PR #6242 完整报告

verl-project/verl

[reward] refactor: refactor RM score assembly in reward loop

合并时间：2026-05-06 11:29

原文链接：http://prhub.com.cn/verl-project/verl/pull/6242

执行摘要

- 一句话：提取奖励分数组装为可覆写钩子
- 推荐动作：建议精读，这是一个典型的重构样例：从函数中提取职责，为扩展开放接口。值得关注的决策是使用 classmethod 而非实例方法，因为组装逻辑通常不需要实例状态。

功能与动机

如 PR body 所述，目的是将奖励分数组装逻辑提取为可覆写的方法，以便下游包能够处理不同的奖励组装方式，如处理图像、视频等场景。

实现拆解

1. 新增 resolve_reward_manager_cls 函数：在 verl/trainer/ppo/reward.py 中新增该函数，从配置中解析出奖励管理器类而不实例化它。该函数处理 register 和 importlib 两种来源，并处理未知来源的异常。原有的 load_reward_manager 函数中对应的内联代码被替换为调用此函数。
2. 新增 assemble_rm_scores 类方法：在 verl/experimental/reward_loop/reward_manager/base.py 的 RewardManagerBase 中新增 assemble_rm_scores 类方法。该方法的默认实现根据 prompts、responses 和 attention_mask 计算每个样本的有效响应长度，并将奖励分数放置在最后一个有效 token 位置。此方法可被子类覆写以支持自定义组装逻辑。
3. 更新 RewardLoopManager：在 verl/experimental/reward_loop/reward_loop.py 中，RewardLoopManager 新增 self.reward_manager_cls 属性，通过 resolve_reward_manager_cls 解析类。在 compute_rm_score 方法中，原本内联的分数组装逻辑被替换为 self.reward_manager_cls.assemble_rm_scores(data, scores) 调用，同时移除了对 torch 的直接导入。

关键文件：

- verl/trainer/ppo/reward.py（模块 训练器；类别 source；类型 core-logic；符号 resolve_reward_manager_cls）：新增 resolve_reward_manager_cls 函数，将奖励管理器类解析从 load_reward_manager 中提取，提升了代码可复用性。配置键和异常处理也在此文件改动。
- verl/experimental/reward_loop/reward_manager/base.py（模块 奖励循环；类别 source；类型 core-logic；符号 assemble_rm_scores）：新增 assemble_rm_scores 类方法，定义了默认的奖励分数组装逻辑，支持子类覆写以处理多模态等场景。

- verl/experimental/reward_loop/reward_loop.py（模块 奖励循环；类别 source；类型 dependency-wiring）：RewardLoopManager 使用 resolve_reward_manager_cls 获取类，并在 compute_rm_score 中调用 assemble_rm_scores，替代了内联逻辑；移除了 torch 的直接导入。

关键符号：resolve_reward_manager_cls, assemble_rm_scores

关键源码片段

verl/trainer/ppo/reward.py

新增 resolve_reward_manager_cls 函数，将奖励管理器类解析从 load_reward_manager 中提取，提升了代码可复用性。配置键和异常处理也在此文件改动。

```
# verl/trainer/ppo/reward.py ( 新增函数 )
# 从 config 解析出 RewardManager 类，不执行实例化
# 支持两种来源："register" 和 "importlib"
def resolve_reward_manager_cls(config: DictConfig) -> type[RewardManagerBase]:
    reward_manager_cfg: RewardManagerConfig = config.reward.reward_manager
    if reward_manager_cfg.source == "register":
        from verl.experimental.reward_loop.reward_manager import get_reward_manager_cls
        return get_reward_manager_cls(reward_manager_cfg.name)
    elif reward_manager_cfg.source == "importlib":
        from verl.utils.import_utils import load_extern_object
        module_cfg: ModuleConfig | None = reward_manager_cfg.module
        assert module_cfg is not None and module_cfg.path is not None, (
            f"Module path is required when {reward_manager_cfg.source=}, but got {module_cfg=}"
        )
        return cast(
            "type[RewardManagerBase]",
            load_extern_object(module_path=module_cfg.path, object_name=reward_manager_cfg.
            name),
        )
    else:
        raise ValueError(f"Unknown reward manager source: {reward_manager_cfg.source}")
```

verl/experimental/reward_loop/reward_manager/base.py

新增 assemble_rm_scores 类方法，定义了默认的奖励分数组装逻辑，支持子类覆写以处理多模态等场景。

```
# verl/experimental/reward_loop/reward_manager/base.py (RewardManagerBase 新增方法 )
import torch

@classmethod
def assemble_rm_scores(cls, data: DataProto, scores: list[float]) -> torch.Tensor:
    """将每个样本的 reward scores 组装为 rm_scores 张量。

    默认实现将分数放到每个样本的最后一个有效 token 位置。
    子类可覆写以支持图像、视频等自定义组装逻辑。
    """
```

```
    prompt_length = data.batch["prompts"].size(1)
    # 计算每个样本的有效响应长度（排除 padding）
    valid_response_length = data.batch["attention_mask"][:, prompt_length:].sum(dim=1)
    rm_scores = torch.zeros_like(data.batch["responses"], dtype=torch.float32)
    # 将分数放置到最后一个有效 token 位置
    rm_scores[torch.arange(rm_scores.size(0), device=rm_scores.device), valid_response_length -
    1] = (
        rm_scores.new_tensor(scores)
    )
    return rm_scores
```

评论区精华

无 review 评论。只有 gemini-code-assist[bot] 的总结性评论和 SamitHuang 的批准，无实质性讨论。

- 暂无高价值评论线程

风险与影响

- 风险：风险较低。变更位于奖励循环模块，且已被 reviewer 验证不会破坏现有功能。但需要注意：
  - 如果下游包覆写了 assemble_rm_scores 但未正确处理 attention_mask 或张量设备，可能导致设备不匹配错误。
  - RewardLoopManager 中原来的 torch.tensor(scores, dtype=torch.float32) 现在通过 rm_scores.new_tensor(scores) 创建，确保类型和 device 与 rm_scores 一致，这是一个隐式改进，但若 rm_scores 不在 CPU 上，则 new_tensor 会在该 device 上创建，不会引入风险。
  - 影响：
  - 用户：无直接影响，API 未改变。
  - 系统：提高了代码复用性和可扩展性，reward 循环的分数组装逻辑现在可通过继承定制。
  - 团队：降低了维护成本，插件化设计便于引入新功能。
  - 风险标记：无实质性讨论，改动集中在非核心路径，已获 reviewer 批准

关联脉络

- PR #6184 [veomni] feat: use VeOmni's native return_log_probs path to compute log_probs: 都涉及 reward 模块的改造，且 PR body 提及下游包（如 verl-omni）需要自定义奖励组装，与 VeOmni 集成相关。
- PR #6200 [BREAKING][misc] refactor: migrate Diffusion RL stack to verl-omni: 迁移到 verl-omni 仓库，本 PR 的改动正是为了支持这类下游包的自定义需求。