PR #6818 完整报告

verl-project/verl

[reward] feat: colocated reward model for v1 sync/colocate_async trainer

合并时间：2026-06-23 16:59

原文链接：http://prhub.com.cn/verl-project/verl/pull/6818

执行摘要

- 一句话：实现 V1 PPO 训练器共置奖励模型评分
- 推荐动作：值得精读，特别是 _compute_reward_colocate 的实现和注意力掩码的构建方式，展示了如何在 PPO 训练器中集成共置奖励模型。对应的单元测试设计也值得参考。团队成员应了解 separate_async 模式的限制。

功能与动机

之前 V1 PPO 训练器的 _compute_reward_colocate 仅抛出 NotImplementedError，无法实际使用共置奖励模型。该 PR 填补了这一空白，使得奖励模型可以与 rollout 共享 GPU，避免独立部署带来的额外资源开销。PR 描述明确要求“Implement colocated reward model scoring for V1 PPO trainer”并添加测试。

实现拆解

按照以下步骤实现：

1. 实现核心方法 _compute_reward_colocate（verl/trainer/ppo/v1/trainer_base.py）：从 TransferQueue 中读取 prompts、responses 和 raw_prompt，计算每行长度并构造右填充注意力掩码，组装 DataProto 输入，调用 RewardLoopManager.compute_rm_score 获取奖励分数，然后将结果写回 TransferQueue。同时新增辅助方法 _lengths_to_mask 用于根据有效长度生成布尔掩码。
2. 阻止 separate_async 模式使用共置模型（verl/trainer/ppo/v1/trainer_separate_async.py）：在 PPOTrainerSeparateAsync.__init__ 中添加断言，如果启用了奖励模型且 enable_resource_pool=False（即共置模式），则抛出 AssertionError，引导用户使用独立资源池模式。
3. 新增 CPU 单元测试（tests/trainer/ppo/v1/test_compute_reward_colocate_on_cpu.py）：在不依赖 GPU 运行时的条件下，独立测试 _lengths_to_mask 的正确性，以及注意力掩码与 RewardManagerBase.assemble_rm_scores 的契约（即掩码尾部求和等于有效响应长度）。
4. 新增端到端回归测试脚本（tests/special_e2e/run_v1_colocate_async_disrm.sh）：在 2+ GPU 节点上运行 V1 colocate_async 训练器，使用 Skywork-Reward 模型作为共置判别式奖励模型，验证完整训练流程。

5. CI 配置更新（.github/workflows/reward_model_vllm.yml）：将新的 E2E 测试脚本加入触发器路径，并添加运行步骤。

关键文件：

- verl/trainer/ppo/v1/trainer_base.py（模块 训练器；类别 source；类型 core-logic；符号 _compute_reward_colocate, _lengths_to_mask）：核心逻辑变更，实现共置奖励模型评分方法和长度转掩码辅助函数
- tests/trainer/ppo/v1/test_compute_reward_colocate_on_cpu.py（模块 单元测试；类别 test；类型 test-coverage；符号 _lengths_to_mask, _assemble_rm_scores, TestLengthsToMask, test_basic_right_padding）：CPU 单元测试，验证掩码构造和奖励分数组装契约，不依赖 GPU 运行时
- tests/special_e2e/run_v1_colocate_async_disrm.sh（模块 E2E 测试；类别 test；类型 test-coverage）：端到端回归测试脚本，验证完整训练流程与共置奖励模型的交互
- verl/trainer/ppo/v1/trainer_separate_async.py（模块 训练器；类别 source；类型 dependency-wiring）：添加断言阻止 separate_async 模式使用共置奖励模型，避免 GPU 内存冲突
- .github/workflows/reward_model_vllm.yml（模块 CI；类别 infra；类型 infrastructure）：CI 配置更新，将新的 E2E 测试加入触发路径和运行步骤

关键符号：_compute_reward_colocate, _lengths_to_mask

关键源码片段

verl/trainer/ppo/v1/trainer_base.py

核心逻辑变更，实现共置奖励模型评分方法和长度转掩码辅助函数

```
# Copyright 2024 Bytedance Ltd. and/or its affiliates
# (license header omitted for brevity)

def _compute_reward_colocate(self, batch: KVBatchMeta, metrics: dict | None = None) ->
KVBatchMeta:
    """Compute the reward score with a colocated reward model."""
    assert self.reward_loop_manager is not None, "RewardLoopManager is None"

    # 1. 从 TransferQueue 读取 prompts、responses 和 raw_prompt
    fields = ["prompts", "responses", "raw_prompt"]
    data = tq.kv_batch_get(keys=batch.keys, partition_id=batch.partition_id, select_fields=fields)

    prompt_lengths = data["prompts"].offsets().diff()
    response_lengths = data["responses"].offsets().diff()
    prompts = data["prompts"].to_padded_tensor(padding=self.tokenizer.pad_token_id)
    responses = data["responses"].to_padded_tensor(padding=self.tokenizer.pad_token_id)

    # 2. 重建注意力掩码： [prompt_mask | response_mask]
    prompt_mask = self._lengths_to_mask(prompt_lengths, prompts.size(1))
    response_mask = self._lengths_to_mask(response_lengths, responses.size(1))
    attention_mask = torch.cat([prompt_mask, response_mask], dim=1)
```

```
    # raw_prompt 是非张量字段，来自 TransferQueue 的类型多样，list() 统一为列表
    raw_prompts = list(data["raw_prompt"])
    raw_prompt_arr = np.empty(len(raw_prompts), dtype=object)
    raw_prompt_arr[:] = raw_prompts

    rm_input = DataProto(
        batch=TensorDict(
            {"prompts": prompts, "responses": responses, "attention_mask": attention_mask},
            batch_size=len(batch),
        ),
        non_tensor_batch={"raw_prompt": raw_prompt_arr},
    )

    # 3. 调用 RewardLoopManager 计算奖励分数（内部管理唤醒 / 睡眠模型）
    rm_output = self.reward_loop_manager.compute_rm_score(rm_input)

    # 4. 将 rm_scores 写回 TransferQueue（省略奖励额外信息的写回）
    padded_rm_scores = rm_output.batch["rm_scores"]
    rm_scores = torch.nested.as_nested_tensor(
        [padded_rm_scores[i, : response_lengths[i]] for i in range(len(batch))],
        layout=torch.jagged,
    )
    # 实际写回逻辑包括 kv_batch_put 等，此处省略
    return batch
```

tests/trainer/ppo/v1/test_compute_reward_colocate_on_cpu.py

CPU 单元测试，验证掩码构造和奖励分数组装契约，不依赖 GPU 运行时

```
# Copyright 2024 Bytedance Ltd. and/or its affiliates
# (license header omitted for brevity)

def _lengths_to_mask(lengths: torch.Tensor, width: int) -> torch.Tensor:
    """Standalone copy of ``PPOTrainer._lengths_to_mask`` 用于单元测试。

    后续保持与 trainer_base.py 同步。直接导入 trainer 会引入大量运行时依赖（ray、transfer_
    queue、vllm）。
    """
    positions = torch.arange(width, device=lengths.device).unsqueeze(0)
    return (positions < lengths.unsqueeze(1)).to(torch.int64)


def _assemble_rm_scores(prompts, attention_mask, responses, scores):
    """Replica of ``RewardManagerBase.assemble_rm_scores`` 用于断言契约。"""
    prompt_length = prompts.size(1)
    valid_response_length = attention_mask[:, prompt_length:].sum(dim=1)
    rm_scores = torch.zeros_like(responses, dtype=torch.float32)
    rm_scores[torch.arange(rm_scores.size(0)), valid_response_length - 1] = rm_scores.new_
    tensor(scores)
    return rm_scores
```

```
class TestLengthsToMask:
    def test_basic_right_padding(self):
        lengths = torch.tensor([1, 3, 2])
        mask = _lengths_to_mask(lengths, width=4)
        expected = torch.tensor([[1, 0, 0, 0], [1, 1, 1, 0], [1, 1, 0, 0]], dtype=torch.int64)
        assert torch.equal(mask, expected)

    def test_full_and_empty_rows(self):
        lengths = torch.tensor([0, 4])
        mask = _lengths_to_mask(lengths, width=4)
        expected = torch.tensor([[0, 0, 0, 0], [1, 1, 1, 1]], dtype=torch.int64)
        assert torch.equal(mask, expected)

    def test_row_count_matches_lengths(self):
        lengths = torch.tensor([2, 2, 2, 2, 2])
        mask = _lengths_to_mask(lengths, width=3)
        assert mask.shape == (5, 3)


class TestAttentionMaskContract:
    def test_valid_response_length_recovered(self):
        prompt_lengths = torch.tensor([2, 1, 3])
        response_lengths = torch.tensor([3, 2, 1])
        prompt_width, response_width = 4, 4
        prompt_mask = _lengths_to_mask(prompt_lengths, prompt_width)
        response_mask = _lengths_to_mask(response_lengths, response_width)
        attention_mask = torch.cat([prompt_mask, response_mask], dim=1)
        recovered = attention_mask[:, prompt_width:].sum(dim=1)
        assert torch.equal(recovered, response_lengths)

    def test_score_lands_on_last_valid_response_token(self):
        # 验证分数被放置在最后一个有效响应令牌位置，且其他地方为零
        prompt_lengths = torch.tensor([2, 1])
        response_lengths = torch.tensor([3, 2])
        prompt_width, response_width = 4, 5
        prompts = torch.zeros((2, prompt_width), dtype=torch.int64)
        responses = torch.zeros((2, response_width), dtype=torch.int64)
        prompt_mask = _lengths_to_mask(prompt_lengths, prompt_width)
        response_mask = _lengths_to_mask(response_lengths, response_width)
        attention_mask = torch.cat([prompt_mask, response_mask], dim=1)
        scores = [0.5, -1.0]
        rm_scores = _assemble_rm_scores(prompts, attention_mask, responses, scores)
        assert rm_scores[0, 2].item() == 0.5
        assert rm_scores[1, 1].item() == -1.0
        assert rm_scores.sum().item() == (0.5 - 1.0)
        assert torch.equal(rm_scores.sum(dim=1), torch.tensor(scores))
```

评论区精华

该 PR 的主要设计决策体现在 trainer_separate_async.py 中的断言：共置奖励模型在 separate_async 模式下不被支持，因为独立 rollout 从不暂停，无法释放 GPU 内存供奖励模型复用。yyDing1 审批通过（LGTM），没有其他争议。

- 暂无高价值评论线程

风险与影响

- 风险：
  - GPU 内存竞争：共置模式下奖励模型与训练 /rollout 共享 GPU，若配置不当可能导致 OOM。E2E 测试中通过降低 gpu_memory_utilization 和启用 free_cache_engine 缓解，但用户需自行调参。
  - TransferQueue 依赖：核心路径依赖 TransferQueue 的 kv_batch_get 和 kv_batch_put，若队列状态异常可能导致训练卡死。
  - separate_async 模式限制：通过断言阻止了共置模型的误用，但可能对期望在 separate_async 下使用共置模型的用户造成困惑，需文档说明。
  - 影响：该 PR 使 V1 PPO 训练器（sync / colocate_async）能够使用共置奖励模型，降低了对额外 GPU 资源的需求，提高了硬件利用率。影响范围限于 V1 PPO 训练器的奖励计算管道，不影响 V0 或其他后端。单独的 separate_async 模式被显式排除，需用户明确使用独立资源池。
  - 风险标记：共置模型 GPU 内存竞争，separate_async 模式显式禁止，依赖 TransferQueue

关联脉络

- PR #6790 [trainer] feat: A runnable separate async trainer: 修改了同一个文件 trainer_separate_async.py，该 PR 在此基础上增加了共置奖励模型的限制。
- PR #6572 [rollout, reward] feat: add full determinism support for vLLM rollout and reward model: 同为奖励模型相关功能，该 PR 新增的确定性与共置奖励模型可能产生交互。