PR #6044 完整报告

verl-project/verl

[fully_async, reward] feat: enable GenRM/DisRM support in fully async training

合并时间：2026-04-27 01:55

原文链接：http://prhub.com.cn/verl-project/verl/pull/6044

执行摘要

- 一句话：在 fully async 训练中启用独立 GenRM/DisRM 奖励模型
- 推荐动作：本 PR 解除了 fully async 模式中长期存在的 GPU 奖励模型限制，对需要大模型评判的场景至关重要。值得深入阅读的设计决策包括：如何将 RM 管理从训练器委托给 Rollouter、为什么强制 standalone 模式、以及遗留的 use_trainer_do_validate 冲突问题。建议合入前先确认 rm_resource_pool 硬编码问题是否已有短期修复计划，若急需使用可先合并但需注意资源冲突风险。

功能与动机

用户希望在与同步管道同样的基础设施支持下，在完全异步 GRPO 中使用生成式 / 判别式奖励模型（GenRM/DisRM）评估推理质量。Issue #5949 指出 FullyAsyncRollouter 硬编码 self.use_rm = False，即使底层奖励循环基础设施已存在，也无法启用 GPU 奖励模型。本 PR 旨在解除这一限制，使用户能够独立部署裁判模型。

实现拆解

1. 修改 FullyAsyncRollouter 初始化（fully_async_rollouter.py）：从固定 False 改为 need_reward_model(config) 动态决定 use_rm，并增加断言要求开启 enable_resource_pool=True（colocate 模式因异步 Rollout 永不暂停而不支持）。将原本同步的 _init_reward_loop 重构为异步 _create_reward_loop_manager，通过 run_in_executor 创建 RewardLoopManager，但 rm_resource_pool 参数目前硬编码为 None。同时覆盖 _create_reward_model_class 为空方法。
2. 调整 FullyAsyncTrainer（fully_async_trainer.py）：新增 _create_reward_model_class 空实现；在 _init_models 中移除 if self.use_rm: 分支（RM 由 Rollouter 管理）。_init_reward_loop 改为异步，且仅在 use_trainer_do_validate=True 时创建 RewardLoopManager。
3. 重新分配资源池（separation/utils.py）：将 Role.RewardModel 从训练池 (training_roles) 中移除，不再与 Actor、Critic 等共享 trainer_pool；当 Role.RewardModel 在 roles 列表中时，新增独立的 assert 验证其 GPU 资源配置。
4. 入口层保护性校验（fully_async_main.py）：显式检查若启用了 GenRM 且 use_trainer_do_validate=True，则直接抛出 NotImplementedError，避免运行时 Ray actor 名称冲突。

5. 测试配套：新增 test_async_genrm_config_on_cpu.py 不依赖 GPU 的配置验证单元测试；新增 run_fully_async_policy_genrm.sh 端到端回归测试脚本（3 张 H100，Actor 0.5B、Judge 3B，GRPO 算法，约 3200 rollout steps 验证收敛）；添加 __init__.py 使测试目录可导入。

关键文件：

- verl/experimental/fully_async_policy/fully_async_rollouter.py（模块 异步策略；类别 source；类型 core-logic；符号 _create_reward_loop_manager, _create_reward_model_class）：核心变更文件：将 use_rm 从硬编码 False 改为动态配置读取，增加 standalone 模式断言，添加异步 _create_reward_loop_manager 和空 _create_reward_model_class，是整个功能的主入口。
- verl/experimental/fully_async_policy/fully_async_trainer.py（模块 异步策略；类别 source；类型 core-logic；符号 _create_reward_model_class）：训练器端修改：避开 RM worker group 创建，将 RM 管理完全交予 Rollouter；同时修改 _init_reward_loop 仅在 use_trainer_do_validate 时创建，防止重复初始化。
- tests/experimental/fully_async_policy/test_async_genrm_config_on_cpu.py（模块 单元测试；类别 test；类型 test-coverage；符号 _make_config, TestNeedRewardModel, test_rm_disabled, test_rm_enabled）：新增的单元测试文件，验证配置解析和断言逻辑，确保不依赖 GPU 也能测试配置正确性。
- tests/special_e2e/run_fully_async_policy_genrm.sh（模块 集成测试；类别 test；类型 test-coverage）：端到端回归测试脚本，演示 3 GPU 上运行 GenRM（Qwen2.5 0.5B + 3B）的基本用法和参数配置。
- verl/experimental/separation/utils.py（模块 分离训练；类别 source；类型 core-logic；符号 create_resource_pool_manager）：资源池管理调整：将 RewardModel 从训练池中移除并独立验证 GPU 资源，确保 GenRM 获得专用 GPU。
- verl/experimental/fully_async_policy/fully_async_main.py（模块 异步策略；类别 source；类型 entrypoint）：入口脚本添加保护性检查，阻止 use_trainer_do_validate 与 GenRM 同时启用，避免 Ray actor 名称冲突。

关键符号：FullyAsyncRollouter.init, FullyAsyncRollouter._create_reward_loop_manager, FullyAsyncRollouter._create_reward_model_class, FullyAsyncTrainer._create_reward_model_class, FullyAsyncTrainer._init_models, create_resource_pool_manager, need_reward_model, FullyAsyncTrainer._init_reward_loop

关键源码片段

verl/experimental/fully_async_policy/fully_async_rollouter.py

核心变更文件：将 use_rm 从硬编码 False 改为动态配置读取，增加 standalone 模式断言，添加异步 _create_reward_loop_manager 和空 _create_reward_model_class，是整个功能的主入口。

```
# fully_async_rollouter.py 核心变更：动态 use_rm 与异步 RewardLoopManager 创建

from verl.trainer.ppo.utils import Role, WorkerType, need_reward_model
```

```
class FullyAsyncRollouter(SeparateRayPPOTrainer):
    def __init__(self, config, tokenizer, role_worker_mapping, resource_pool_manager, ...):
        # ... 之前的初始化代码 ...
        # 原先: self.use_rm = False
        # 现在: 从配置动态决定是否启用 reward model
        self.use_rm = need_reward_model(self.config)
        if self.use_rm:
            # 异步模式不支持 colocate, 因为 rollout 永不暂停
            assert self.config.reward.reward_model.enable_resource_pool, (
                "GenRM/DisRM in fully async mode requires standalone mode (enable_resource_pool=
                True). "
                "Colocate mode is not supported because async rollout never pauses."
            )
        # ... 其他初始化 ...

    async def init_workers(self):
        # ... 原有初始化 ...
        self._init_async_objects()
        self._create_worker_classes()
        await self._create_reward_loop_manager() # 替换同步 _init_reward_loop
        await self._init_async_rollout_manager()

    async def _create_reward_loop_manager(self):
        """异步创建 RewardLoopManager, 确保 Rollouter 拥有奖励模型的 Server。
        TODO: rm_resource_pool 应通过资源池获取, 当前硬编码为 None 有风险。
        """
        from verl.experimental.reward_loop import RewardLoopManager
        loop = asyncio.get_running_loop()
        # 通过 run_in_executor 避免 RewardLoopManager 内部 asyncio.run() 阻塞事件循环
        self.reward_loop_manager = await loop.run_in_executor(
            None,
            lambda: RewardLoopManager(config=self.config, rm_resource_pool=None),
        )

    def _create_reward_model_class(self):
        # 异步模式下 RM 由 RewardLoopManager 管理, 无需创建 worker group
        pass
```

verl/experimental/fully_async_policy/fully_async_trainer.py

训练器端修改: 避开 RM worker group 创建, 将 RM 管理完全交予 Rollouter; 同时修改 _init_reward_loop 仅在 use_trainer_do_validate 时创建, 防止重复初始化。

```
# fully_async_trainer.py 核心变更: 空 _create_reward_model_class 和移除 RM worker group
初始化

class FullyAsyncTrainer:
    def _create_reward_model_class(self):
        # 在异步模式中, 奖励模型由 Rollouter 端的 RewardLoopManager 管理,
        # 训练器不需要也创建 RM worker group, 因此此方法为空。
```

```
        pass

    def _init_models(self):
        # ... 原有初始化 ...
        # 注意：原先对 use_rm 和 rm_wg 的初始化已被移除，
        # 因为奖励模型的 worker group 不在训练器中创建了。
        # - 之前：if self.use_rm:
        # - self.rm_wg = self.all_wg[str(Role.RewardModel)]
        # - self.rm_wg.init_model()
        self.actor_wg = self.all_wg[str(self.train_role)]
        self.actor_wg.init_model()
        # ... 其他初始化 ...

    async def _init_reward_loop(self):
        # 只有 use_trainer_do_validate=True 时，训练器才需要创建自己的 RewardLoopManager
        #（做验证生成时的奖励计算）；否则跳过，由 Rollouter 统一管理。
        if self.config.async_training.use_trainer_do_validate:
            print("[FullyAsyncTrainer] Init reward loop")
            super()._init_reward_loop()
```

tests/experimental/fully_async_policy/test_async_genrm_config_on_cpu.py

新增的单元测试文件，验证配置解析和断言逻辑，确保不依赖 GPU 也能测试配置正确性。

```
# test_async_genrm_config_on_cpu.py 关键测试逻辑

from omegaconf import OmegaConf
from verl.trainer.ppo.utils import need_reward_model

def _make_config(reward_model_enable=False, enable_resource_pool=False):
    """构建最小配置，用于测试 reward model 设置"""
    return OmegaConf.create({
        "reward": {
            "reward_model": {
                "enable": reward_model_enable,
                "enable_resource_pool": enable_resource_pool,
                "n_gpus_per_node": 2,
                "nnodes": 1,
                "model_path": "dummy/model",
                "rollout": {
                    "name": "vllm",
                    "tensor_model_parallel_size": 1,
                    "gpu_memory_utilization": 0.5,
                    "skip_tokenizer_init": False,
                },
            },
            "custom_reward_function": {"path": None, "name": None},
        },
    })

class TestAsyncRollouterRMAssert(unittest.TestCase):
```

```
    """验证 FullyAsyncRollouter 中的配置断言逻辑（不实例化完整类）"""

    @staticmethod
    def _validate_async_rm_config(config):
        """复现 Rollouter.__init__ 中的 RM 检查逻辑"""
        use_rm = need_reward_model(config)
        if use_rm:
            # 异步模式必须使用 standalone resource pool
            assert config.reward.reward_model.enable_resource_pool, (
                "GenRM/DisRM in fully async mode requires standalone mode (enable_resource_pool=
                True)."
            )
        return use_rm

    def test_rm_enabled_standalone_passes(self):
        # 正确配置：RM 开启 + standalone 资源池
        config = _make_config(reward_model_enable=True, enable_resource_pool=True)
        assert self._validate_async_rm_config(config) is True

    def test_rm_enabled_colocate_fails(self):
        # 错误配置：RM 开启但未使用 standalone，应触发断言
        config = _make_config(reward_model_enable=True, enable_resource_pool=False)
        with pytest.raises(AssertionError, match="standalone mode"):
            self._validate_async_rm_config(config)
```

评论区精华

- 资源池硬编码风险：gemini-code-assist[bot] 指出 fully_async_rollouter.py 中 RewardLoopManager 的 rm_resource_pool 硬编码为 None，绕过了 separation/utils.py 中定义的资源分配逻辑，可能导致资源冲突或 OOM。此问题在合并时未解决，仅作为已知风险记录。
- use_trainer_do_validate 兼容性争议：wuxibin89 认为 GenRM 使用独立资源池，与 use_trainer_do_validate 应兼容；但 xiefan46 通过实验发现两者同时使用会导致 Ray actor 名称冲突（vllm_server、reward_loop_worker 等命名重复），并已提交探索性 PR #6151。yyDing1 确认这是一个根本性问题，并建议通过重命名推理服务器 actor（如 rollouter、trainer_rollouter、reward_model）来全面解决。
  - rm_resource_pool 硬编码为 None 绕过资源分配逻辑 (design): 作为已知问题未在本次 PR 中修复，作者 xiefan46 未正面回应；后续需通过 #6151 或其他 PR 解决。
  - use_trainer_do_validate 与 GenRM 冲突 (design): 当前 PR 将冲突检测提前到入口层，直接抛出 NotImplementedError，避免运行时错误。长期需要重构 actor 命名方案。

风险与影响

- 风险：
  - 资源分配绕过：RewardLoopManager 的 rm_resource_pool 目前硬编码 None，未使用 resource_pool_manager.get_resource_pool(Role.RewardModel)。这可能导致在整卡

训练环境中奖励模型与训练进程资源冲突，引发 OOM 或 CUDA 错误。

- 配置兼容性限制：要求必须启用 *enable_resource_pool=True*，任何使用 *colocate* 模式的现有配置将自动报错。虽然这在异步模式下是合理限制，但对突然升级的用户形成 breakage。
- use_trainer_do_validate 禁用：当启用 GenRM 时，若用户同时设置 use_trainer_do_validate=True，程序将直接抛出 NotImplementedError，阻止训练启动。这虽然避免了运行时报错，但也意味着验证生成完全禁用，可能影响实验监控。
- 异步初始化性能：RewardLoopManager.__init__ 内部使用 asyncio.run()，迫使 Rollouter 通过 run_in_executor 在单独线程中同步调用，可能增加启动延迟。后续应提供真正的异步初始化接口。

- 影响：
  - 用户影响：fully async 用户现在可以通过配置 reward.reward_model.enable=True 和 reward.reward_model.enable_resource_pool=True 启用 GenRM/DisRM，将奖励模型部署在独立 GPU 上。但仍有局限（不能与 use_trainer_do_validate 共用）。
  - 系统影响：需要至少 3 张 GPU（Rollout、Train、RM 各一）。资源池管理增加了对 Role.RewardModel 的独立配置校验。
  - 团队影响：实现了异步管道中奖励模型的基本支持，但遗留了资源硬编码和 actor 命名等未解决技术债务，预计后续需通过更彻底的架构调整（如 #6151）来解决。
  - 风险标记：资源池硬编码，Ray actor 名称冲突，配置限制（standalone 必需），use_trainer_do_validate 禁用

关联脉络

- PR #6151 [fully_async] draft: explore use_trainer_do_validate + GenRM support: 由作者 xiefan46 在讨论中提及，作为后续解决 actor 名称冲突的草案，与当前 PR 的冲突检测直接相关。