PR #2114 完整报告

THUDM/slime

fix(ppo): preserve raw KL so rollout/kl logging is correct

合并时间：2026-08-21 10:51

原文链接：http://prhub.com.cn/THUDM/slime/pull/2114

执行摘要

- 一句话：修复 PPO 原地修改 KL 张量导致日志指标失真
- 推荐动作：值得精读。这是一个典型的「Python 原地修改导致别名污染」bug：表面只改了几行，但根因是共享引用被隐式写入。建议关注两点：一是 out-of-place 修复如何在保持训练语义不变的前提下修正日志数据契约；二是回归测试用 sys.modules monkeypatch 构造 megatron 桩的手法，可用于同类依赖重型后端的单测场景。

功能与动机

PR body 明确指出：compute_advantages_and_returns 把 per-token KL 写入 rollout_data["kl"]，而该字段正被当作 KL 指标记录；PPO 分支随后 k *= kl_coef 原地修改了同一张量，并在 cp_rank == 0 时于最后一个 token 上叠加 scalar reward，导致日志看到的不是 KL。GRPO / GSPO / CISPO / R++ 均不这样修改 kl，因此这是 PPO 独有的正确性缺陷。

实现拆解

1. 定位根因：在 slime/backends/megatron_utils/loss.py 的 compute_advantages_and_returns 中，PPO 分支通过 k *= kl_coef 原地缩放 KL 张量。因为 k 与 rollout_data["kl"] 是同一张量引用，日志指标在计算优势前后悄悄变成了「缩放后叠加 reward」的张量。
2. 修复核心逻辑：改为 token_level_rewards = per_token_kl * kl_coef 的 out-of-place 乘法，再在 cp_rank == 0 时于最后一个 token 上加 reward，rewards 列表继续交给 get_advantages_and_returns_batch 做 GAE 计算；rollout_data["kl"] 保持原始 KL。数值上新旧逻辑等价（per_token_kl * kl_coef 与 k *= kl_coef 结果相同），因此训练行为不变，只修正了日志数据契约。
3. 新增回归测试：新建 tests/test_ppo_kl_metric.py，用 monkeypatch 构造 megatron / megatron.core 模块桩，直接调用 compute_advantages_and_returns，断言 PPO 计算后 rollout_data["kl"][0] 与 compute_approx_kl 的期望值一致。测试覆盖了 kl_loss_type="k1"、kl_coef=0.05、单卡上下文并关闭 use_opd 的默认路径。
4. CI 配套：在 .github/workflows/pr-test.yml 与模板 pr-test.yml.j2 的测试清单中同步加入 test_ppo_kl_metric.py（num_gpus=0），确保该回归测试在无 GPU 的常规 PR 测试中执行。

关键文件：

- slime/backends/megatron_utils/loss.py（模块 损失计算；类别 source；类型 core-logic；符号 compute_advantages_and_returns）：核心修复文件：PPO 分支从原地修改 KL 张量改为 out-of-place 构建 token-level rewards，是本次变更的主路径。
- tests/test_ppo_kl_metric.py（模块 指标测试；类别 test；类型 test-coverage；符号 test_ppo_estimator_does_not_corrupt_logged_kl）：新增回归测试：验证 PPO 计算后 rollout_data["kl"] 仍等于 compute_approx_kl，是本 PR 唯一的测试覆盖。
- .github/workflows/pr-test.yml（模块 CI 配置；类别 infra；类型 infrastructure）：CI 工作流中注册新测试，确保回归测试在 PR 阶段自动执行。
- .github/workflows/pr-test.yml.j2（模块 CI 配置；类别 infra；类型 infrastructure）：CI 模板文件同步新增测试条目，保证重新生成工作流时配置不丢失。

关键符号：compute_advantages_and_returns, test_ppo_estimator_does_not_corrupt_logged_kl

关键源码片段

slime/backends/megatron_utils/loss.py

核心修复文件：PPO 分支从原地修改 KL 张量改为 out-of-place 构建 token-level rewards，是本次变更的主路径。

```
# compute_advantages_and_returns 中，先统一计算 per-token KL 并存入
# rollout_data["kl"]，供后续日志上报（GRPO / GSPO / CISPO / R++ 均只读它）。
if args.kl_coef == 0 or not log_probs:
    # 当 kl_coef 为 0 时不会计算 ref_log_prob，用零张量占位
    xs = log_probs or rollout_log_probs or values
    kl = [torch.zeros_like(x, dtype=torch.float32, device=x.device) for x in xs]
else:
    kl = [
        compute_approx_kl(log_probs[i], ref_log_probs[i], kl_loss_type=args.kl_loss_type)
        for i in range(len(log_probs))
    ]
rollout_data["kl"] = kl

# PPO 分支修复：旧写法 k *= kl_coef 会原地改写 k 指向的张量，而 k 正是
# 上面存入 rollout_data["kl"] 的对象，日志随之被污染。现在先乘出新的
# token_level_rewards 张量，再在 cp_rank == 0 时于最后一个 token 上叠加
# scalar reward；rewards 仍交给 GAE 计算，原始 per_token_kl 保持纯净。
elif args.advantage_estimator == "ppo":
    old_rewards = rewards
    rewards = []
    kl_coef = -args.kl_coef
    cp_rank = mpu.get_context_parallel_rank()
    for reward, per_token_kl in zip(old_rewards, kl, strict=False):
        token_level_rewards = per_token_kl * kl_coef
        if cp_rank == 0:
            token_level_rewards[-1] += reward
        rewards.append(token_level_rewards)
```

```
    advantages, returns = get_advantages_and_returns_batch(
        total_lengths, response_lengths, values, rewards, args.gamma, args.lambd
    )
```

tests/test_ppo_kl_metric.py

新增回归测试：验证 PPO 计算后 rollout_data["kl"] 仍等于 compute_approx_kl，是本 PR 唯一的测试覆盖。

```
# 回归测试：PPO 计算完成后，rollout_data["kl"] 必须仍是原始近似 KL。
# 通过 monkeypatch 注入 megatron / megatron.core 模块桩，隔离真实框架依赖。
def test_ppo_estimator_does_not_corrupt_logged_kl(monkeypatch):
    # 弹出真实模块，塞入最小桩：单卡上下文、pipeline 末级
    previous_loss = sys.modules.pop("slime.backends.megatron_utils.loss", None)
    previous_cp_utils = sys.modules.pop("slime.backends.megatron_utils.cp_utils", None)
    mpu_stub = types.SimpleNamespace(
        get_context_parallel_world_size=lambda: 1,
        get_context_parallel_rank=lambda: 0,
        is_pipeline_last_stage=lambda: True,
    )
    megatron_mod = types.ModuleType("megatron")
    core_mod = types.ModuleType("megatron.core")
    core_mod.mpu = mpu_stub
    monkeypatch.setitem(sys.modules, "megatron", megatron_mod)
    monkeypatch.setitem(sys.modules, "megatron.core", core_mod)

    try:
        from slime.backends.megatron_utils.loss import compute_advantages_and_returns

        # 构造一组简单的 log_probs / ref_log_probs，先算出期望 KL
        log_probs = [torch.tensor([0.5, 0.7, 0.9])]
        ref_log_probs = [torch.tensor([0.4, 0.5, 0.6])]
        expected_kl = compute_approx_kl(log_probs[0], ref_log_probs[0], kl_loss_type="k1")
        rollout_data = {
            "log_probs": log_probs,
            "ref_log_probs": ref_log_probs,
            "rewards": [1.0],
            "values": [torch.zeros(3)],
            "response_lengths": [3],
            "total_lengths": [5],
            "loss_masks": [torch.ones(3)],
        }
        args = Namespace(
            advantage_estimator="ppo",
            kl_coef=0.05,
            kl_loss_type="k1",
            use_rollout_logprobs=False,
            custom_advantage_function_path=None,
            normalize_advantages=False,
            use_opd=False,
```

```
            gamma=1.0,
            lambd=1.0,
        )
        compute_advantages_and_returns(args, rollout_data)
        # 核心断言：PPO 之后 KL 指标仍是原始 KL，而不是缩放 + 叠加 reward 的张量
        torch.testing.assert_close(rollout_data["kl"][0], expected_kl)
    finally:
        # 恢复被弹出的真实模块，避免污染其他测试
        if previous_loss is None:
            sys.modules.pop("slime.backends.megatron_utils.loss", None)
        else:
            sys.modules["slime.backends.megatron_utils.loss"] = previous_loss
        if previous_cp_utils is None:
            sys.modules.pop("slime.backends.megatron_utils.cp_utils", None)
        else:
            sys.modules["slime.backends.megatron_utils.cp_utils"] = previous_cp_utils
```

评论区精华

该 PR 没有形成多轮 review 交锋，仅有作者 EazyReal 两次向合入者 zhuzilin 请求 review 的评论，但其中包含了关键设计说明：

> EazyReal：PPO reward shaping was mutating the raw KL tensor before metrics, so rollout/kl logging could report shaped rewards instead of KL. The fix keeps raw KL separate and mirrors the local-k pattern used by the reinforce loss helpers.

第二次评论进一步补充：修复后 PPO 仍正常施加 KL 惩罚，只是日志恢复正确。即「训练行为不变、监控指标修正」是本次变更的设计边界。

- PPO reward shaping 原地修改 KL 导致日志失真 (correctness): 采用 out-of-place 方式构建 token_level_rewards，PPO 继续施加 KL 惩罚，rollout_data['kl'] 保持原始 KL 指标；该方案已随 PR 合入解决。

风险与影响

- 风险：
  - 训练数值等价性：per_token_kl * kl_coef 与旧写法 k *= kl_coef 的值完全一致，PPO 的 advantages / returns 及权重更新不受影响，这是本 PR 风险最低的关键点。
  - 下游行为变化：任何依赖 rollout_data["kl"] 已被缩放后值的隐式逻辑会观察到新行为。从代码看该字段仅用于日志上报，但修复后若存在未发现的消费者，其含义会从「scaled + reward」变回「raw KL」。
  - 上下文并行语义：cp_rank == 0 分支保留，非 0 rank 不叠加 reward，跨 rank 通信模式没有变化；新张量 token_level_rewards 与旧 k 一样是本地构造，无新增通信开销。
  - 测试可信度：测试通过模块桩隔离 megatron 依赖，若真实 Megatron 环境的 mpu 行为或批处理约定与桩不一致，存在漏测可能；但断言点单一明确，足以覆盖本次回归。
  - 性能：每个样本多一次张量乘法的临时分配，相比 GAE 与反向传播开销可忽略。
- 影响：

- 用户（实验者）：修复后 rollout/kl 曲线反映真实 KL，此前监控图中可能出现负值或偏移（因为 kl_coef = -args.kl_coef 缩放后会改变符号和数值），影响实验判断。
- 系统：PPO 训练热路径上多一次轻量张量分配，数值语义不变；本次变更明确了 rollout_data["kl"] 只保存原始 KL 的数据契约，与 GRPO / GSPO / CISPO / R++ 的行为对齐。
- 团队：为后续新增 advantage estimator 提供了明确的约定——KL 指标字段不可被原地改写，reward shaping 必须走独立张量；测试文件同时示范了如何用模块桩测试依赖 megatron 的内部函数。
- 风险标记：核心训练路径变更，张量原地修改导致潜在下游依赖变化，测试依赖模块桩

关联脉络

- PR #2247 fix: forward dual-clip PPO epsilon: 同改 slime/backends/megatron_utils/loss.py，同属 PPO 损失计算正确性修复线。
- PR #2235 fix: whiten advantages over the DP group that includes context parallel: 同改 loss.py，涉及优势归一化与进程组的正确性，与本 PR 共享同一函数上下文。
- PR #2266 Refactor --save-debug-train-data: 同改 loss.py，重构训练数据转储路径，与本 PR 都触及 compute_advantages_and_returns 周边行为。
- PR #2205 perf: vectorize REINFORCE++ discounted returns: 涉及 slime/utils/ppo_utils.py 的 KL 与回报计算，与本 PR 的 KL 指标语义相关。