PR #5951 完整报告

verl-project/verl

[5/n][trainer] feat: flowgrpo trainer

合并时间：2026-04-17 12:11

原文链接：http://prhub.com.cn/verl-project/verl/pull/5951

执行摘要

- 一句话：新增基于 Ray 的 FlowGRPO 扩散模型训练器，支持图像生成强化学习。
- 推荐动作：该 PR 值得精读，特别是 ray_diffusion_trainer.py 中的训练循环设计和 diffusion_algos.py 中的优势计算实现。关注点包括：(1) 扩散模型如何适配 VERL 的 DataProto 和训练框架；(2) 针对时间步的掩码和指标计算与语言模型处理的差异；(3) 审阅中关于优势计算标准差的未决争议，这反映了算法实现与原始论文的权衡。

功能与动机

PR 作者在 body 中说明，这是“让 flowgrpo 训练器可运行的最后一环”，旨在为扩散模型（特别是图像生成任务如 Qwen-Image）提供强化学习训练能力。它延续了 #5297 的工作，并与 @AndyZhou952 合作完成，目标是支持基于扩散模型的 RL 训练。

实现拆解

1. 新增训练器核心模块：在 verl/trainer/diffusion/ 下新增 ray_diffusion_trainer.py，定义了 RayFlowGRPOTrainer 类，负责扩散模型的分布式训练循环、数据加载、日志记录和验证生成。关键函数包括 compute_response_mask（为扩散潜变量生成全 1 掩码）和 compute_advantage（调用扩散专用优势估计器）。
2. 新增训练入口和配置：新增 verl/trainer/main_flowgrpo.py 作为 Hydra 入口点，包含 run_flowgrpo 函数和 TaskRunner 类，负责 Ray 集群初始化和资源池管理。同时更新了扩散训练和 rollout 的配置文件（如 diffusion_rollout.yaml），引入了 extra_configs 字段以容纳算法或模型特定参数。
3. 实现扩散专用算法和指标：在 verl/trainer/diffusion/diffusion_algos.py 中新增 DiffusionAdvantageEstimator.FLOW_GRPO 枚举和 compute_flow_grpo_outcome_advantage 函数，实现了针对扩散模型（每个去噪时间步均为有效优化步）的 GRPO 优势计算逻辑。新增 diffusion_metric_utils.py，包含 compute_data_metrics_diffusion 等函数，用于计算扩散训练特有的指标（如基于时间步的奖励、优势统计）。
4. 提供数据预处理和示例脚本：在 examples/flowgrpo_trainer/ 下新增 OCR 数据集预处理脚本 qwenimage_ocr.py，包含 extract_solution、make_map_fn 等函数，将原始文本数据转换为包含系统提示、负提示和奖励模型信息的训练格式。同时提供了端到端运行脚本和 LoRA 训练示例。

5. 配套测试和现有模块适配：新增了 CPU 单元测试 test_diffusion_core_algos_on_cpu.py 和端到端测试脚本 run_flowgrpo_trainer_diffusers.sh。对现有模块进行了小范围适配，包括：修改 verl/utils/dataset/rl_dataset.py 中的 _build_messages 以支持负提示和多模态数据；更新 verl/workers/engine_workers.py 的导入关系以支持新训练器；调整 examples/flowgrpo_trainer/vllm_omni/pipeline_qwenimage.py 中的 _coalesce_not_none 逻辑。

关键文件：

- verl/trainer/diffusion/ray_diffusion_trainer.py（模块 扩散训练器；类别 source；类型 core-logic；符号 compute_response_mask, compute_advantage, RayFlowGRPOTrainer, init）：新增的 FlowGRPO 训练器核心实现，负责分布式训练循环、数据加载、优势计算和日志记录。
- verl/trainer/main_flowgrpo.py（模块 训练入口；类别 source；类型 entrypoint；符号 main, run_flowgrpo, TaskRunner, init）：FlowGRPO 训练的 Hydra 入口点，负责 Ray 初始化和任务运行器管理。
- verl/trainer/diffusion/diffusion_algos.py（模块 扩散算法；类别 source；类型 core-logic；符号 DiffusionAdvantageEstimator, compute_flow_grpo_outcome_advantage）：扩散专用优势估计算法实现，包含 Flow-GRPO 的核心逻辑。
- examples/flowgrpo_trainer/data_process/qwenimage_ocr.py（模块 示例脚本；类别 source；类型 data-contract；符号 extract_solution, make_map_fn, process_fn）：OCR 数据集预处理示例，展示了如何为扩散 RL 训练准备多模态数据。
- tests/trainer/diffusion/test_diffusion_core_algos_on_cpu.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_flow_grpo_advantage_return, test_compute_policy_loss_flow_grpo）：扩散核心算法的单元测试，验证 Flow-GRPO 优势计算和政策损失的正确性。

关键符号：compute_response_mask, compute_advantage, RayFlowGRPOTrainer, run_flowgrpo, compute_flow_grpo_outcome_advantage, compute_data_metrics_diffusion, extract_solution, _build_messages

关键源码片段

verl/trainer/diffusion/ray_diffusion_trainer.py

新增的 FlowGRPO 训练器核心实现，负责分布式训练循环、数据加载、优势计算和日志记录。

```
def compute_response_mask(data: DataProto):
    """计算扩散潜变量的有效步掩码。

    对于扩散模型，每个去噪时间步都是一个有效的优化步，
    因此返回的掩码是全1的，覆盖所有时间步。
    """
    all_latents = data.batch["all_latents"]
    b, t = all_latents.shape[:2] # 获取批次大小和时间步数
    return torch.ones((b, t), dtype=torch.int32, device=all_latents.device) # 生成全1掩码

def compute_advantage(
```

```
    data: DataProto,
    adv_estimator: str,
    norm_adv_by_std_in_grpo: bool = True,
    global_std: bool = True,
    config: Optional[DiffusionAlgoConfig] = None,
) -> DataProto:
    """为扩散策略优化计算优势估计。

    此函数使用注册的优势估计器（如Flow-GRPO）为扩散模型计算优势估计。
    优势估计用于指导跨去噪时间步的策略优化。
    """
    if "response_mask" not in data.batch.keys():
        data.batch["response_mask"] = compute_response_mask(data) # 确保响应掩码存在

    adv_kwargs = {
        "sample_level_rewards": data.batch["sample_level_rewards"],
        "response_mask": data.batch["response_mask"],
        "config": config,
    }
    if "uid" in data.non_tensor_batch:
        adv_kwargs["index"] = data.non_tensor_batch["uid"] # 添加分组索引
    # 调用注册的优势估计器函数进行计算
    adv_fn = get_adv_estimator_fn(adv_estimator)
    advantages, returns = adv_fn(**adv_kwargs)
    data.batch["advantages"] = advantages
    data.batch["returns"] = returns
    return data
```

verl/trainer/diffusion/diffusion_algos.py

扩散专用优势估计算法实现，包含 Flow-GRPO 的核心逻辑。

```
@register_adv_est(DiffusionAdvantageEstimator.FLOW_GRPO)
def compute_flow_grpo_outcome_advantage(
    sample_level_rewards: torch.Tensor,
    response_mask: torch.Tensor,
    index: np.ndarray,
    epsilon: float = 1e-4,
    norm_adv_by_std_in_grpo: bool = True,
    global_std: bool = True,
    config: Optional[DictConfig] = None,
) -> tuple[torch.Tensor, torch.Tensor]:
    """为GRPO计算优势，仅处理结果奖励（每个响应只有一个标量奖励）。

    注意：如果norm_adv_by_std_in_grpo为True，则优势按标准差缩放，如原始GRPO；
    如果为False，则不缩放，如Dr.GRPO。
    """
    scores = sample_level_rewards
    if scores.ndim == 1:
        scores = scores.unsqueeze(-1)
```

```
    scores = scores.expand_as(response_mask).clone() # 将奖励扩展到所有时间步

    id2score = defaultdict(list)
    id2mean = {}
    id2std = {}

    with torch.no_grad():
        if global_std:
            batch_std = torch.std(scores) # 计算全局标准差 （审阅指出此处可能低估）
        else:
            batch_std = None

        bsz = scores.shape[0]
        for i in range(bsz):
            id2score[index[i]].append(scores[i]) # 按索引分组存储分数
        for idx in id2score:
            if len(id2score[idx]) == 1:
                id2mean[idx] = id2score[idx][0].mean() # 单样本组：均值设为自身分数（已修复）
                if global_std:
                    id2std[idx] = batch_std
                else:
                    id2std[idx] = torch.tensor(1.0)
            elif len(id2score[idx]) > 1:
                scores_tensor = torch.stack(id2score[idx])
                id2mean[idx] = torch.mean(scores_tensor)
                if global_std:
                    id2std[idx] = batch_std
                else:
                    id2std[idx] = torch.std(scores_tensor) # 组内计算标准差
            else:
                raise ValueError(f"no score in prompt index: {idx}")
        for i in range(bsz):
            if norm_adv_by_std_in_grpo:
                scores[i] = (scores[i] - id2mean[index[i]]) / (id2std[index[i]] + epsilon)
            else:
                scores[i] = scores[i] - id2mean[index[i]] # 计算优势

    return scores, scores # 返回优势和回报（此处相同）
```

评论区精华

1. 优势计算逻辑的正确性争议：gemini-code-assist[bot] 指出 compute_flow_grpo_outcome_advantage 中存在两个关键问题：(a) 对于单样本组，优势应设为 0，但原实现将组均值设为 0 导致优势等于样本分数，作者 zhtmike 已修复；(b) 标准偏差计算应在样本级别而非扩展的时间步张量上进行，否则会因重复而低估标准差，导致优势膨胀，作者回应“遵循 flowgrpo 实现”，未作修改。
2. 配置设计权衡：审阅者 SamitHuang 建议在配置中明确 extra_configs 可接受的参数列表以提高易用性，作者 zhtmike 承认这是临时设计，将在后续 PR 中重构配置以避免臃肿，并

计划提供更清晰的结构。

3. 代码细节修正：SamitHuang 指出数据预处理脚本中 local_save_dir 未使用 os.path.expanduser 展开用户目录，可能导致路径错误，作者在后续提交中采纳建议并更新。
4. FSDP 包装器方法转发疑问：gemini-code-assist[bot] 认为 verl/workers/engine/fsdp/diffusers_impl.py 中的 disable_adapter 上下文管理器直接调用 self.module.disable_adapters() 可能因 self.module 是 FSDP 包装器而失败，作者回应“FSDP 自动处理包装属性”，认为现有实现可行。

- GRPO 优势计算中单样本组和标准差计算问题 (correctness): 作者修复了单样本组问题，但拒绝了修正标准差计算的建议，表示遵循原始 FlowGRPO 实现。
- 配置设计易用性和 extra_configs 字段 (design): 作者承认 extra_configs 是临时设计，将在后续 PR 中重构配置以避免臃肿，并计划提供更清晰的结构。
- 数据预处理脚本路径展开问题 (correctness): 作者在后续提交中采纳建议，更新了代码以正确展开用户目录路径。

风险与影响

- 风险：1. 算法正确性风险：diffusion_algos.py 中的优势计算未按审阅建议修正标准差计算方式（在扩展的时间步张量上计算），可能导致优势值被高估，影响训练稳定性和策略优化效果。 2. 兼容性风险：新增的 RayFlowGRPOTrainer 强制要求使用新引擎路径（trainer.use_legacy_worker_impl=disable），与旧有工作流不兼容，可能影响现有用户的迁移。 3. 配置复杂性风险：临时引入的 extra_configs 字段缺乏明确文档和验证，用户可能错误配置或难以理解可用参数，增加使用门槛和调试成本。 4. 依赖风险：示例脚本依赖特定版本的 vllm 和 vllm-omni（0.18），若环境不匹配可能导致运行时错误。
- 影响：1. 对系统的影响：为 VERL 框架新增了扩散模型强化学习训练能力，扩展了其应用场景至图像生成等领域。新训练器与现有 PPO 训练器并行，不影响原有语言模型训练流程。 2. 对用户的影响：研究人员和工程师现在可以使用 FlowGRPO 算法对扩散模型进行 RL 训练，并通过提供的 OCR 示例快速上手。但需要适应新的配置结构和依赖要求。 3. 对团队的影响：引入了新的训练器模块和算法，需要团队成员熟悉扩散模型特有的数据处理、优势计算和指标统计逻辑。配置的临时性设计暗示后续将有重构，需关注演进。
- 风险标记：算法正确性争议，配置临时设计，兼容性约束

关联脉络

- PR #5297 [trainer] feat: flowgrpo trainer (part 1): PR body 中提及本 PR 是继 #5297 之后的“最后一环”，两者共同构成 FlowGRPO 训练器的完整实现，可能存在功能拆分或依赖关系。
- PR #5997 [trainer,algo] feat: Support On-Policy Distillation in main_ppo_sync: 同为训练器模块的新增功能，涉及算法集成和资源池管理，可对比学习 VERL 中训练器的扩展模式。
- PR #5978 [tool, rollout, cfg] feat: per-sample tool environment routing for ToolAgentLoop: 都涉及 rollout 配置和实验性功能扩展，展示了 VERL 在多样化任务（工具使用、图像生成）上的 RL 适配思路。