_rollout_ref.rollout.n, interleave=True)
842                     batch = batch.union(gen_batch_output)
843
844                     # balance the number of valid tokens on each dp rank.
845                     # Note that this breaks the order of data inside the batch.
846                     # Please take care when you implement group based adv computation such as GRPO and rloo
847                     self._balance_batch(batch, metrics=metrics)
848
849                     # compute global_valid_tokens
```

i)from_dict 初始化Dataproto(verl/protocol.py)

```
291     @classmethod
292     def from_dict(cls, tensors: Dict[str, torch.Tensor], non_tensors=None, meta_info=None, num_batch_dims=1):
293         """Create a DataProto from a dict of tensors. This assumes that
294         1. All the tensor in tensors have the same dim0
295         2. Only dim0 is the batch dim
296         """
297         assert len(tensors) > 0, 'tensors must not be empty'
298         assert num_batch_dims > 0, 'num_batch_dims must be greater than zero'
299         if non_tensors is not None:
300             assert num_batch_dims == 1, 'only support num_batch_dims=1 when non_tensors is not None.'
301
302         if meta_info is None:
303             meta_info = {}
304         if non_tensors is None:
305             non_tensors = {}
306
307         assert isinstance(non_tensors, dict)
308
309         # get and check batch size
310         batch_size = None
311         pivot_key = None
312         for key, tensor in tensors.items():
313             if batch_size is None:
314                 batch_size = tensor.shape[:num_batch_dims]
315                 pivot_key = key
316             else:
317                 current_batch = tensor.shape[:num_batch_dims]
318                 assert batch_size == current_batch, \
319                     f'Not all the tensor in tensors have the same batch size with batch_dims={num_batch_dims}. Got {pivot_key} has
320
321         for key, val in non_tensors.items():
322             #Xiao(25/02/12, type(val) is ndarray, not p.ndarray)
323             non_tensors[key] = np.array(val, dtype=object)#Xiao: 这句代码的作用是将 val（一个 numpy.ndarray 类型的数组）转换为一个新的 NumPy
324
325         tensor_dict = TensorDict(source=tensors, batch_size=batch_size)#Xiao:TensorDict 是 PyTorch 中的一个容器类，用来存储多个张量 (torc
326         return cls(batch=tensor_dict, non_tensor_batch=non_tensors, meta_info=meta_info)
327
```

ii)DataProto::pop (verl/protocol.py)

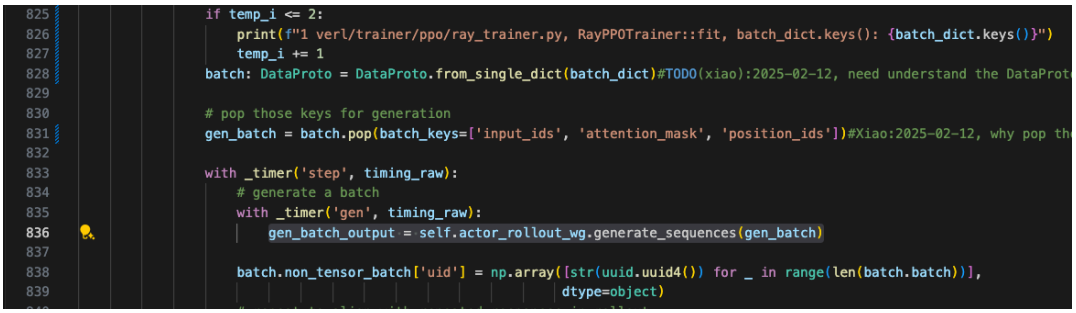
gen_batch = batch.pop(batch_keys=['input_ids', 'attention_mask', 'position_ids'])#Xiao:2025-02-12, why pop these keys

```
376
377     def pop(self, batch_keys=None, non_tensor_batch_keys=None, meta_info_keys=None) -> 'DataProto':
378         """Pop a subset of the DataProto via `batch_keys` and `meta_info_keys`
379
380         Args:
381             batch_keys (list, optional): a list of strings indicating the keys in batch to pop
382             meta_info_keys (list, optional): a list of keys indicating the meta info to pop
383
384         Returns:
385             DataProto: the DataProto with the poped batch_keys and meta_info_keys
386         """
387         assert batch_keys is not None
388         if meta_info_keys is None:
389             meta_info_keys = []
390         if non_tensor_batch_keys is None:
391             non_tensor_batch_keys = []
392
393         tensors = {}
394         # tensor batch
395         for key in batch_keys:
396             assert key in self.batch.keys()
397
398             tensors[key] = self.batch.pop(key)#25/02/12 xiao:pop它的作用是移除并返回 TensorDict 中指定键
399             value = tensors[key]
400             print(f"2 verl/protocol.py, pop, key: {key}, value.shape: {value.shape}")
401             #TODO:attention mask如何初始化？ 25/02/12 xiao: dataloader里面就有?
402         non_tensors = {}
403         # non tensor batch
404         for key in non_tensor_batch_keys:
405             assert key in self.non_tensor_batch.keys()
406             non_tensors[key] = self.non_tensor_batch.pop(key)
407         meta_info = {}
408         for key in meta_info_keys:
409             assert key in self.meta_info.keys()
410             meta_info[key] = self.meta_info.pop(key)
411         return DataProto.from_dict(tensors=tensors, non_tensors=non_tensors, meta_info=meta_info)
412
```

将这个batch的input_ids, attention_mask, position_ids拿出来，变成一个新的DataProto(line 408

问题8:#TODO:attention mask如何初始化？ 25/02/12 xiao: dataloader里面就有？

iii)回到verl/trainer/ppo/ray_trainer.py



```
825                 if temp_i <= 2:
826                     print(f"1 verl/trainer/ppo/ray_trainer.py, RayPPOTrainer::fit, batch_dict.keys(): {batch_dict.keys()}")
827                     temp_i += 1
828                 batch: DataProto = DataProto.from_single_dict(batch_dict)#TODO(xiao):2025-02-12, need understand the DataProto
829
830                 # pop those keys for generation
831                 gen_batch = batch.pop(batch_keys=['input_ids', 'attention_mask', 'position_ids'])#Xiao:2025-02-12, why pop th
832
833                 with _timer('step', timing_raw):
834                     # generate a batch
835                     with _timer('gen', timing_raw):
836                         gen_batch_output = self.actor_rollout_wg.generate_sequences(gen_batch)
837
838                     batch.non_tensor_batch['uid'] = np.array([str(uuid.uuid4()) for _ in range(len(batch.batch))],
839                                                              dtype=object)
```

2)gen_batch_output = self.actor_rollout_wg.generate_sequences(gen_batch)

==========

回到verl/workers/fsdp_workers.py(ActorRolloutRefWorker)

a. self.rollout的init,

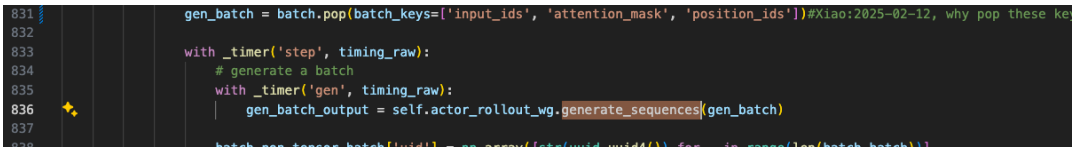
```
295  def _build_rollout(self):
296      from torch.distributed.device_mesh import init_device_mesh
297      # TODO(sgm): support FSDP hybrid shard for larger model
298      infer_tp = self.config.rollout.tensor_model_parallel_size
299      dp = self.world_size // infer_tp
300      assert self.world_size % infer_tp == 0, f'rollout world_size: {self.world_size} is not divisible by infer_tp: {infer_tp}'
301      rollout_device_mesh = init_device_mesh('cuda', mesh_shape=(dp, infer_tp), mesh_dim_names=['dp', 'infer_tp'])
302
303      if self.config.rollout.name == 'hf':
304          from verl.workers.rollout import HFRollout
305          from verl.workers.sharding_manager import BaseShardingManager
306          rollout = HFRollout(module=self.actor_module_fsdp, config=self.config.rollout)
307          rollout_sharding_manager = BaseShardingManager()
308          # TODO: a sharding manager that do nothing?
309      elif self.config.rollout.name == 'vllm':
310          from verl.workers.rollout.vllm_rollout import vLLMRollout
311          from verl.workers.sharding_manager import FSDPVLLMShardingManager
312          log_gpu_memory_usage('Before building vllm rollout', logger=None)
313          rollout = vLLMRollout(actor_module=self.actor_module_fsdp,
314                                config=self.config.rollout,
315                                tokenizer=self.tokenizer,
316                                model_hf_config=self.actor_model_config)
317          log_gpu_memory_usage('After building vllm rollout', logger=None)
318          if torch.distributed.get_world_size() == 1:
319              self.config.rollout.load_format = 'dummy_hf'
320          rollout_sharding_manager = FSDPVLLMShardingManager(module=self.actor_module_fsdp,
321                                                             inference_engine=rollout.inference_engine,
322                                                             model_config=self.actor_model_config,
323                                                             full_params='hf' in self.config.rollout.load_format,
324                                                             device_mesh=rollout_device_mesh)
325          log_gpu_memory_usage('After building sharding manager', logger=None)
326
327      return rollout, rollout_sharding_manager
```

问题9: line320的rollout_sharding_manager = FSDPVLLMShardingManager作用是什么

line 313 vLLMRollout就是vllm engine

==========

gen_batch_output = self.actor_rollout_wg.generate_sequences(gen_batch)(verl/trainer/ppo/ray_trainer.py)



```
831          gen_batch = batch.pop(batch_keys=['input_ids', 'attention_mask', 'position_ids'])#Xiao:2025-02-12, why pop these ke
832
833          with _timer('step', timing_raw):
834              # generate a batch
835              with _timer('gen', timing_raw):
836                  gen_batch_output = self.actor_rollout_wg.generate_sequences(gen_batch)
837
```

回到verl/workers/fsdp_workers.py(ActorRolloutRefWorker::generate_sequences)

ActorRolloutRefWorker::generate_sequences(verl/workers/fsdp_workers.py)

a)

问题10: line 460不是很理解，干什么用的

```python
@register(dispatch_mode=Dispatch.DP_COMPUTE_PROTO)
def generate_sequences(self, prompts: DataProto):
    prompts = prompts.to('cuda')

    assert self._is_rollout
    if self._is_offload_param:
        load_fsdp_param_and_grad(module=self.actor_module_fsdp,
                                 device_id=torch.cuda.current_device(),
                                 load_grad=self._is_offload_grad)#TODO(xiao) 25/02/12, do not understand this, why this

    prompts.batch = prompts.batch.cuda()
    meta_info = {
        'eos_token_id':
            self.generation_config.eos_token_id
            if self.generation_config is not None else self.tokenizer.eos_token_id,
        'pad_token_id':
            self.generation_config.pad_token_id
            if self.generation_config is not None else self.tokenizer.pad_token_id,
    }
```

==========

b)

```python
        'pad_token_id':
            self.generation_config.pad_token_id
            if self.generation_config is not None else self.tokenizer.pad_token_id,
    }
    prompts.meta_info.update(meta_info)
    with self.rollout_sharding_manager:
        log_gpu_memory_usage('After entering rollout sharding manager', logger=logger)

        prompts = self.rollout_sharding_manager.preprocess_data(prompts)#TODO(xiao) 25/02/12, do not understand this, why this, very
        output = self.rollout.generate_sequences(prompts=prompts) #TODO(xiao) 25/02/12, do not understand this, why this, very import

        log_gpu_memory_usage('After rollout generation', logger=logger)

        output = self.rollout_sharding_manager.postprocess_data(output)

    output = output.to('cpu')

    if self._is_offload_param:
        # NOTE(sgm): the grad is already in CPU, only offload param here
```

==========

b-1) line477 verl/workers/fsdp_workers.py

```python
            self.generation_config.pad_token_id
            if self.generation_config is not None else self.tokenizer.pad_token_id,
    }
    prompts.meta_info.update(meta_info)
    with self.rollout_sharding_manager:
        log_gpu_memory_usage('After entering rollout sharding manager', logger=logger)

        prompts = self.rollout_sharding_manager.preprocess_data(prompts)#TODO(xiao) 25/02/12, do not understand this, why this, very
        output = self.rollout.generate_sequences(prompts=prompts) #TODO(xiao) 25/02/12, do not understand this, why this, very import

        log_gpu_memory_usage('After rollout generation', logger=logger)

        output = self.rollout_sharding_manager.postprocess_data(output)

    output = output.to('cpu')

    if self._is_offload_param:
        # NOTE(sgm): the grad is already in CPU, only offload param here
        offload_fsdp_param_and_grad(module=self.actor_module_fsdp, offload_grad=self._is_offload_grad)
    # clear kv cache
```

问题11:FSDPUlyssesShardingManager::preprocess_data干什么用的(verl/workers/sharding_manager/fsdp_ulysses.py)，看来需要很了解FSDP

```python
    def preprocess_data(self, data: DataProto) -> DataProto:#TODO(xiao) 25/02/12, do not understand this, why this, very important
        """
        AllGather data from sp region
        This is because the data is first sharded along the FSDP dimension as we utilize the DP_COMPUTE
        In Ulysses, we need to make sure the same data is used across a SP group
        """
        if self.device_mesh is not None:
            sp_size = self.device_mesh['sp'].size()
            group = self.device_mesh['sp'].get_group()

            prev_device = data.batch.device
            data.batch = data.batch.cuda(device=torch.cuda.current_device())
            data.batch = allgather_dict_tensors(data.batch.contiguous(), size=sp_size, group=group, dim=0)
            data.batch = data.batch.to(prev_device)
            # all gather non_tensor_batch
            all_non_tensor_batch = [None for _ in range(sp_size)]
            torch.distributed.all_gather_object(all_non_tensor_batch, data.non_tensor_batch, group=group)
            data.non_tensor_batch = {
                k: np.concatenate([d[k] for d in all_non_tensor_batch]) for k in data.non_tensor_batch
            }
        return data

```

为什么需要sp_size, grouo, 然后all_gather,

b-2) line477 verl/workers/fsdp_workers.py

```python
            'pad_token_id':
                self.generation_config.pad_token_id
                if self.generation_config is not None else self.tokenizer.pad_token_id,
        }
        prompts.meta_info.update(meta_info)
        with self.rollout_sharding_manager:
            log_gpu_memory_usage('After entering rollout sharding manager', logger=logger)

            prompts = self.rollout_sharding_manager.preprocess_data(prompts)#TODO(xiao) 25/02/12, do not understand this, why this, very
            output = self.rollout.generate_sequences(prompts=prompts) #TODO(xiao) 25/02/12, do not understand this, why this, very impor

            log_gpu_memory_usage('After rollout generation', logger=logger)

            output = self.rollout_sharding_manager.postprocess_data(output)

        output = output.to('cpu')

        if self._is_offload_param:
            # NOTE(sgm): the grad is already in CPU, only offload param here
            offload_fsdp_param_and_grad(module=self.actor_module_fsdp, offload_grad=self._is_offload_grad)
        # clear kv cache
        torch.cuda.empty_cache()
```

line 478 output = self.rollout.generate_sequences(prompts=prompts),调用vllm engine, 然后prompts是在cuda的DataProto

==========

关于_pre_process_inputs

```python

# NOTE(sgm): add for verl. We can optimize it by making the dataloader yield List[int] without padding.
def _pre_process_inputs(pad_token_id, prompt_token_ids: torch.Tensor) -> List[int]:
    # remove the left padding in the prompt token_id
    # pad_token_id = self.llm_engine.tokenizer.pad_token_id if self.llm_engine.tokenizer.pad_token_id is not None else self.llm_engine.tokenizer.eos_token
    non_pad_index = torch.nonzero(prompt_token_ids != pad_token_id, as_tuple=False)[0][0]
    token_ids = prompt_token_ids[non_pad_index:].tolist()
    return token_ids
    """
    这个 `_pre_process_inputs` 函数的作用是对输入的 `prompt_token_ids` 进行预处理，具体来说，是**去除左侧的填充（padding）部分**，并返回一个不包含填充的 token ID 列表。以

```

```python
    @torch.no_grad()
    def generate_sequences(self, prompts: DataProto, **kwargs) -> DataProto:
        # rebuild vllm cache engine
        if self.config.free_cache_engine:
            self.inference_engine.init_cache_engine()

        idx = prompts.batch['input_ids']  # (bs, prompt_length)
        # left-padded attention_mask
        attention_mask = prompts.batch['attention_mask']
        position_ids = prompts.batch['position_ids']

        # used to construct attention_mask
        eos_token_id = prompts.meta_info['eos_token_id']

        batch_size = idx.size(0)

        idx_list = []
        # parse idx from torch.Tensor to List[List[str]]
        for i in range(batch_size):
            idx_list.append(_pre_process_inputs(self.pad_token_id, idx[i]))

        do_sample = prompts.meta_info.get('do_sample', True)
        if not do_sample:
            kwargs = {
                'best_of': 1,
                'top_p': 1.0,
                'top_k': -1,
                'min_p': 0.0,
                'temperature': 0,
                'n': 1  # if greedy, only 1 response
            }

        # users can customize different sampling_params at different run
        with self.update_sampling_params(**kwargs):
            output = self.inference_engine.generate(
                prompts=None,  # because we have already convert it to prompt token id
                sampling_params=self.sampling_params,
                prompt_token_ids=idx_list,
                use_tqdm=False)
```

```python
# NOTE(sgm): add for verl. We can optimize it by making the dataloader yield List[int] without padding.
def _pre_process_inputs(pad_token_id, prompt_token_ids: torch.Tensor) -> List[int]:
    # remove the left padding in the prompt token_id
    # pad_token_id = self.llm_engine.tokenizer.pad_token_id if self.llm_engine.tokenizer.pad_token_id is not None else self.llm_engine.tokenizer.eos_token
    non_pad_index = torch.nonzero(prompt_token_ids != pad_token_id, as_tuple=False)[0][0]
    token_ids = prompt_token_ids[non_pad_index:].tolist()
    return token_ids
    """
    这个 `_pre_process_inputs` 函数的作用是对输入的 `prompt_token_ids` 进行预处理，具体来说，是**去除左侧的填充（padding）部分**，并返回一个不包含填充的 token ID 列表。以下

    ---
```

这个 `_pre_process_inputs` 函数的作用是对输入的 `prompt_token_ids` 进行预处理，具体来说，是**去除左侧的填充（padding）部分**，并返回一个不包含填充的 token ID 列表。以下是对代码的详细解释：

函数功能

1. **输入**：
   - `pad_token_id`：填充 token 的 ID（通常是 `tokenizer.pad_token_id`）。
   - `prompt_token_ids`：一个包含 token ID 的 PyTorch 张量（`torch.Tensor`），通常是经过填充的 token ID 序列。
2. **输出**：
   - 返回一个不包含左侧填充的 token ID 列表（`List[int]`）。

代码逐行解释

1. `non_pad_index = torch.nonzero(prompt_token_ids != pad_token_id, as_tuple=False)[0][0]`

- **作用**：
  - 找到第一个非填充 token 的索引。

- **实现细节**：
  - `prompt_token_ids != pad_token_id`：生成一个布尔张量，标记哪些位置的 token 不是填充 token。
  - `torch.nonzero(...)`：返回所有非零元素（即非填充 token）的索引。
  - `[0][0]`：取第一个非填充 token 的索引。

2. `token_ids = prompt_token_ids[non_pad_index:].tolist()`

- **作用**：
  - 从第一个非填充 token 开始，截取 `prompt_token_ids` 的剩余部分，并将其转换为 Python 列表。
- **实现细节**：
  - `prompt_token_ids[non_pad_index:]`：从 `non_pad_index` 开始截取张量，去除左侧的填充部分。
  - `.tolist()`：将 PyTorch 张量转换为 Python 列表。

3. `return token_ids`

- **作用**：
  - 返回不包含左侧填充的 token ID 列表。

示例

假设：

- `pad_token_id = 0`
- `prompt_token_ids = torch.tensor([0, 0, 0, 1, 2, 3, 0, 0])`

执行过程：

1. **找到第一个非填充 token 的索引**：
   - `prompt_token_ids != pad_token_id` 的结果是 `[False, False, False, True, True, True, False, False]`。
   - `torch.nonzero(...)` 的结果是 `[[3], [4], [5]]`。
   - `[0][0]` 的结果是 `3`（第一个非填充 token 的索引）。
2. **截取并转换**：
   - `prompt_token_ids[3:]` 的结果是 `[1, 2, 3, 0, 0]`。
   - `.tolist()` 的结果是 `[1, 2, 3, 0, 0]`。
3. **返回值**：
   - 返回 `[1, 2, 3, 0, 0]`。

优化建议

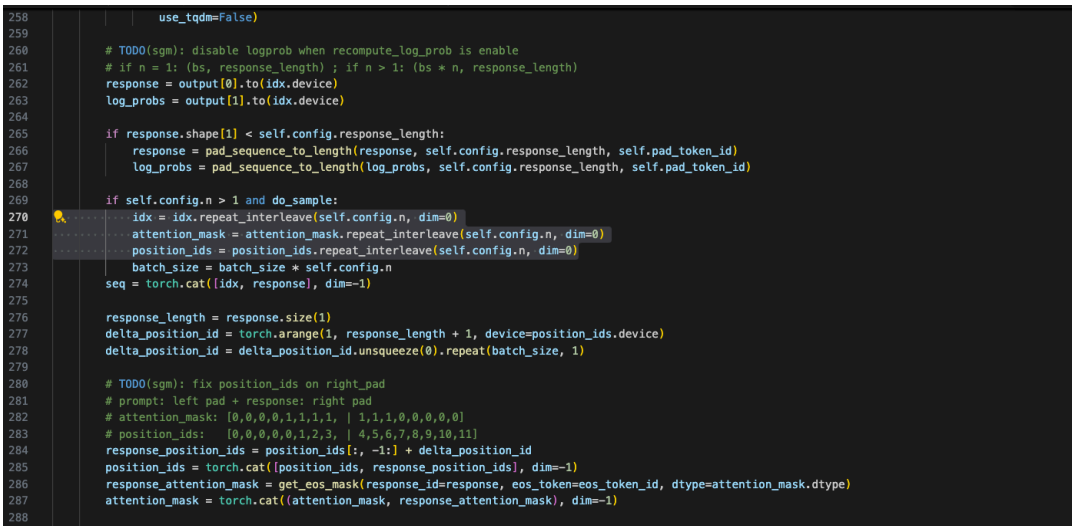
根据代码中的注释（`NOTE(sgm)`），当前的实现是为了处理数据加载器（`dataloader`）返回的填充后的 token ID 序列。如果数据加载器可以直接返回未填充的 token ID 列表（`List[int]`），则可以避免这个预处理步骤，从而提高效率。

总结

`_pre_process_inputs` 函数的作用是去除输入 token ID 序列中的左侧填充部分，并返回一个不包含填充的 token ID 列表。它的主要应用场景是处理经过填充的 token ID 序列，以便后续的模型处理可以忽略填充部分。

==========

问题12:vLLMRollout::generate_seqeuence的一些细节问题 line 262~line 287，非常重要



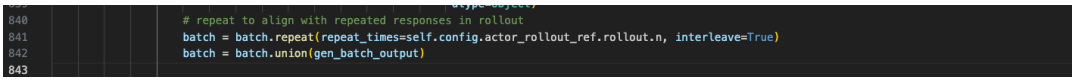
```
# users can customize different sampling_params at different run
with self.update_sampling_params(**kwargs):
    output = self.inference_engine.generate(
        prompts=None,  # because we have already convert it to prompt token id
        sampling_params=self.sampling_params,
        prompt_token_ids=idx_list,
        use_tqdm=False)

# TODO(sgm): disable logprob when recompute_log_prob is enable
# if n = 1: (bs, response_length) ; if n > 1: (bs * n, response_length)
response = output[0].to(idx.device)
log_probs = output[1].to(idx.device)

if response.shape[1] < self.config.response_length:
    response = pad_sequence_to_length(response, self.config.response_length, self.pad_token_id)
    log_probs = pad_sequence_to_length(log_probs, self.config.response_length, self.pad_token_id)

if self.config.n > 1 and do_sample:
    idx = idx.repeat_interleave(self.config.n, dim=0)
    attention_mask = attention_mask.repeat_interleave(self.config.n, dim=0)
    position_ids = position_ids.repeat_interleave(self.config.n, dim=0)
    batch_size = batch_size * self.config.n
seq = torch.cat([idx, response], dim=-1)

response_length = response.size(1)
```

```
        delta_position_id = torch.arange(1, response_length + 1, device=position_ids.device)
        delta_position_id = delta_position_id.unsqueeze(0).repeat(batch_size, 1)

        # TODO(sgm): fix position_ids on right_pad
        # prompt: left pad + response: right pad
        # attention_mask: [0,0,0,0,1,1,1,1, | 1,1,1,0,0,0,0,0]
        # position_ids:   [0,0,0,0,0,1,2,3, | 4,5,6,7,8,9,10,11]
        response_position_ids = position_ids[:, -1:] + delta_position_id
        position_ids = torch.cat([position_ids, response_position_ids], dim=-1)
        response_attention_mask = get_eos_mask(response_id=response, eos_token=eos_token_id, dtype=atte
        attention_mask = torch.cat((attention_mask, response_attention_mask), dim=-1)
```

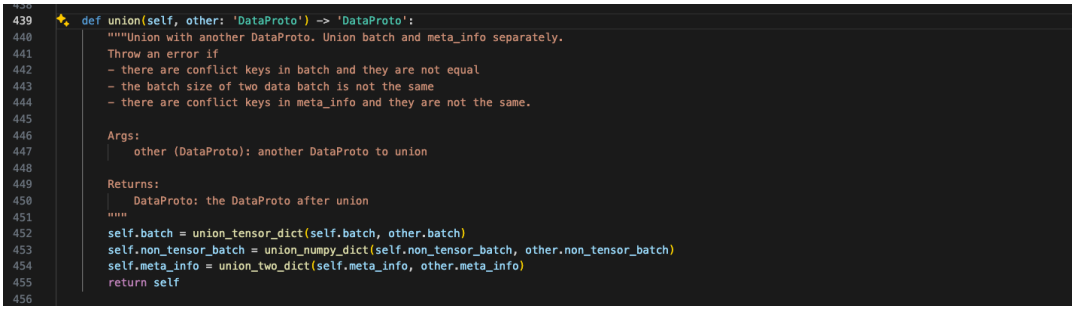
==========

3) verl/trainer/ppo/ray_trainer.py

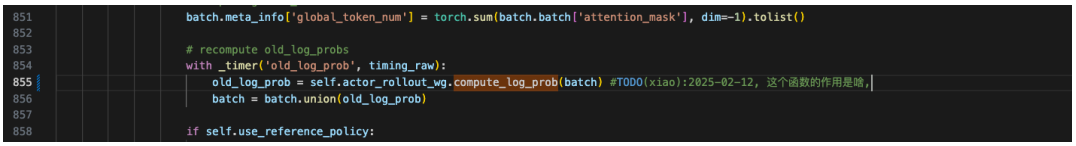
问题13:这里为啥需要batch.repeat和batch.union



问题14(verl/protocol.py)为什么假设there are conflict keys in batch and they are not equal ?

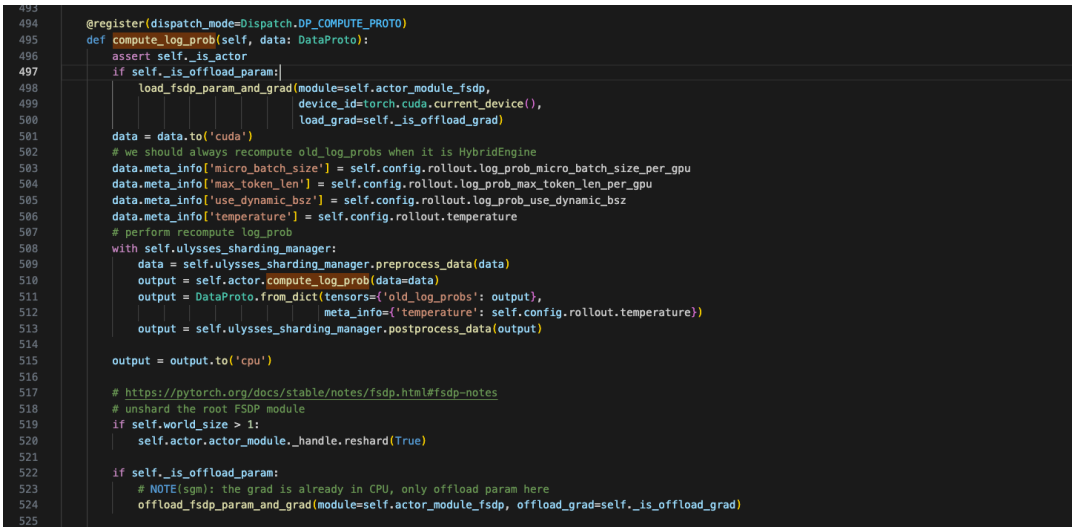


4)verl/trainer/ppo/ray_trainer.py

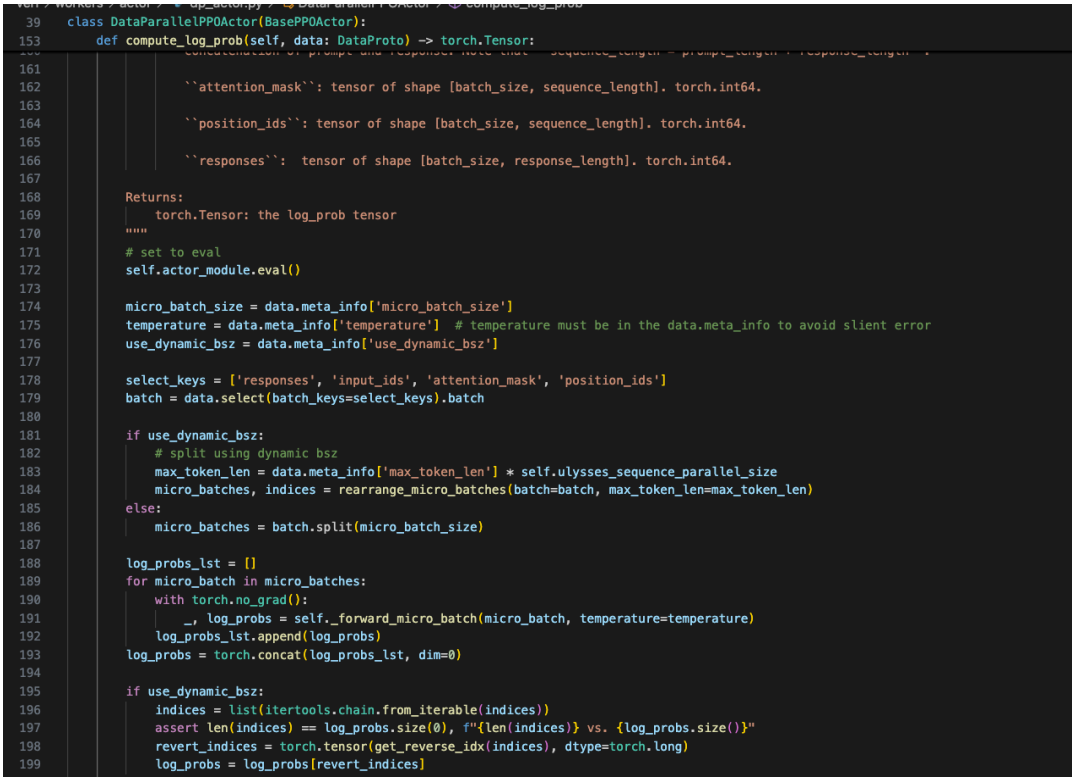


问题15: line 855: old_log_prob = self.actor_rollout_wg.compute_log_prob(batch) #TODO(xiao):2025-02-12, 这个函数的作用是啥

i)line 518 output = self.actor.compute_log_prob(data=data) → DataParallelPPOActor::compute_log_prob(verl/workers/actor/dp_actor.py)



```python
@register(dispatch_mode=Dispatch.DP_COMPUTE_PROTO)
def compute_log_prob(self, data: DataProto):
    assert self._is_actor
    if self._is_offload_param:
        load_fsdp_param_and_grad(module=self.actor_module_fsdp,
                                 device_id=torch.cuda.current_device(),
                                 load_grad=self._is_offload_grad)
    data = data.to('cuda')
    # we should always recompute old_log_probs when it is HybridEngine
    data.meta_info['micro_batch_size'] = self.config.rollout.log_prob_micro_batch_size_per_gpu
    data.meta_info['max_token_len'] = self.config.rollout.log_prob_max_token_len_per_gpu
    data.meta_info['use_dynamic_bsz'] = self.config.rollout.log_prob_use_dynamic_bsz
    data.meta_info['temperature'] = self.config.rollout.temperature
    # perform recompute log_prob
    with self.ulysses_sharding_manager:
        data = self.ulysses_sharding_manager.preprocess_data(data)
        output = self.actor.compute_log_prob(data=data)
        output = DataProto.from_dict(tensors={'old_log_probs': output},
                                     meta_info={'temperature': self.config.rollout.temperature})
        output = self.ulysses_sharding_manager.postprocess_data(output)

    output = output.to('cpu')

    # https://pytorch.org/docs/stable/notes/fsdp.html#fsdp-notes
    # unshard the root FSDP module
    if self.world_size > 1:
        self.actor.actor_module._handle.reshard(True)

    if self._is_offload_param:
        # NOTE(sgm): the grad is already in CPU, only offload param here
        offload_fsdp_param_and_grad(module=self.actor_module_fsdp, offload_grad=self._is_offload_grad)
```



```python
class DataParallelPPOActor(BasePPOActor):
    def compute_log_prob(self, data: DataProto) -> torch.Tensor:
        """
        ``attention_mask``: tensor of shape [batch_size, sequence_length]. torch.int64.

        ``position_ids``: tensor of shape [batch_size, sequence_length]. torch.int64.

        ``responses``:  tensor of shape [batch_size, response_length]. torch.int64.

    Returns:
        torch.Tensor: the log_prob tensor
    """
    # set to eval
    self.actor_module.eval()

    micro_batch_size = data.meta_info['micro_batch_size']
    temperature = data.meta_info['temperature']  # temperature must be in the data.meta_info to avoid slient error
    use_dynamic_bsz = data.meta_info['use_dynamic_bsz']

    select_keys = ['responses', 'input_ids', 'attention_mask', 'position_ids']
    batch = data.select(batch_keys=select_keys).batch

    if use_dynamic_bsz:
        # split using dynamic bsz
        max_token_len = data.meta_info['max_token_len'] * self.ulysses_sequence_parallel_size
        micro_batches, indices = rearrange_micro_batches(batch=batch, max_token_len=max_token_len)
    else:
        micro_batches = batch.split(micro_batch_size)

    log_probs_lst = []
    for micro_batch in micro_batches:
        with torch.no_grad():
            _, log_probs = self._forward_micro_batch(micro_batch, temperature=temperature)
        log_probs_lst.append(log_probs)
    log_probs = torch.concat(log_probs_lst, dim=0)

    if use_dynamic_bsz:
        indices = list(itertools.chain.from_iterable(indices))
        assert len(indices) == log_probs.size(0), f"{len(indices)} vs. {log_probs.size()}"
        revert_indices = torch.tensor(get_reverse_idx(indices), dtype=torch.long)
        log_probs = log_probs[revert_indices]
```

问题15: line 191的self._forward_micro_batch很重要(verl/workers/actor/dp_actor.py)

ii)line 855~ line 876 (verl/trainer/ppo/ray_trainer.py)

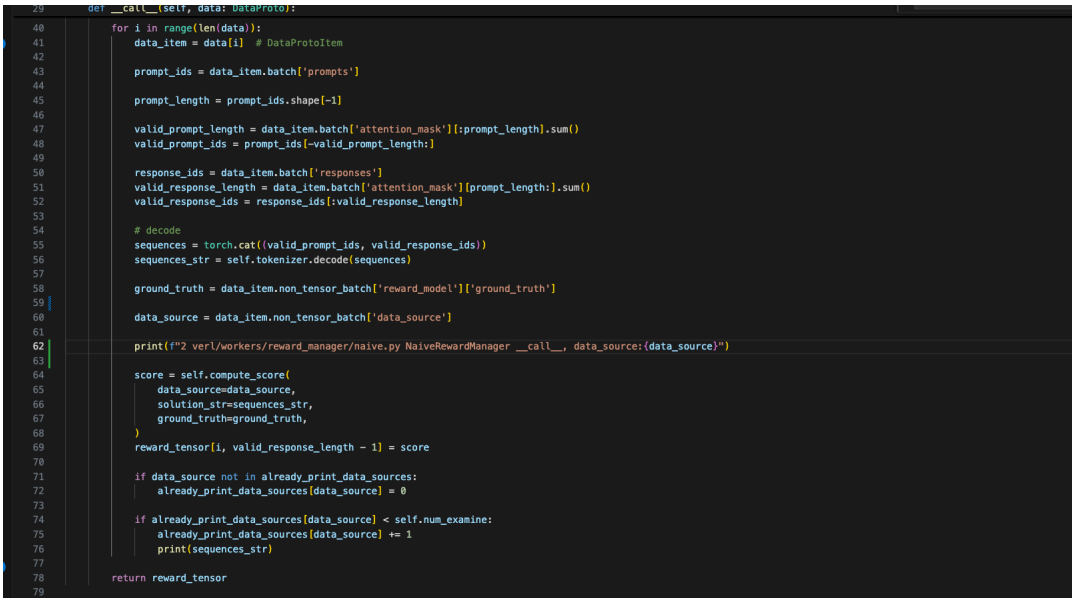
```python
# compute global_valid tokens
batch.meta_info['global_token_num'] = torch.sum(batch.batch['attention_mask'], dim=-1).tolist()

# recompute old_log_probs
with _timer('old_log_prob', timing_raw):
    old_log_prob = self.actor_rollout_wg.compute_log_prob(batch) #TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算log_prob
    batch = batch.union(old_log_prob)

if self.use_reference_policy:
    # compute reference log_prob
    with _timer('ref', timing_raw):
        ref_log_prob = self.ref_policy_wg.compute_ref_log_prob(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算ref_log_prob
        batch = batch.union(ref_log_prob)

# compute values
if self.use_critic:
    with _timer('values', timing_raw):
        values = self.critic_wg.compute_values(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算values
        batch = batch.union(values)

with _timer('adv', timing_raw):
    # compute scores. Support both model and function-based.
    # We first compute the scores using reward model. Then, we call reward_fn to combine
    # the results from reward model and rule-based results.
    if self.use_rm:
        # we first compute reward model score
        reward_tensor = self.rm_wg.compute_rm_score(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算reward_tensor
        batch = batch.union(reward_tensor)
```

问题16: self.actor_rollout_wg.compute_log_prob(batch)，self.ref_policy_wg.compute_ref_log_prob， self.critic_wg.compute_values，self.rm_wg.compute_rm_score为什么是这么计算的

iv)line 880(verl/trainer/ppo/ray_trainer.py), 非常重要

```python
        reward_tensor = self.rm_wg.compute_rm_score(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算reward_tensor
        batch = batch.union(reward_tensor)

    # we combine with rule-based rm
    reward_tensor = self.reward_fn(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算reward_tensor
    batch.batch['token_level_scores'] = reward_tensor

    # compute rewards. apply_kl_penalty if available
    if not self.config.actor_rollout_ref.actor.get('use_kl_loss', False):
        batch, kl_metrics = apply_kl_penalty(batch,
                                             kl_ctrl=self.kl_ctrl,
                                             kl_penalty=self.config.algorithm.kl_penalty)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算kl_metrics
        metrics.update(kl_metrics)
    else:
        batch.batch['token_level_rewards'] = batch.batch['token_level_scores']

    # compute advantages, executed on the driver process
    batch = compute_advantage(batch,
                              adv_estimator=self.config.algorithm.adv_estimator,
                              gamma=self.config.algorithm.gamma,
                              lam=self.config.algorithm.lam,
                              num_repeat=self.config.actor_rollout_ref.rollout.n)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算adv_metrics
```

reward_fn的一个实现（verl/workers/reward_manager/naive.py）



```
    def __call__(self, data: DataProto):

        for i in range(len(data)):
            data_item = data[i]  # DataProtoItem

            prompt_ids = data_item.batch['prompts']

            prompt_length = prompt_ids.shape[-1]

            valid_prompt_length = data_item.batch['attention_mask'][:prompt_length].sum()
            valid_prompt_ids = prompt_ids[-valid_prompt_length:]

            response_ids = data_item.batch['responses']
            valid_response_length = data_item.batch['attention_mask'][prompt_length:].sum()
            valid_response_ids = response_ids[:valid_response_length]

            # decode
            sequences = torch.cat((valid_prompt_ids, valid_response_ids))
            sequences_str = self.tokenizer.decode(sequences)

            ground_truth = data_item.non_tensor_batch['reward_model']['ground_truth']

            data_source = data_item.non_tensor_batch['data_source']

            print(f"2 verl/workers/reward_manager/naive.py NaiveRewardManager __call__, data_source:{data_source}")

            score = self.compute_score(
                data_source=data_source,
                solution_str=sequences_str,
                ground_truth=ground_truth,
            )
            reward_tensor[i, valid_response_length - 1] = score

            if data_source not in already_print_data_sources:
                already_print_data_sources[data_source] = 0

            if already_print_data_sources[data_source] < self.num_examine:
                already_print_data_sources[data_source] += 1
                print(sequences_str)

        return reward_tensor
```

问题17: 还是要仔细读下data是如何初始化的

==========

v)初始化role_worker_mapping(verl/trainer/main_ppo.py)

```
role_worker_mapping = {
    Role.ActorRollout: ray.remote(ActorRolloutRefWorker),
    Role.Critic: ray.remote(CriticWorker),
    Role.RefPolicy: ray.remote(ActorRolloutRefWorker)
}
role_worker_mapping[Role.RewardModel] = ray.remote(RewardModelWorker)
```

很重要，ray的用法

a)verl/trainer/ppo/ray_trainer.py

其中self.ray_worker_group_cls是

from verl.single_controller.ray import RayWorkerGroup

ray_worker_group_cls = RayWorkerGroup

```
    # initialize WorkerGroup
    # NOTE: if you want to use a different resource pool for each role, which can support different parallel size,
    # you should not use `create_colocated_worker_cls`. Instead, directly pass different resource pool to different worker g
    # See https://github.com/volcengine/verl/blob/master/examples/ray/tutorial.ipynb for more information.
    all_wg = {}
    self.wg_dicts = []
    for resource_pool, class_dict in self.resource_pool_to_cls.items():
        worker_dict_cls = create_colocated_worker_cls(class_dict=class_dict)
        wg_dict = self.ray_worker_group_cls(resource_pool=resource_pool, ray_cls_with_init=worker_dict_cls)
        spawn_wg = wg_dict.spawn(prefix_set=class_dict.keys())
        all_wg.update(spawn_wg)
        # keep the referece of WorkerDict to support ray >= 2.31. Ref: https://github.com/ray-project/ray/pull/45699
        self.wg_dicts.append(wg_dict)
```

b)verl/single_controller/ray/base.py很重要

问题18:RayWorkerGroup::init, RayWorkerGroup::_init_with_resource_pool, RayWorkerGroup::spawn(verl/single_controller/ray/base.py) 很重要

2025/02/13

==========

5)critic training(verl/trainer/ppo/ray_trainer.py) line 902

```
899                 # update critic
900                 if self.use_critic:
901                     with _timer('update_critic', timing_raw):
902                         critic_output = self.critic_wg.update_critic(batch)
903                     critic_output_metrics = reduce_metrics(critic_output.meta_info['metrics'])
904                     metrics.update(critic_output_metrics)
905
```

i)def update_critic

```
797     @register(dispatch_mode=Dispatch.DP_COMPUTE_PROTO)
798     def update_critic(self, data: DataProto):
799         data = data.to('cuda')
800         if self._is_offload_param:
801             load_fsdp_param_and_grad(module=self.critic_module,
802                                      device_id=torch.cuda.current_device(),
803                                      load_grad=self._is_offload_grad)
804         if self._is_offload_optimizer:
805             load_fsdp_optimizer(optimizer=self.critic_optimizer, device_id=torch.cuda.current_device())
806
807         # perform forward computation
808         with self.ulysses_sharding_manager:
809             data = self.ulysses_sharding_manager.preprocess_data(data=data)
810
811             with Timer(name='update_critic', logger=None) as timer:
812                 metrics = self.critic.update_critic(data=data)
813             delta_time = timer.last
814
815             global_num_tokens = data.meta_info['global_token_num']
816             estimated_flops, promised_flops = self.flops_counter.estimate_flops(global_num_tokens, delta_time)
817             metrics['mfu/critic'] = estimated_flops * self.config.ppo_epochs / promised_flops / self.world_size
818
819             self.critic_lr_scheduler.step()
820             lr = self.critic_lr_scheduler.get_last_lr()[0]
821             metrics['critic/lr'] = lr
822
823             output = DataProto(batch=None, meta_info={'metrics': metrics})
824             output = self.ulysses_sharding_manager.postprocess_data(data=output)
825
826         if self._is_offload_param:
827             offload_fsdp_param_and_grad(module=self.critic_module, offload_grad=self._is_offload_grad)
828         if self._is_offload_optimizer:
829             offload_fsdp_optimizer(optimizer=self.critic_optimizer)
830         torch.cuda.empty_cache()
831         output = output.to('cpu')
832         return output
```

a)问题20: self.ulysses_sharding_manager的理解(25/02/13)

b)问题21:DataParallelPPOCritic::update_critic(verl/workers/critic/dp_critic.py)很重要,为什么line 157~line 161 要split mini_batch, 同时

```
class DataParallelPPOCritic(BasePPOCritic):
    def update_critic(self, data: DataProto):
        for batch_idx, data in enumerate(dataloader):
            # split batch into micro_batches
            mini_batch = data
            if self.config.use_dynamic_bsz:
                max_token_len = self.config.ppo_max_token_len_per_gpu * self.ulysses_sequence_parallel_size
                micro_batches, _ = rearrange_micro_batches(batch=mini_batch, max_token_len=max_token_len)
            else:
                micro_batches = mini_batch.split(self.config.ppo_micro_batch_size_per_gpu)
                self.gradient_accumulation = self.config.ppo_mini_batch_size // self.config.ppo_micro_batch_size_per_gpu

            self.critic_optimizer.zero_grad()

            for data in micro_batches:
                data = data.cuda()  # critic device is cpu when using offload
                input_ids = data['input_ids']
                responses = data['responses']
                attention_mask = data['attention_mask']
                position_ids = data['position_ids']
                values = data['values']
                returns = data['returns']
                response_length = responses.size(1)

                eos_mask = attention_mask[:, -response_length - 1:-1]

                vpreds = self._forward_micro_batch(data)

                # assert not torch.any(torch.isnan(vpreds)).item()

                vf_loss, vf_clipfrac = core_algos.compute_value_loss(vpreds=vpreds,
                                                                     values=values,
                                                                     returns=returns,
                                                                     eos_mask=eos_mask,
                                                                     cliprange_value=self.config.cliprange_value)
                if self.config.use_dynamic_bsz:
                    # relative to the dynamic bsz
                    loss = vf_loss * (len(data) / self.config.ppo_mini_batch_size)
                else:
                    loss = vf_loss / self.gradient_accumulation

                loss.backward()
```

6)actor training(verl/trainer/ppo/ray_trainer.py) line 910,

```
                # implement critic warmup
                if self.config.trainer.critic_warmup <= self.global_steps:
                    # update actor
                    with _timer('update_actor', timing_raw):
                        actor_output = self.actor_rollout_wg.update_actor(batch)
                    actor_output_metrics = reduce_metrics(actor_output.meta_info['metrics'])
                    metrics.update(actor_output_metrics)
```

==========

a)ActorRolloutRefWorker::update_actor(verl/workers/fsdp_workers.py) → DataParallelPPOActor::update_policy(verl/workers/actor/dp_actor.py)很重要

```python
class ActorRolloutRefWorker(Worker):
    def update_actor(self, data: DataProto):
        if self._is_offload_optimizer:
            load_fsdp_optimizer(optimizer=self.actor_optimizer, device_id=torch.cuda.current_device())

        data.batch = data.batch.cuda()

        log_gpu_memory_usage('Before update policy', logger=logger)

        with self.ulysses_sharding_manager:
            data = self.ulysses_sharding_manager.preprocess_data(data=data)
            # perform training
            with Timer(name='update_policy', logger=None) as timer:
                metrics = self.actor.update_policy(data=data)
            delta_time = timer.last
            global_num_tokens = data.meta_info['global_token_num']
            estimated_flops, promised_flops = self.flops_counter.estimate_flops(global_num_tokens, delta_time)
            metrics['mfu/actor'] = estimated_flops * self.config.actor.ppo_epochs / promised_flops / self.world_size

            self.actor_lr_scheduler.step()
            lr = self.actor_lr_scheduler.get_last_lr()[0]
            metrics['actor/lr'] = lr

            log_gpu_memory_usage('After update policy', logger=logger)

            # TODO: here, we should return all metrics
            output = DataProto(meta_info={'metrics': metrics})

            output = self.ulysses_sharding_manager.postprocess_data(data=output)
            output = output.to('cpu')

        if self._is_offload_param:
            offload_fsdp_param_and_grad(module=self.actor_module_fsdp, offload_grad=self._is_offload_grad)
        if self._is_offload_optimizer:
            offload_fsdp_optimizer(optimizer=self.actor_optimizer)
        torch.cuda.empty_cache()
        return output
```

7)这个data的作用

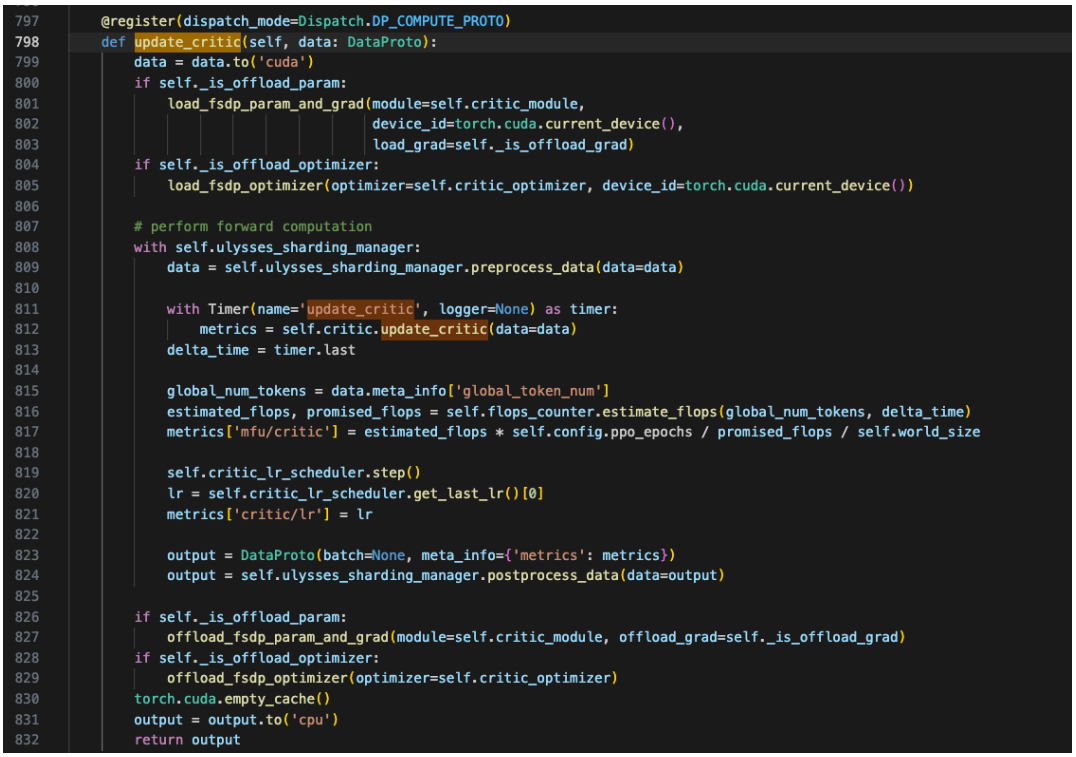
==========02/27==========

1

critic training(verl/trainer/ppo/ray_trainer.py) line 902

```python
            # update critic
            if self.use_critic:
                with _timer('update_critic', timing_raw):
                    critic_output = self.critic_wg.update_critic(batch)
                critic_output_metrics = reduce_metrics(critic_output.meta_info['metrics'])
                metrics.update(critic_output_metrics)
```

i)def update_critic (verl/workers/fsdp_workers.py)



```
797     @register(dispatch_mode=Dispatch.DP_COMPUTE_PROTO)
798     def update_critic(self, data: DataProto):
799         data = data.to('cuda')
800         if self._is_offload_param:
801             load_fsdp_param_and_grad(module=self.critic_module,
802                                      device_id=torch.cuda.current_device(),
803                                      load_grad=self._is_offload_grad)
804         if self._is_offload_optimizer:
805             load_fsdp_optimizer(optimizer=self.critic_optimizer, device_id=torch.cuda.current_device())
806 
807         # perform forward computation
808         with self.ulysses_sharding_manager:
809             data = self.ulysses_sharding_manager.preprocess_data(data=data)
810 
811             with Timer(name='update_critic', logger=None) as timer:
812                 metrics = self.critic.update_critic(data=data)
813             delta_time = timer.last
814 
815             global_num_tokens = data.meta_info['global_token_num']
816             estimated_flops, promised_flops = self.flops_counter.estimate_flops(global_num_tokens, delta_time)
817             metrics['mfu/critic'] = estimated_flops * self.config.ppo_epochs / promised_flops / self.world_size
818 
819             self.critic_lr_scheduler.step()
820             lr = self.critic_lr_scheduler.get_last_lr()[0]
821             metrics['critic/lr'] = lr
822 
823             output = DataProto(batch=None, meta_info={'metrics': metrics})
824             output = self.ulysses_sharding_manager.postprocess_data(data=output)
825 
826         if self._is_offload_param:
827             offload_fsdp_param_and_grad(module=self.critic_module, offload_grad=self._is_offload_grad)
828         if self._is_offload_optimizer:
829             offload_fsdp_optimizer(optimizer=self.critic_optimizer)
830         torch.cuda.empty_cache()
831         output = output.to('cpu')
832         return output
```

注意line 812(verl/workers/critic/dp_critic.py)

ii)问题20: self.ulysses_sharding_manager的理解(25/02/13, 25/02/27)

iii)问题21:DataParallelPPOCritic::update_critic(verl/workers/critic/dp_critic.py)很重要,为什么line 157~line 161 要split mini_batch, 同时

```
39  class DataParallelPPOCritic(BasePPOCritic):
143     def update_critic(self, data: DataProto):
154         for batch_idx, data in enumerate(dataloader):
155             # split batch into micro_batches
156             mini_batch = data
157             if self.config.use_dynamic_bsz:
158                 max_token_len = self.config.ppo_max_token_len_per_gpu * self.ulysses_sequence_parallel_size
159                 micro_batches, _ = rearrange_micro_batches(batch=mini_batch, max_token_len=max_token_len)
160             else:
161                 micro_batches = mini_batch.split(self.config.ppo_micro_batch_size_per_gpu)
162                 self.gradient_accumulation = self.config.ppo_mini_batch_size // self.config.ppo_micro_batch_size_per_gpu
163
164             self.critic_optimizer.zero_grad()
165
166             for data in micro_batches:
167                 data = data.cuda()  # critic device is cpu when using offload
168                 input_ids = data['input_ids']
169                 responses = data['responses']
170                 attention_mask = data['attention_mask']
171                 position_ids = data['position_ids']
172                 values = data['values']
173                 returns = data['returns']
174                 response_length = responses.size(1)
175
176                 eos_mask = attention_mask[:, -response_length - 1:-1]
177
178                 vpreds = self._forward_micro_batch(data)
179
180                 # assert not torch.any(torch.isnan(vpreds)).item()
181
182                 vf_loss, vf_clipfrac = core_algos.compute_value_loss(vpreds=vpreds,
183                                                                      values=values,
184                                                                      returns=returns,
185                                                                      eos_mask=eos_mask,
186                                                                      cliprange_value=self.config.cliprange_value)
187                 if self.config.use_dynamic_bsz:
188                     # relative to the dynamic bsz
189                     loss = vf_loss * (len(data) / self.config.ppo_mini_batch_size)
190                 else:
191                     loss = vf_loss / self.gradient_accumulation
192
193                 loss.backward()
```

2 actor training(verl/trainer/ppo/ray_trainer.py) line 910,

```
905
906                 # implement critic warmup
907                 if self.config.trainer.critic_warmup <= self.global_steps:
908                     # update actor
909                     with _timer('update_actor', timing_raw):
910                         actor_output = self.actor_rollout_wg.update_actor(batch)
911                     actor_output_metrics = reduce_metrics(actor_output.meta_info['metrics'])
912                     metrics.update(actor_output_metrics)
913
914                 # validate
```

==========

a)ActorRolloutRefWorker::update_actor(verl/workers/fsdp_workers.py) → DataParallelPPOActor::update_policy(verl/workers/actor/dp_actor.py)很重要

```python
class ActorRolloutRefWorker(Worker):
    def update_actor(self, data: DataProto):
        if self._is_offload_optimizer:
            load_fsdp_optimizer(optimizer=self.actor_optimizer, device_id=torch.cuda.current_device())

        data.batch = data.batch.cuda()

        log_gpu_memory_usage('Before update policy', logger=logger)

        with self.ulysses_sharding_manager:
            data = self.ulysses_sharding_manager.preprocess_data(data=data)
            # perform training
            with Timer(name='update_policy', logger=None) as timer:
                metrics = self.actor.update_policy(data=data)
            delta_time = timer.last
            global_num_tokens = data.meta_info['global_token_num']
            estimated_flops, promised_flops = self.flops_counter.estimate_flops(global_num_tokens, delta_time)
            metrics['mfu/actor'] = estimated_flops * self.config.actor.ppo_epochs / promised_flops / self.world_size

            self.actor_lr_scheduler.step()
            lr = self.actor_lr_scheduler.get_last_lr()[0]
            metrics['actor/lr'] = lr

            log_gpu_memory_usage('After update policy', logger=logger)

            # TODO: here, we should return all metrics
            output = DataProto(meta_info={'metrics': metrics})

            output = self.ulysses_sharding_manager.postprocess_data(data=output)
            output = output.to('cpu')

        if self._is_offload_param:
            offload_fsdp_param_and_grad(module=self.actor_module_fsdp, offload_grad=self._is_offload_grad)
        if self._is_offload_optimizer:
            offload_fsdp_optimizer(optimizer=self.actor_optimizer)
        torch.cuda.empty_cache()
        return output
```

7)这个data的作用

3

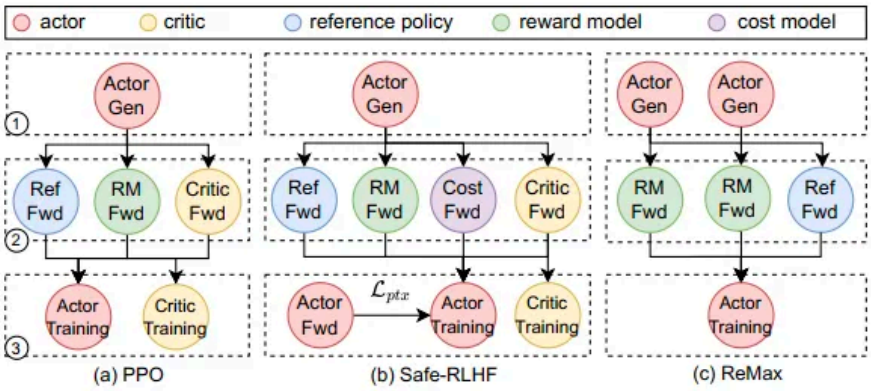


Figure 1. Dataflow graph of 3 RLHF algorithms [19, 43, 55]. Stage ①, ②, ③ represent Generation, Preparation, and Training, respectively.

这里的Actor Gen和Actor Training是同一个model， Actor training会更新model weight, 然后接着Actor Gen会用更新后的weight 来generate tokens, 因为Actor Gen和Actor training

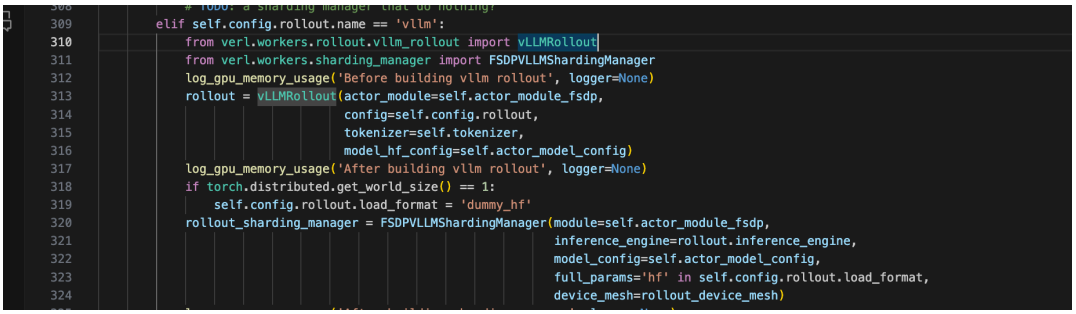
是不同的torch group, 此时Actor Gen的weight会broadcast到Actor training, 同时Actor training训练完了后，它的weight也会broadcast到actor gen, 目前verl的设计是基于ray的，此时ray会将actor gen在gpu的weight拷贝到cpu, 然后再拷贝到actor training的torch group, 这部分代码在哪？

1)在ActorRolloutRefWorker::init_model(verl/workers/fsdp_workers.py), Actor::gen的torch group的初始化

```
373         # load from checkpoint
374         if self._is_actor:
375             OmegaConf.set_struct(self.config.actor, True)
376             with open_dict(self.config.actor):
377                 self.config.actor.use_remove_padding = use_remove_padding
378             self.actor = DataParallelPPOActor(config=self.config.actor,
379                                               actor_module=self.actor_module_fsdp,
380                                               actor_optimizer=self.actor_optimizer)
381
382         if self._is_rollout:
383             self.rollout, self.rollout_sharding_manager = self._build_rollout()
384
385         if self._is_ref:
386             self.ref_module_fsdp = self._build_model_optimizer(model_path=self.config.model.path,
387                                                                 fsdp_config=self.config.ref.fsdp_config,
388                                                                 optim_config=None,
```

line 383 初始化self.rollout(是vllm engine)

ActorRolloutRefWorker::_build_rollout, line 313 初始化vLLMRollout



```
309         elif self.config.rollout.name == 'vllm':
310             from verl.workers.rollout.vllm_rollout import vLLMRollout
311             from verl.workers.sharding_manager import FSDPVLLMShardingManager
312             log_gpu_memory_usage('Before building vllm rollout', logger=None)
313             rollout = vLLMRollout(actor_module=self.actor_module_fsdp,
314                                   config=self.config.rollout,
315                                   tokenizer=self.tokenizer,
316                                   model_hf_config=self.actor_model_config)
317             log_gpu_memory_usage('After building vllm rollout', logger=None)
318             if torch.distributed.get_world_size() == 1:
319                 self.config.rollout.load_format = 'dummy_hf'
320             rollout_sharding_manager = FSDPVLLMShardingManager(module=self.actor_module_fsdp,
321                                                                inference_engine=rollout.inference_engine,
322                                                                model_config=self.actor_model_config,
323                                                                full_params='hf' in self.config.rollout.load_format,
324                                                                device_mesh=rollout_device_mesh)
```

一路调用→LLM::init(verl/third_party/vllm/vllm_v_0_6_3/llm.py) →SPMDGPUExecutor::init(verl/third_party/vllm/vllm_v_0_6_3/spmd_gpu_executor.py)
→Worker::init_device(/home/xiaoxiang/verl/verl/third_party/vllm/vllm_v_0_6_3/worker.py)

```python
class SPMDGPUExecutor(ExecutorBase):
    def _init_workers_sp(self, model, distributed_init_method: str):
        local_rank = int(os.getenv("LOCAL_RANK"))
        print(f"local rank {local_rank}")

        # see https://github.com/NVIDIA/nccl/issues/1234
        os.environ["NCCL_CUMEM_ENABLE"] = "0"

        self.worker = Worker(
            model,
            self.model_config,
            self.parallel_config,
            self.scheduler_config,
            self.device_config,
            self.cache_config,
            self.load_config,
            local_rank,
            rank,
            distributed_init_method,
            lora_config=self.lora_config,
            speculative_config=None,
            prompt_adapter_config=self.speculative_config,
            is_driver_worker=True,
            model_runner_cls=None,  # use the default one
        )

        # NOTE(shengguangming): torch.distributed.init_process_group will be called inside the init_model()
        self.worker.init_device()
        self.worker.load_model()

    def determine_num_available_blocks(self) -> Tuple[int, int]:
        """Determine the number of available KV blocks.

        This invokes `determine_num_available_blocks` on each worker and takes
        the min of the results, guaranteeing that the selected cache sizes are
        compatible with all workers.
```

```python
class Worker(Worker):
    def init_device(self) -> None:
        if self.device_config.device.type == "cuda":
            # torch.distributed.all_reduce does not free the input tensor until
            # the synchronization point. This causes the memory usage to grow
            # as the number of all_reduce calls increases. This env var disables
            # this behavior.
            # Related issue:
            # https://discuss.pytorch.org/t/cuda-allocation-lifetime-for-inputs-to-distributed-all-reduce/191573
            os.environ["TORCH_NCCL_AVOID_RECORD_STREAMS"] = "1"

            # NOTE(sgm): Modify for verl, Env vars will be set by TORCHRUN.
            self.rank = self.rank if self.rank is not None else int(os.getenv("RANK", "-1"))
            local_rank = int(os.getenv("LOCAL_RANK", "0"))
            self.device = torch.device(f"cuda:{local_rank}")
            if self.rank < 0:
                raise ValueError("Invalid or unspecified rank.")
            torch.cuda.set_device(self.device)

            # Use the world_size set by TORCHRUN
            world_size = int(os.getenv("WORLD_SIZE", "-1"))
            assert world_size != -1, "The world_size is set to -1, not initialized by TORCHRUN"
            self.parallel_config.world_size = world_size

            _check_if_gpu_supports_dtype(self.model_config.dtype)
            torch.cuda.empty_cache()
            self.init_gpu_memory = torch.cuda.mem_get_info()[0]
        else:
            raise RuntimeError(f"Not support device type: {self.device_config.device}")

        # Initialize the distributed environment.
        init_worker_distributed_environment(self.parallel_config, self.rank, self.distributed_init_method,
                                            self.local_rank)
        # Set random seed.
        set_random_seed(self.model_config.seed)
        # self.model = get_model(actor_model=self.model, model_config=self.model_config)
```

2)

==========02/28==========

问题25: 哪些参数的作用

问题26: Actor trainer和Actor gen 如何sync weight

1

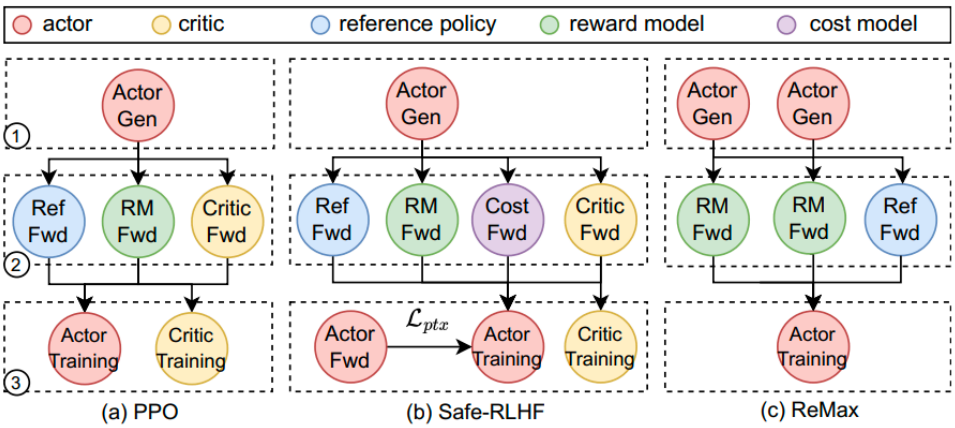


Figure 1. Dataflow graph of 3 RLHF algorithms [19, 43, 55]. Stage ①, ②, ③ represent Generation, Preparation, and Training, respectively.

在这里，Actor gen和actor training是同一个model, training更新model weight后，需要sync weight 给gen

1)verl/trainer/main_ppo.py

```
68
69      from verl.trainer.ppo.ray_trainer import ResourcePoolManager, Role
70
71      role_worker_mapping = {
72          Role.ActorRollout: ray.remote(ActorRolloutRefWorker),
73          Role.Critic: ray.remote(CriticWorker),
74          Role.RefPolicy: ray.remote(ActorRolloutRefWorker)
75      }
76
77      global_pool_id = 'global_pool'
78      resource_pool_spec = {
79          global_pool_id: [config.trainer.n_gpus_per_node] * config.trainer.nnodes,
80      }
81      mapping = {
82          Role.ActorRollout: global_pool_id,
83          Role.Critic: global_pool_id,
84          Role.RefPolicy: global_pool_id,
85      }
```

这里Actor gen和actor trainer在同一个worker group

2)创建Actor trainer

i)verl/trainer/ppo/ray_trainer.py

```
692             self.rm_wg = all_wg['rm']
693             self.rm_wg.init_model()
694
695         # we should create rollout at the end so that vllm can have a better estimation of kv
696         self.actor_rollout_wg = all_wg['actor_rollout']
697         print(f"1 verl/trainer/ppo/ray_trainer.py, RayPPOTrainer::init_workers, self.actor_rol
698         self.actor_rollout_wg.init_model()
699
```

self.actor_rollout_wg是class ActorRolloutRefWorker

ii)verl/workers/fsdp_workers.py

```
355         self.actor_module_fsdp, self.actor_optimizer, self.actor_lr_scheduler, self.actor_model_config = self._build_model_optimizer(
356             model_path=self.config.model.path,
357             fsdp_config=fsdp_config,
358             optim_config=optim_config,
359             override_model_config=override_model_config,
360             use_remove_padding=use_remove_padding,
361             enable_gradient_checkpointing=self.config.model.get('enable_gradient_checkpointing', False),
362             trust_remote_code=self.config.model.get('trust_remote_code', False),
363             use_liger=self.config.model.get('use_liger', False),
364             role='actor')
365
```

self._build_model_optimizer是创建了aself.actor_moule_fsdp

a)self._build_model_optimizer

```
206
207     with init_context(), warnings.catch_warnings():
208         warnings.simplefilter("ignore")
209         actor_module = AutoModelForCausalLM.from_pretrained(pretrained_model_name_or_path=local_path,
210                                                             torch_dtype=torch_dtype,
211                                                             config=actor_model_config,
212                                                             attn_implementation='flash_attention_2',
213                                                             trust_remote_code=trust_remote_code)
214         # Apply Liger kernel to the model if use_liger is set to True
215         if use_liger:
216             from liger_kernel.transformers.monkey_patch import _apply_liger_kernel_to_instance
217             _apply_liger_kernel_to_instance(model=actor_module)
218
219         # some parameters may not in torch_dtype. TODO(zhangchi.usc1992) remove this after we switch to fsdp2
220         actor_module.to(torch_dtype)
221
```

AutoModelForCausalLM.from_pretrained生成这个model的actor_module

问题27: actor_module 在cpu上还是gpu上

```
257         # We force reference policy to use CPUOffload to save memory.
258         # We force turn off CPUOffload for actor because it causes incorrect results when using grad accumulation
259         cpu_offload = None if role == 'actor' else CPUOffload(offload_params=True)
260         actor_module_fsdp = FSDP(
261             actor_module,
262             cpu_offload=cpu_offload,
263             param_init_fn=init_fn,
264             use_orig_params=False,
265             auto_wrap_policy=auto_wrap_policy,
266             device_id=torch.cuda.current_device(),#Xiao 02/27, 放到gpu上
267             sharding_strategy=sharding_strategy,  # zero3
268             mixed_precision=mixed_precision,
269             sync_module_states=True,
270             device_mesh=self.device_mesh,
271             forward_prefetch=False)
272
```

line 260利用FSDP创建actor_module_fsdp

iii)回到ActorRolloutRefWorker::init_model

a) line 368很重要

```
366
367             # get the original unwrapped module
368             self.actor_module = self.actor_module_fsdp._fsdp_wrapped_module
369
```

b)

```
377         # load from checkpoint
378         if self._is_actor:
379             OmegaConf.set_struct(self.config.actor, True)
380             with open_dict(self.config.actor):
381                 self.config.actor.use_remove_padding = use_remove_padding
382             self.actor = DataParallelPPOActor(config=self.config.actor,
383                                               actor_module=self.actor_module_fsdp,
384                                               actor_optimizer=self.actor_optimizer)
385
```

line 382 利用self.actor_module_fsdp创建的self.actor, 这个是trainer

c) 去self._build_rollout, rollout就是Actor gen

```
384                                               actor_optimizer=self.actor_optimizer)
385
386         if self._is_rollout:
387             self.rollout, self.rollout_sharding_manager = self._build_rollout()
```

```
298
299     def _build_rollout(self):
300         from torch.distributed.device_mesh import init_device_mesh
301         # TODO(sgm): support FSDP hybrid shard for larger model
302         infer_tp = self.config.rollout.tensor_model_parallel_size
303         dp = self.world_size // infer_tp
304         assert self.world_size % infer_tp == 0, f'rollout world_size: {self.world_size} is not divisible by infer_tp: {infer_tp}'
305         rollout_device_mesh = init_device_mesh('cuda', mesh_shape=(dp, infer_tp), mesh_dim_names=['dp', 'infer_tp'])
306
```

似乎是只有rollout采用了tp

```
302         infer_tp = self.config.rollout.tensor_model_parallel_size
303         dp = self.world_size // infer_tp
304         assert self.world_size % infer_tp == 0, f'rollout world_size: {self.world_size} is not divisible by infer_tp: {infer_tp}'
305         rollout_device_mesh = init_device_mesh('cuda', mesh_shape=(dp, infer_tp), mesh_dim_names=['dp', 'infer_tp'])
306 
307         if self.config.rollout.name == 'hf':
308             from verl.workers.rollout import HFRollout
309             from verl.workers.sharding_manager import BaseShardingManager
310             rollout = HFRollout(module=self.actor_module_fsdp, config=self.config.rollout)
311             rollout_sharding_manager = BaseShardingManager()
312             # TODO: a sharding manager that do nothing?
313         elif self.config.rollout.name == 'vllm':
314             from verl.workers.rollout.vllm_rollout import vLLMRollout
315             from verl.workers.sharding_manager import FSDPVLLMShardingManager
316             log_gpu_memory_usage('Before building vllm rollout', logger=None)
317             rollout = vLLMRollout(actor_module=self.actor_module_fsdp,
318                                   config=self.config.rollout,
319                                   tokenizer=self.tokenizer,
320                                   model_hf_config=self.actor_model_config)
321             log_gpu_memory_usage('After building vllm rollout', logger=None)
322             if torch.distributed.get_world_size() == 1:
323                 self.config.rollout.load_format = 'dummy_hf'
324             rollout_sharding_manager = FSDPVLLMShardingManager(module=self.actor_module_fsdp,
325                                                                inference_engine=rollout.inference_engine,
326                                                                model_config=self.actor_model_config,
327                                                                full_params='hf' in self.config.rollout.load_format,
328                                                                device_mesh=rollout_device_mesh)
329             log_gpu_memory_usage('After building sharding manager', logger=None)
330 
331         return rollout, rollout_sharding_manager
```

line 317同样用·self.actor_module_fsdp来创建self.rolllout和rollout_sharding_manager ,注意rollout_sharding_manager很重要

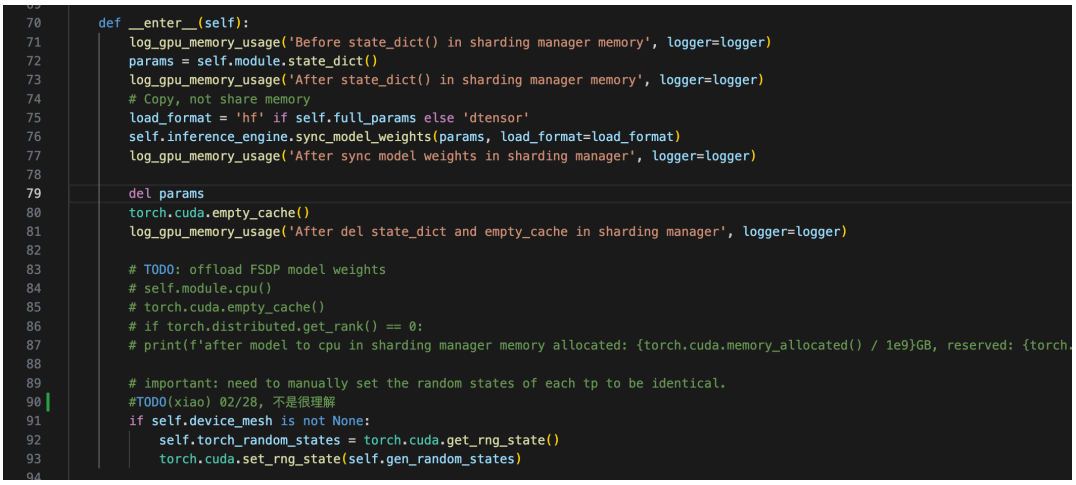
---

iv)rollout和actor如何sync_weight

```
73  class ActorRolloutRefWorker(Worker):
459     def generate_sequences(self, prompts: DataProto):
473                 if self.generation_config is not None else self.tokenizer.eos_token_id,
474             'pad_token_id':
475                 self.generation_config.pad_token_id
476                 if self.generation_config is not None else self.tokenizer.pad_token_id,
477         }
478         prompts.meta_info.update(meta_info)
479         with self.rollout_sharding_manager:
480             log_gpu_memory_usage('After entering rollout sharding manager', logger=logger)
481 
482             prompts = self.rollout_sharding_manager.preprocess_data(prompts)#TODO(xiao) 25/02/12, do not understand this, why this,
483             output = self.rollout.generate_sequences(prompts=prompts) #TODO(xiao) 25/02/12, do not understand this, why this, very i
484             """
485             #TODO(Xiao): 02/27, 这里的rollout初始化是
486             rollout = vLLMRollout(actor_module=self.actor_module_fsdp,
487                       config=self.config.rollout,
488                       tokenizer=self.tokenizer,
489                       model_hf_config=self.actor_model_config)
490             """
491 
492             log_gpu_memory_usage('After rollout generation', logger=logger)
493 
494             output = self.rollout_sharding_manager.postprocess_data(output)
495 
496         output = output.to('cpu')
497 
```

当rollout call generate_sequences, 会call line 479，然后进 入verl/workers/sharding_manager/fsdp_vllm.py

a)FSDPVLLMShardingManager::__enter__(/home/xiaoxiang/verl/verl/workers/sharding_manager/fsdp_vllm.py)



```python
    def __enter__(self):
        log_gpu_memory_usage('Before state_dict() in sharding manager memory', logger=logger)
        params = self.module.state_dict()
        log_gpu_memory_usage('After state_dict() in sharding manager memory', logger=logger)
        # Copy, not share memory
        load_format = 'hf' if self.full_params else 'dtensor'
        self.inference_engine.sync_model_weights(params, load_format=load_format)
        log_gpu_memory_usage('After sync model weights in sharding manager', logger=logger)

        del params
        torch.cuda.empty_cache()
        log_gpu_memory_usage('After del state_dict and empty_cache in sharding manager', logger=logger)

        # TODO: offload FSDP model weights
        # self.module.cpu()
        # torch.cuda.empty_cache()
        # if torch.distributed.get_rank() == 0:
        # print(f'after model to cpu in sharding manager memory allocated: {torch.cuda.memory_allocated() / 1e9}GB, reserved: {torch.

        # important: need to manually set the random states of each tp to be identical.
        #TODO(xiao) 02/28, 不是很理解
        if self.device_mesh is not None:
            self.torch_random_states = torch.cuda.get_rng_state()
            torch.cuda.set_rng_state(self.gen_random_states)
```

line 73：获取self.actor_module_fsdp的param weight, 因为self.rollout和self.actor
在一个worker group, 利用FSDP，这两个传递指针，

line 76: sync weight, 一路调用到verl/third_party/vllm/vllm_v_0_6_3/worker.py

```
 # assume the input is .state_dict()
def sync_model_weights(self, actor_weights: Dict, load_format: str):
    print(f"1 verl/third_party/vllm/vllm_v_0_6_3/worker.py, sync_model_weights, load_format: {load_format}")
    if load_format in [LoadFormat.MEGATRON, LoadFormat.AUTO]:
        load_megatron_weights(actor_weights, self.model_runner.model)
    elif load_format == LoadFormat.HF:
        # full model state dict without no sharding
        load_hf_weights(actor_weights, self.model_runner.model)
    elif load_format == LoadFormat.DTENSOR:
        load_dtensor_weights(actor_weights, self.model_runner.model)
```

这个对我的优化很严重

在分布式训练中，**随机种子（random seed）** 和 **随机状态（random state）** 的管理非常重要，尤其是在使用 **模型并行（Model Parallelism）** 或 **数据并行（Data Parallelism）** 时。以下是对代码中随机状态管理的详细解释：

---

1. 随机状态的作用

- **随机种子**：
  - 用于初始化随机数生成器，确保每次运行程序时生成的随机数序列是相同的。
- **随机状态**：
  - 随机数生成器的内部状态，决定了下一个随机数的生成。

- 在分布式训练中，不同进程的随机状态需要保持一致，以确保模型的行为是可复现的。

---

2. 代码中的随机状态管理

1. 初始化随机状态

在 `__init__` 方法中，代码设置了随机种子并保存了随机状态：

```
if self.device_mesh is not None:
    gen_dp_rank = self.device_mesh['dp'].get_local_rank()
    torch.cuda.manual_seed(gen_dp_rank + 1000)  # 设置随机种子
    self.gen_random_states = torch.cuda.get_rng_state()  # 保存随机状态
    torch.cuda.set_rng_state(self.torch_random_states)  # 恢复之前的随机状态
```

- `torch.cuda.manual_seed`：
  - 为当前 GPU 设置随机种子。
  - `gen_dp_rank + 1000` 确保不同 GPU 的随机种子不同，但仍然是确定性的。
- `self.gen_random_states`：
  - 保存当前 GPU 的随机状态。
- `torch.cuda.set_rng_state`：
  - 恢复之前的随机状态，确保随机数生成器的行为一致。

2. 在 `__enter__` 方法中恢复随机状态

在 `__enter__` 方法中，代码再次恢复随机状态：

```
if self.device_mesh is not None:
    self.torch_random_states = torch.cuda.get_rng_state()  # 保存当前随机状态
    torch.cuda.set_rng_state(self.gen_random_states)  # 恢复初始化时的随机状态
```

- `self.torch_random_states`：
  - 保存当前 GPU 的随机状态。
- `torch.cuda.set_rng_state(self.gen_random_states)`：
  - 恢复初始化时保存的随机状态。

---

3. 为什么需要在 `sync_model_weights` 后恢复随机状态？

1. `sync_model_weights` 的可能影响

- `sync_model_weights` 是一个同步操作，可能会涉及 GPU 上的计算或数据传输。
- 在这个过程中，GPU 的随机状态可能会被修改（例如，某些操作可能会隐式地使用随机数生成器）。

2. 确保随机状态的一致性

- 在分布式训练中，不同 GPU 的随机状态需要保持一致，以确保模型的行为是可复现的。

- 如果在 `sync_model_weights` 后不恢复随机状态，不同 GPU 的随机状态可能会不一致，导致模型的行为不可预测。

3. 恢复随机状态的作用

- 通过恢复初始化时保存的随机状态，确保所有 GPU 的随机状态保持一致。
- 这样可以避免由于随机状态不一致导致的模型行为差异。

---

4. 总结

- 在分布式训练中，随机状态的管理非常重要，尤其是在涉及 GPU 计算或数据传输的操作（如 `sync_model_weights`）之后。
- 代码中通过保存和恢复随机状态，确保所有 GPU 的随机状态保持一致，从而保证模型的行为是可复现的。

希望这个解释对你有帮助！如果还有其他问题，欢迎继续提问！

==========02/28==========

还有疑惑的代码

1 1)verl/single_controller/base/decorator.py（解决了02/28)

def register解释

```
393
394 def register(dispatch_mode=Dispatch.ALL_TO_ALL, execute_mode=Execute.ALL, blocking=True, materialize_futures=True):
395     _check_dispatch_mode(dispatch_mode=dispatch_mode)
396     _check_execute_mode(execute_mode=execute_mode)
397
398     def decorator(func):
399
400         @wraps(func)
401         def inner(*args, **kwargs):
402             if materialize_futures:
403                 args, kwargs = _materialize_futures(*args, **kwargs)
404             return func(*args, **kwargs)
405
406         attrs = {'dispatch_mode': dispatch_mode, 'execute_mode': execute_mode, 'blocking': blocking}
407         setattr(inner, MAGIC_ATTR, attrs)
408         return inner
409
410     return decorator
411
```

- `register` **的作用**：
  - 接受一些参数（如 `dispatch_mode`、`execute_mode` 等）。
  - 返回一个装饰器函数 `decorator`。
- `decorator` **的作用**：
  - 接受一个函数 `func` 作为参数。
  - 返回一个新的函数 `inner`，它会在调用 `func` 之前执行一些逻辑（如 `_materialize_futures`）。
  - 为 `inner` 函数设置一些属性（如 `dispatch_mode`、`execute_mode` 等）。

•

`@register` 是 Python 的装饰器语法，它会调用 `register` 函数，并将 `execute_with_func_generator` 作为参数传递给 `register` 返回的装饰器。

2)verl/single_controller/base/worker_group.py, WorkerGroup::_bind_worker_method

3)verl/single_controller/ray/base.py

4)verl/trainer/ppo/core_algos.py是RL相关的数学计算，可以先跳过

5)verl/trainer/ppo/ray_trainer.py RPOTrainer 优先级没那么重要

a)def apply_kl_penalty

b)def compute_advantage

6)verl/trainer/fsdp_sft_trainer.py

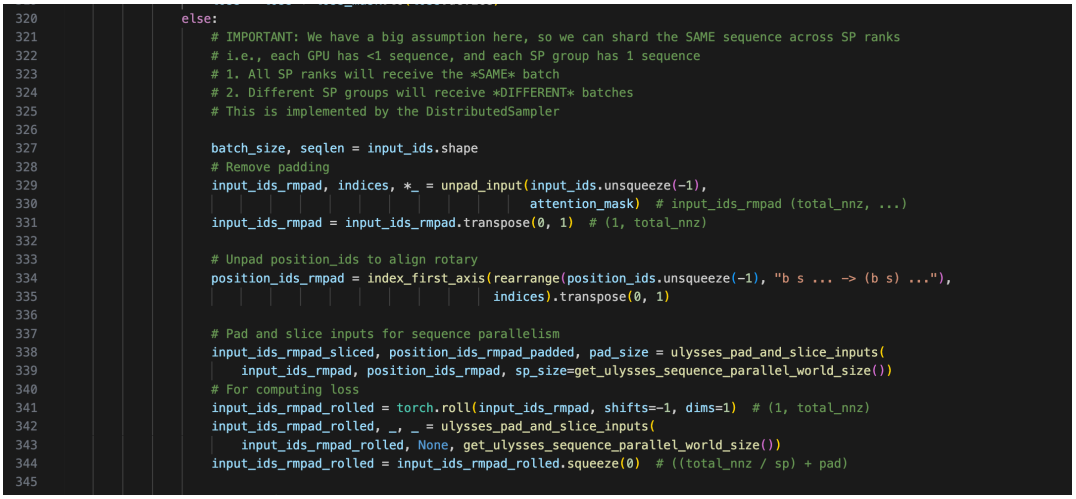
FSDP::_compute_loss_and_backward 优先级没那么重要

一些计算细节

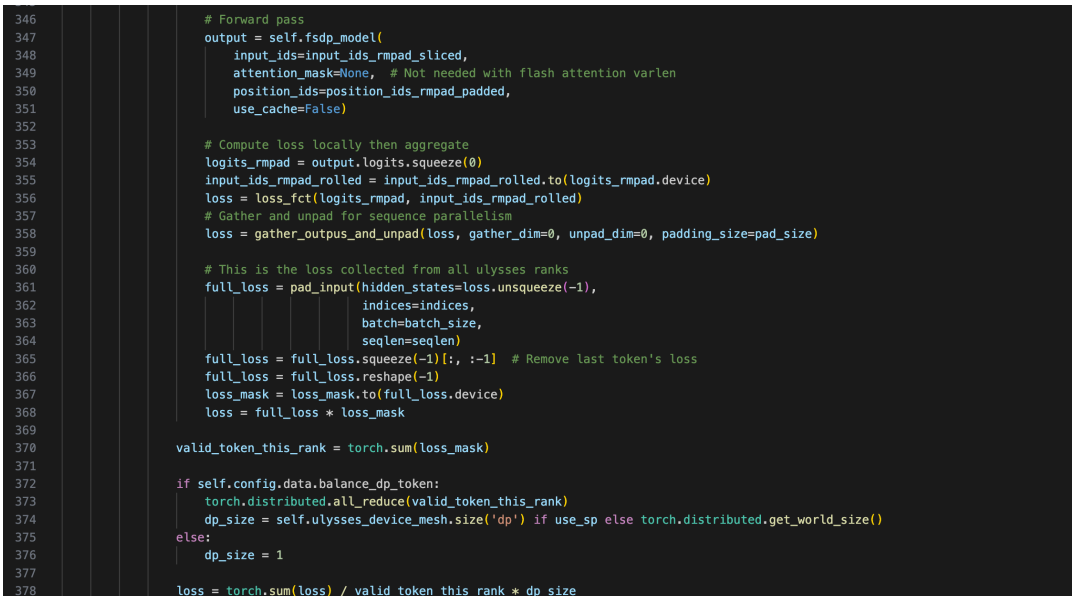
问题28: 不支持sp的时候，是这样计算， line 311不是很懂，为什么是shift_logits = logits[..., :-1, :].contiguous()

```
301                 with torch.autocast(device_type='cuda', dtype=torch.bfloat16):
302                     if not use_sp:
303                         # Standard forward pass without sequence parallel
304                         labels = input_ids[:, 1:].contiguous()
305                         output = self.fsdp_model(input_ids=input_ids,
306                                                  attention_mask=attention_mask,
307                                                  position_ids=position_ids,
308                                                  use_cache=False)
309                         logits = output.logits
310 
311                         shift_logits = logits[..., :-1, :].contiguous()
312                         shift_labels = labels.contiguous()
313                         # Flatten the tokens
314                         shift_logits = shift_logits.view(-1, self.model.config.vocab_size)
315                         shift_labels = shift_labels.view(-1)
316                         # Enable model parallelism
317                         shift_labels = shift_labels.to(shift_logits.device)
318                         loss = loss_fct(shift_logits, shift_labels)
319                         loss = loss * loss_mask.to(loss.device)
```

支持sp的时候, line 327~ line 378不是很懂，我只能直
接拿来用，抄



```
            else:
                # IMPORTANT: We have a big assumption here, so we can shard the SAME sequence across SP ranks
                # i.e., each GPU has <1 sequence, and each SP group has 1 sequence
                # 1. All SP ranks will receive the *SAME* batch
                # 2. Different SP groups will receive *DIFFERENT* batches
                # This is implemented by the DistributedSampler

                batch_size, seqlen = input_ids.shape
                # Remove padding
                input_ids_rmpad, indices, *_ = unpad_input(input_ids.unsqueeze(-1),
                                                           attention_mask)  # input_ids_rmpad (total_nnz, ...)
                input_ids_rmpad = input_ids_rmpad.transpose(0, 1)  # (1, total_nnz)

                # Unpad position_ids to align rotary
                position_ids_rmpad = index_first_axis(rearrange(position_ids.unsqueeze(-1), "b s ... -> (b s) ..."),
                                                      indices).transpose(0, 1)

                # Pad and slice inputs for sequence parallelism
                input_ids_rmpad_sliced, position_ids_rmpad_padded, pad_size = ulysses_pad_and_slice_inputs(
                    input_ids_rmpad, position_ids_rmpad, sp_size=get_ulysses_sequence_parallel_world_size())
                # For computing loss
                input_ids_rmpad_rolled = torch.roll(input_ids_rmpad, shifts=-1, dims=1)  # (1, total_nnz)
                input_ids_rmpad_rolled, _, _ = ulysses_pad_and_slice_inputs(
                    input_ids_rmpad_rolled, None, get_ulysses_sequence_parallel_world_size())
                input_ids_rmpad_rolled = input_ids_rmpad_rolled.squeeze(0)  # ((total_nnz / sp) + pad)

```



```
                # Forward pass
                output = self.fsdp_model(
                    input_ids=input_ids_rmpad_sliced,
                    attention_mask=None,  # Not needed with flash attention varlen
                    position_ids=position_ids_rmpad_padded,
                    use_cache=False)

                # Compute loss locally then aggregate
                logits_rmpad = output.logits.squeeze(0)
                input_ids_rmpad_rolled = input_ids_rmpad_rolled.to(logits_rmpad.device)
                loss = loss_fct(logits_rmpad, input_ids_rmpad_rolled)
                # Gather and unpad for sequence parallelism
                loss = gather_outpus_and_unpad(loss, gather_dim=0, unpad_dim=0, padding_size=pad_size)

                # This is the loss collected from all ulysses ranks
                full_loss = pad_input(hidden_states=loss.unsqueeze(-1),
                                      indices=indices,
                                      batch=batch_size,
                                      seqlen=seqlen)
                full_loss = full_loss.squeeze(-1)[:, :-1]  # Remove last token's loss
                full_loss = full_loss.reshape(-1)
                loss_mask = loss_mask.to(full_loss.device)
                loss = full_loss * loss_mask

            valid_token_this_rank = torch.sum(loss_mask)

            if self.config.data.balance_dp_token:
                torch.distributed.all_reduce(valid_token_this_rank)
                dp_size = self.ulysses_device_mesh.size('dp') if use_sp else torch.distributed.get_world_size()
            else:
                dp_size = 1

            loss = torch.sum(loss) / valid_token_this_rank * dp_size
```

7)verl/utils/fsdp_utils.py

8)verl/utils/model.py

a)def pad_packed_inputs

9)verl/utils/ulysses.py 需要很了解sequence parallel

10)verl/workers/actor/dp_actor.py DataParallelPPOActor::_forward_micro_batch, 可以直接抄， 这部分代码如果有空，得好好看看

11)verl/workers/critic/dp_critic.py DataParallelPPOCritic::_forward_micro_batch , 可以直接抄， 这部分代码如果有空，得好好看看

12)verl/workers/fsdp_workers.py

a)ActorRolloutRefWorker::update_actor, 很值得看******

b)ActorRolloutRefWorker::compute_log_prob, 这部分是ai相关，得看看为什么这么计算***

c)ActorRolloutRefWorker::compute_ref_log_prob，这部分是ai相关，得看看为什么这么计算

d)CriticWorker::compute_values, 这部分是ai相关，得看看为什么这么计算 ***

e)CriticWorker::update_critic，很值得看******

f)RewardModelWorker::compute_rm_score, 很值得看******

7

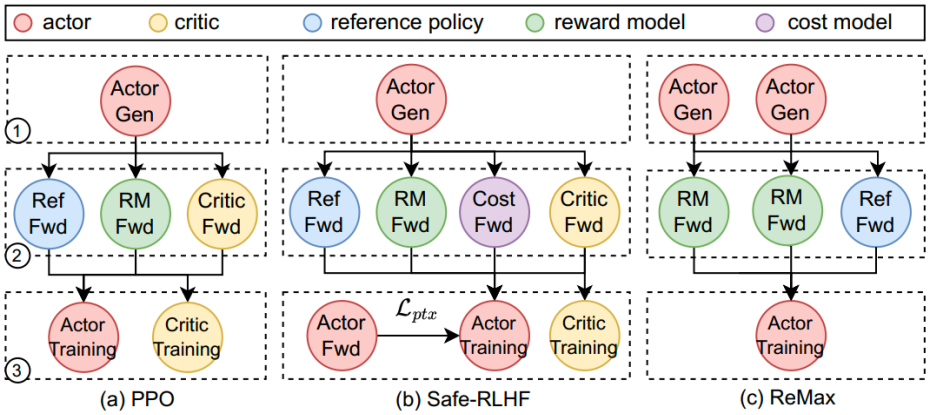


Figure 1. Dataflow graph of 3 RLHF algorithms [19, 43, 55]. Stage ①, ②, ③ represent Generation, Preparation, and Training, respectively.

比如说PPO算法，Actor(rollout) Gen后的output, 然后传给Ref/RM/Critic model,, 这三个model的input只跟Actor Gen有关

1）代码是这样的verl/trainer/ppo/ray_trainer.py

```
837            with _timer('step', timing_raw):
838                # generate a batch
839                with _timer('gen', timing_raw):
840                    gen_batch_output = self.actor_rollout_wg.generate_sequences(gen_batch)
841
842                batch.non_tensor_batch['uid'] = np.array([str(uuid.uuid4()) for _ in range(len(batch.batch))],
843                                                         dtype=object)
844                # repeat to align with repeated responses in rollout
845                batch = batch.repeat(repeat_times=self.config.actor_rollout_ref.rollout.n, interleave=True)
846                batch = batch.union(gen_batch_output) #TODO(xiao) 02/28, 这个gen_batch_output是rollout gen的，为什么要和batch union
847
848                # balance the number of valid tokens on each dp rank.
849                # Note that this breaks the order of data inside the batch.
850                # Please take care when you implement group based adv computation such as GRPO and rloo
851                self._balance_batch(batch, metrics=metrics)
852
853                # compute global_valid tokens
854                batch.meta_info['global_token_num'] = torch.sum(batch.batch['attention_mask'], dim=-1).tolist()
855
856                # recompute old_log_probs
857                with _timer('old_log_prob', timing_raw):
858                    old_log_prob = self.actor_rollout_wg.compute_log_prob(batch) #TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算log_prob
859                    batch = batch.union(old_log_prob)
860                    #TODO(xiao):2025-02-28,old_log_prob是self.actor_rollout_wg.compute_log_prob，为什么需要计算old_log_prob，为什么又要和batch union
```

2)然后Ref/RM/Critic model,,计算batch 注意

```
862                if self.use_reference_policy:
863                    # compute reference log_prob
864                    with _timer('ref', timing_raw):
865                        ref_log_prob = self.ref_policy_wg.compute_ref_log_prob(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算ref_log_prob
866                        batch = batch.union(ref_log_prob)
867
868                # compute values
869                if self.use_critic:
870                    with _timer('values', timing_raw):
871                        values = self.critic_wg.compute_values(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算values
872                        batch = batch.union(values)
873
874                with _timer('adv', timing_raw):
875                    # compute scores. Support both model and function-based.
876                    # We first compute the scores using reward model. Then, we call reward_fn to combine
877                    # the results from reward model and rule-based results.
878                    if self.use_rm:
879                        # we first compute reward model score
880                        reward_tensor = self.rm_wg.compute_rm_score(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算reward_tensor
881                        batch = batch.union(reward_tensor)
```

类似line 865, line 871, line 880最后都调用了ray.get, 这个batch刚开始的时候只是一个future, ray的作用

3)然后critic和actor分别训练更新权重, line 906和line 914 ,

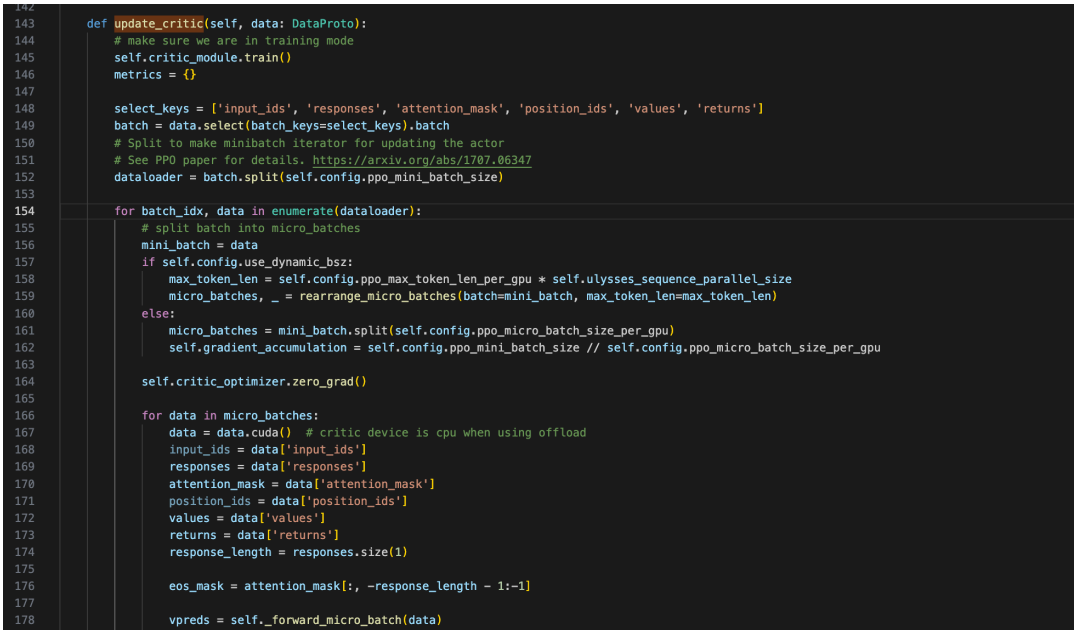
```
902
903                # update critic
904                if self.use_critic:
905                    with _timer('update_critic', timing_raw):
906                        critic_output = self.critic_wg.update_critic(batch)
907                    critic_output_metrics = reduce_metrics(critic_output.meta_info['metrics'])
908                    metrics.update(critic_output_metrics)
909
910                # implement critic warmup
911                if self.config.trainer.critic_warmup <= self.global_steps:
912                    # update actor
913                    with _timer('update_actor', timing_raw):
914                        actor_output = self.actor_rollout_wg.update_actor(batch)
915                    actor_output_metrics = reduce_metrics(actor_output.meta_info['metrics'])
916                    metrics.update(actor_output_metrics)
917
```

ActorRolloutRefWorker::update_actor和CriticWorker::update_critic都是@register装饰函数，最后都会用到ray.get, 这是一个阻塞操作，返回一个future

==========03/01==========

==========03/04==========

1 DataParallelPPOCritic::update_critic(verl/workers/critic/dp_critic.py)



```
142
143     def update_critic(self, data: DataProto):
144         # make sure we are in training mode
145         self.critic_module.train()
146         metrics = {}
147
148         select_keys = ['input_ids', 'responses', 'attention_mask', 'position_ids', 'values', 'returns']
149         batch = data.select(batch_keys=select_keys).batch
150         # Split to make minibatch iterator for updating the actor
151         # See PPO paper for details. https://arxiv.org/abs/1707.06347
152         dataloader = batch.split(self.config.ppo_mini_batch_size)
153
154         for batch_idx, data in enumerate(dataloader):
155             # split batch into micro_batches
156             mini_batch = data
157             if self.config.use_dynamic_bsz:
158                 max_token_len = self.config.ppo_max_token_len_per_gpu * self.ulysses_sequence_parallel_size
159                 micro_batches, _ = rearrange_micro_batches(batch=mini_batch, max_token_len=max_token_len)
160             else:
161                 micro_batches = mini_batch.split(self.config.ppo_micro_batch_size_per_gpu)
162                 self.gradient_accumulation = self.config.ppo_mini_batch_size // self.config.ppo_micro_batch_size_per_gpu
163
164             self.critic_optimizer.zero_grad()
165
166             for data in micro_batches:
167                 data = data.cuda()  # critic device is cpu when using offload
168                 input_ids = data['input_ids']
169                 responses = data['responses']
170                 attention_mask = data['attention_mask']
171                 position_ids = data['position_ids']
172                 values = data['values']
173                 returns = data['returns']
174                 response_length = responses.size(1)
175
176                 eos_mask = attention_mask[:, -response_length - 1:-1]
177
178                 vpreds = self._forward_micro_batch(data)
```

line 148这里的input_ids是来自prompt, attention_mask也是有的，然后response是actor gen的, 1)values是verl/trainer/ppo/ray_trainer.py

```
872
873                 # compute values
874                 if self.use_critic:
875                     with _timer('values', timing_raw):
876                         values = self.critic_wg.compute_values(batch)#TODO(xiao):2025-02-12, 这个函数的作用是啥,为什么计算values
877                         batch = batch.union(values)
878
```

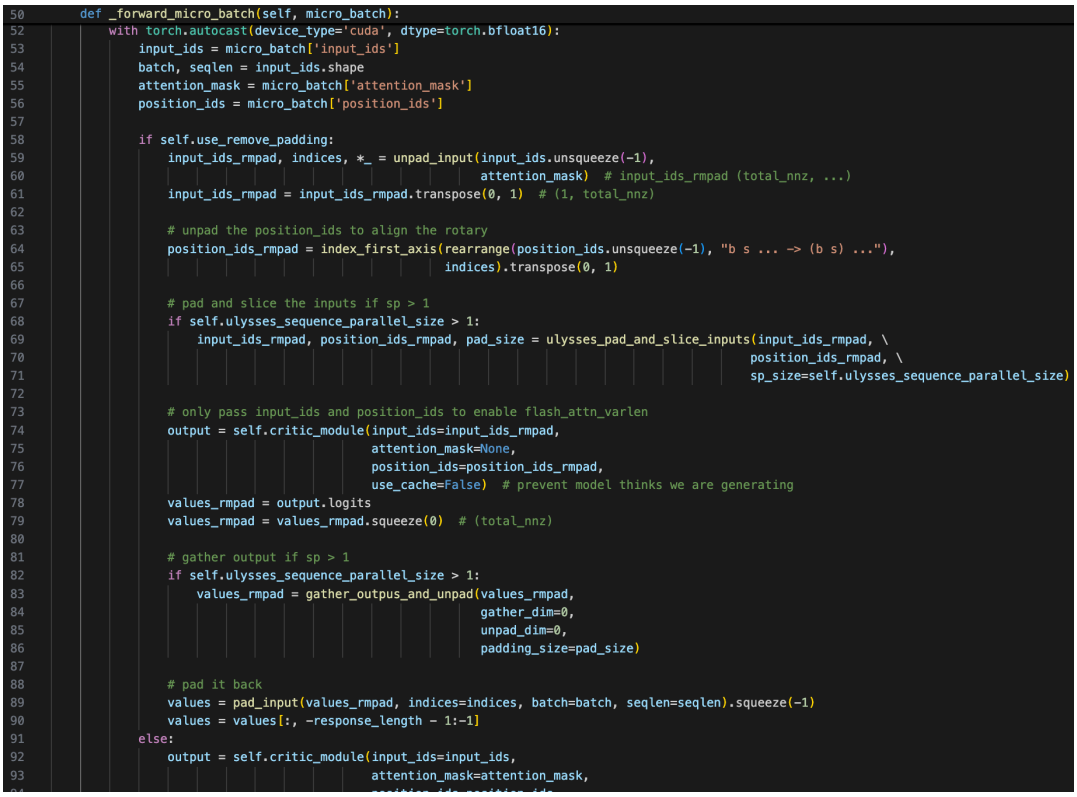
2)returns verl/trainer/ppo/ray_trainer.py

```python
def compute_advantage(data: DataProto, adv_estimator, gamma=1.0, lam=1.0, num_repeat=1):
    # prepare response group
    # TODO: add other ways to estimate advantages
    if adv_estimator == 'gae':
        values = data.batch['values']
        responses = data.batch['responses']
        response_length = responses.size(-1)
        attention_mask = data.batch['attention_mask']
        response_mask = attention_mask[:, -response_length:]
        token_level_rewards = data.batch['token_level_rewards'] #batch.batch['token_level_scores'] = reward_tensor,
        advantages, returns = core_algos.compute_gae_advantage_return(token_level_rewards=token_level_rewards,
                                                                      values=values,
                                                                      eos_mask=response_mask,
                                                                      gamma=gamma,
                                                                      lam=lam)
        data.batch['advantages'] = advantages
        data.batch['returns'] = returns
    elif adv_estimator == 'grpo':
        token_level_rewards = data.batch['token_level_rewards']
        index = data.non_tensor_batch['uid']
        responses = data.batch['responses']
        response_length = responses.size(-1)
        attention_mask = data.batch['attention_mask']
        response_mask = attention_mask[:, -response_length:]
        advantages, returns = core_algos.compute_grpo_outcome_advantage(token_level_rewards=token_level_rewards,
                                                                        eos_mask=response_mask,
                                                                        index=index)
        data.batch['advantages'] = advantages
        data.batch['returns'] = returns
    elif adv_estimator == 'reinforce_plus_plus':
        token_level_rewards = data.batch['token_level_rewards']
        responses = data.batch['responses']
        response_length = responses.size(-1)
        attention_mask = data.batch['attention_mask']
        response_mask = attention_mask[:, -response_length:]
        advantages, returns = core_algos.compute_reinforce_plus_plus_outcome_advantage(
            token_level_rewards=token_level_rewards, eos_mask=response_mask, gamma=gamma)
        data.batch['advantages'] = advantages
        data.batch['returns'] = returns
    else:
        raise NotImplementedError
    return data

```

3)line 178 vpreds = self._forward_micro_batch(data)(verl/workers/critic/dp_critic.py)

```python
    def update_critic(self, data: DataProto):
        # make sure we are in training mode
        self.critic_module.train()
        metrics = {}

        select_keys = ['input_ids', 'responses', 'attention_mask', 'position_ids', 'values', 'returns']
        batch = data.select(batch_keys=select_keys).batch
        # Split to make minibatch iterator for updating the actor
        # See PPO paper for details. https://arxiv.org/abs/1707.06347
        dataloader = batch.split(self.config.ppo_mini_batch_size)

        for batch_idx, data in enumerate(dataloader):
            # split batch into micro_batches
            mini_batch = data
            if self.config.use_dynamic_bsz:
                max_token_len = self.config.ppo_max_token_len_per_gpu * self.ulysses_sequence_parallel_size
                micro_batches, _ = rearrange_micro_batches(batch=mini_batch, max_token_len=max_token_len)
            else:
                micro_batches = mini_batch.split(self.config.ppo_micro_batch_size_per_gpu)
                self.gradient_accumulation = self.config.ppo_mini_batch_size // self.config.ppo_micro_batch_size_per_gpu

            self.critic_optimizer.zero_grad()

            for data in micro_batches:
                data = data.cuda()  # critic device is cpu when using offload
                input_ids = data['input_ids']
                responses = data['responses']
                attention_mask = data['attention_mask']
                position_ids = data['position_ids']
                values = data['values']
                returns = data['returns']
                response_length = responses.size(1)

                eos_mask = attention_mask[:, -response_length - 1:-1]

                vpreds = self._forward_micro_batch(data)
```

4)_forward_micro_batch(data)(verl/workers/critic/dp_critic.py)

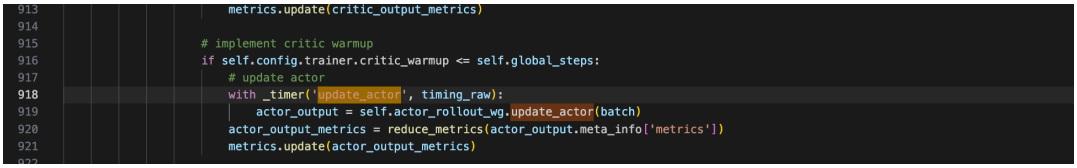


```python
    def _forward_micro_batch(self, micro_batch):
        with torch.autocast(device_type='cuda', dtype=torch.bfloat16):
            input_ids = micro_batch['input_ids']
            batch, seqlen = input_ids.shape
            attention_mask = micro_batch['attention_mask']
            position_ids = micro_batch['position_ids']

            if self.use_remove_padding:
                input_ids_rmpad, indices, *_ = unpad_input(input_ids.unsqueeze(-1),
                                                           attention_mask)  # input_ids_rmpad (total_nnz, ...)
                input_ids_rmpad = input_ids_rmpad.transpose(0, 1)  # (1, total_nnz)

                # unpad the position_ids to align the rotary
                position_ids_rmpad = index_first_axis(rearrange(position_ids.unsqueeze(-1), "b s ... -> (b s) ..."),
                                                      indices).transpose(0, 1)

                # pad and slice the inputs if sp > 1
                if self.ulysses_sequence_parallel_size > 1:
                    input_ids_rmpad, position_ids_rmpad, pad_size = ulysses_pad_and_slice_inputs(input_ids_rmpad, \
                                                                                                 position_ids_rmpad, \
                                                                                                 sp_size=self.ulysses_sequence_parallel_size)

                # only pass input_ids and position_ids to enable flash_attn_varlen
                output = self.critic_module(input_ids=input_ids_rmpad,
                                            attention_mask=None,
                                            position_ids=position_ids_rmpad,
                                            use_cache=False)  # prevent model thinks we are generating
                values_rmpad = output.logits
                values_rmpad = values_rmpad.squeeze(0)  # (total_nnz)

                # gather output if sp > 1
                if self.ulysses_sequence_parallel_size > 1:
                    values_rmpad = gather_outpus_and_unpad(values_rmpad,
                                                           gather_dim=0,
                                                           unpad_dim=0,
                                                           padding_size=pad_size)

                # pad it back
                values = pad_input(values_rmpad, indices=indices, batch=batch, seqlen=seqlen).squeeze(-1)
                values = values[:, -response_length - 1:-1]
            else:
                output = self.critic_module(input_ids=input_ids,
                                            attention_mask=attention_mask,
```

就是对模型走一次forwad计算得到logits,

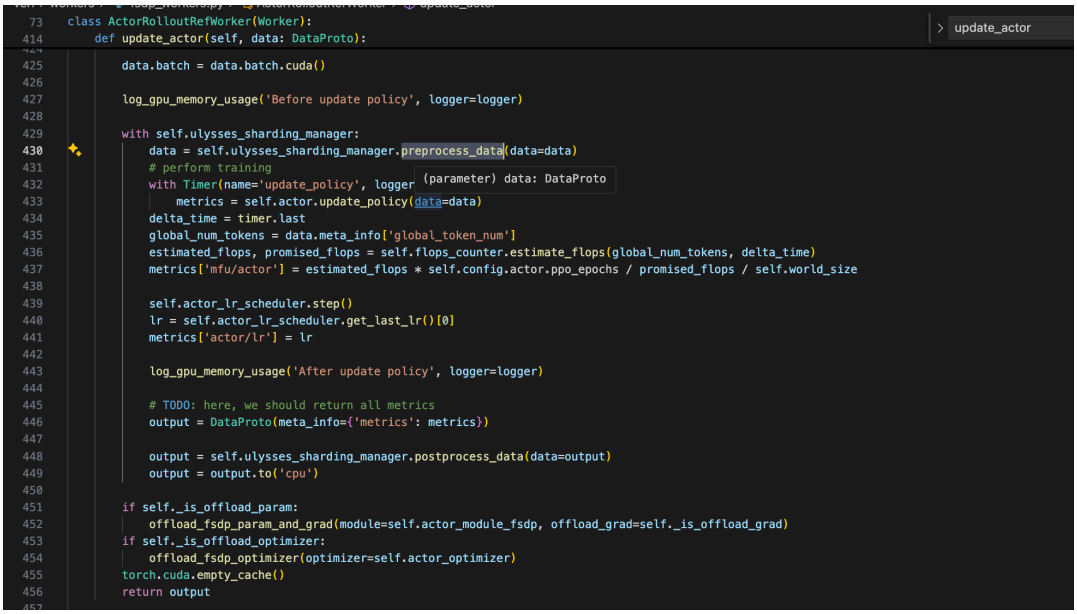
2 DataParallelPPOActor::update_actor

verl/trainer/ppo/ray_trainer.py, line 919

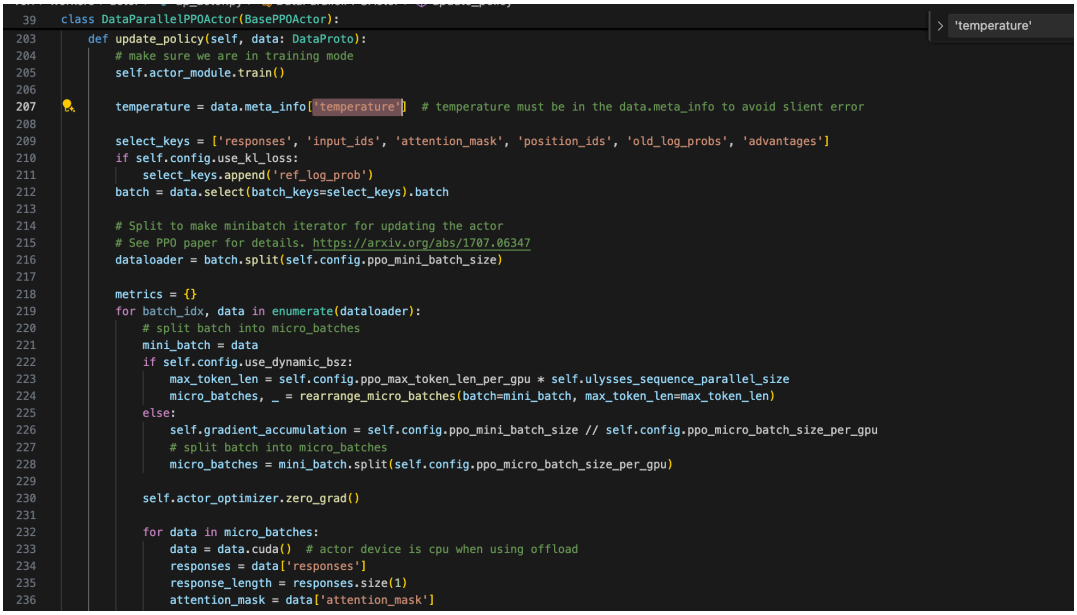


```python
                    metrics.update(critic_output_metrics)

                # implement critic warmup
                if self.config.trainer.critic_warmup <= self.global_steps:
                    # update actor
                    with _timer('update_actor', timing_raw):
                        actor_output = self.actor_rollout_wg.update_actor(batch)
                    actor_output_metrics = reduce_metrics(actor_output.meta_info['metrics'])
                    metrics.update(actor_output_metrics)
```

1)DataParallelPPOActor::update_policy

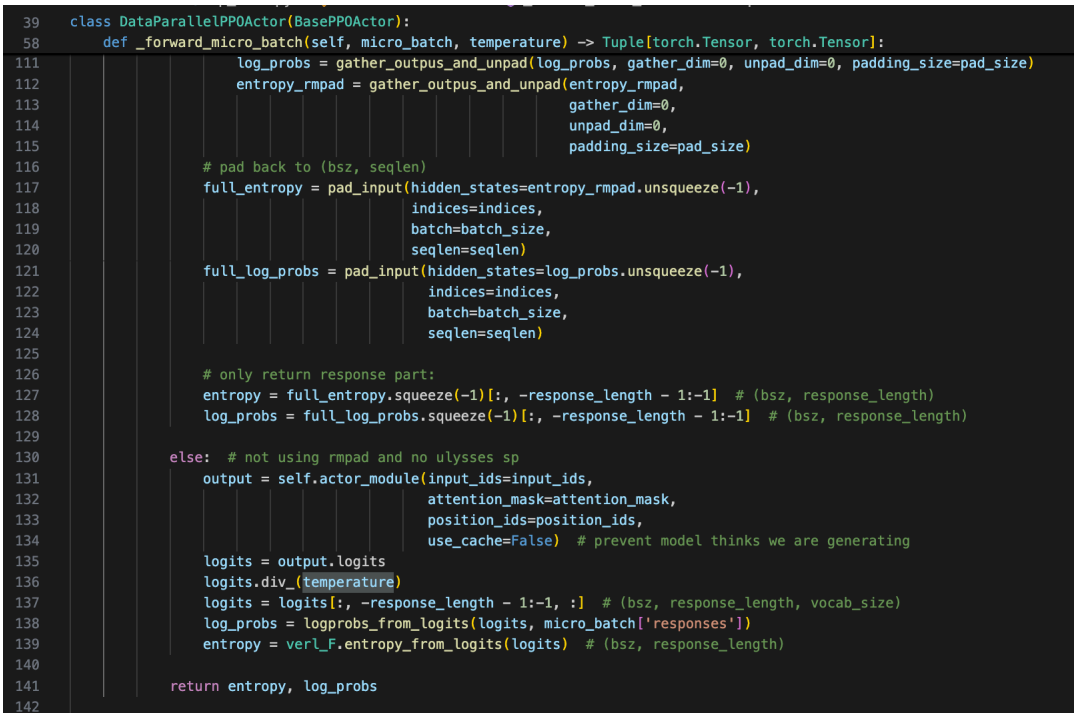


2)DataParallelPPOActor::update_policy (verl/workers/actor/dp_actor.py)



l)注意line 209的old_log_probs,其计算是在(verl/trainer/ppo/ray_trainer.py)

II)回到DataParallelPPOActor::update_policy (verl/workers/actor/dp_actor.py)和 DataParallelPPOActor::forward_micro_batch, 这里的计算得好好学学



```
class DataParallelPPOActor(BasePPOActor):
    def _forward_micro_batch(self, micro_batch, temperature) -> Tuple[torch.Tensor, torch.Tensor]:
                    log_probs = gather_outpus_and_unpad(log_probs, gather_dim=0, unpad_dim=0, padding_size=pad_size)
                    entropy_rmpad = gather_outpus_and_unpad(entropy_rmpad,
                                                            gather_dim=0,
                                                            unpad_dim=0,
                                                            padding_size=pad_size)
                # pad back to (bsz, seqlen)
                full_entropy = pad_input(hidden_states=entropy_rmpad.unsqueeze(-1),
                                         indices=indices,
                                         batch=batch_size,
                                         seqlen=seqlen)
                full_log_probs = pad_input(hidden_states=log_probs.unsqueeze(-1),
                                           indices=indices,
                                           batch=batch_size,
                                           seqlen=seqlen)

                # only return response part:
                entropy = full_entropy.squeeze(-1)[:, -response_length - 1:-1]  # (bsz, response_length)
                log_probs = full_log_probs.squeeze(-1)[:, -response_length - 1:-1]  # (bsz, response_length)

            else:  # not using rmpad and no ulysses sp
                output = self.actor_module(input_ids=input_ids,
                                           attention_mask=attention_mask,
                                           position_ids=position_ids,
                                           use_cache=False)  # prevent model thinks we are generating
                logits = output.logits
                logits.div_(temperature)
                logits = logits[:, -response_length - 1:-1, :]  # (bsz, response_length, vocab_size)
                log_probs = logprobs_from_logits(logits, micro_batch['responses'])
                entropy = verl_F.entropy_from_logits(logits)  # (bsz, response_length)

            return entropy, log_probs
```

3

==========03/05==========

1 一些ray的用法 verl/single_controller/ray/base.py，值得学习

2 明天的任务，看ResourcePool和RayWorkerGroup