PR #2067 完整报告

THUDM/slime

[algo] Add CISPO advantage estimator (MiniMax-M1)

合并时间：2026-06-15 15:08

原文链接：http://prhub.com.cn/THUDM/slime/pull/2067

执行摘要

- 一句话：新增 CISPO 优势估计器，保留裁剪 token 的梯度
- 推荐动作：值得精读。实现简洁，与现有架构无缝集成，是研究 RL 损失裁剪策略的优秀示例。关注点：stop-gradient 的写法、配置警告的用法、测试的梯度路由验证方法。

功能与动机

PPO/GRPO 风格裁剪中，重要性采样比率超出 clip band 的 token 贡献零梯度，因为裁剪后的代理损失 min(r*A, clip(r)*A) 完全杀死了该 token 的更新。MiniMax-M1 (https://arxiv.org/abs/2506.13585) 指出，在 off-policy 更新下，这些不成比例的低概率 "fork" token（如 However、Recheck、Wait）恰好是推理 RL 需要梯度的推理分支。CISPO 通过 stop-gradient 裁剪 IS 权重，使每个 token 的梯度都通过 log_probs 流动。slime 目前没有 CISPO 支持。

实现拆解

1. 核心损失函数：在 slime/utils/ppo_utils.py 中新增 compute_cispo_loss，使用 ratio_truncated.detach() 实现 stop-gradient，公式为 -sg(clip(ratio, 1 - eps_clip, 1 + eps_clip_high)) * advantages * log_probs。
2. 集成到训练流程：在 slime/backends/megatron_utils/loss.py 中导入 compute_cispo_loss，在 policy_loss_function 中添加 if args.advantage_estimator == "cispo" 分支；同时在 compute_advantages_and_returns 中将 "cispo" 加入 grpo 返回和标准化路径。
3. 配置与验证：在 slime/utils/arguments.py 中将 "cispo" 加入 --advantage-estimator 的 choices，并在 slime_validate_args 中增加警告：当 eps_clip < 1.0 时提醒 canonical 单边设置。
4. Rollout 奖励处理：在 slime/ray/rollout.py 的 _post_process_rewards 中将 "cispo" 加入需要进行 group normalization 和 std normalization 的 estimator 列表。
5. 测试：新增 tests/test_cispo_loss.py，包含两个参数化测试：验证 loss 值与闭形式匹配、验证梯度仅流经 log_probs 而不经过 IS ratio 路径。
6. 基础设施与文档：在 .github/workflows/pr-test.yml.j2 中将测试文件加入 CPU 测试矩阵，生成更新 pr-test.yml；中英文使用文档添加 cispo 的简要说明。

关键文件：

- slime/utils/ppo_utils.py（模块 损失函数；类别 source；类型 core-logic；符号 compute_cispo_loss）：新增 compute_cispo_loss 函数，实现 CISPO 核心损失计算
- slime/backends/megatron_utils/loss.py（模块 损失集成；类别 source；类型 core-logic）：将 compute_cispo_loss 导入并集成到 policy_loss_function 和 compute_advantages_and_returns 中
- slime/utils/arguments.py（模块 配置；类别 source；类型 core-logic）：将 cispo 加入 --advantage-estimator 选项，添加配置验证警告
- slime/ray/rollout.py（模块 Rollout；类别 source；类型 core-logic）：将 cispo 加入 reward group normalization 和 std normalization 的条件列表
- tests/test_cispo_loss.py（模块 测试；类别 test；类型 test-coverage；符号 test_compute_cispo_loss_matches_closed_form_surrogate, test_compute_cispo_loss_gradient_flows_only_through_log_probs）：新增 CPU 测试，验证 CISPO 损失闭形式匹配和梯度路由正确性
- .github/workflows/pr-test.yml（模块 CI；类别 infra；类型 infrastructure）：将 test_cispo_loss.py 加入 CPU 测试矩阵

关键符号：compute_cispo_loss, test_compute_cispo_loss_matches_closed_form_surrogate, test_compute_cispo_loss_gradient_flows_only_through_log_probs

关键源码片段

slime/utils/ppo_utils.py

新增 compute_cispo_loss 函数，实现 CISPO 核心损失计算

```
# slime/utils/ppo_utils.py（新增）

@torch.compile(dynamic=True)
def compute_cispo_loss(
    ppo_kl: torch.Tensor,
    log_probs: torch.Tensor,
    advantages: torch.Tensor,
    eps_clip: float,
    eps_clip_high: float,
):
    """CISPO 损失来自 MiniMax-M1 (https://arxiv.org/abs/2506.13585, Eq. 4-5):
    ``-sg(clip(ratio, 1 - eps_clip, 1 + eps_clip_high)) * advantages * log_probs``.

    与 PPO 不同，IS 比率在 stop-gradient 下裁剪，梯度流经 ``log_probs``，
    因此裁剪 token 仍然贡献梯度。边界复用 ``compute_policy_loss`` 的 delta-from-1 约定；
    canonical CISPO 禁用下界 (``eps_clip >= 1.0``)。
    """
    ratio = (-ppo_kl).exp() # 计算重要性采样比率 exp(-KL)
    # 在 stop-gradient 下裁剪比率
    ratio_truncated = torch.clamp(ratio, min=1.0 - eps_clip, max=1.0 + eps_clip_high)
    # 损失 = - 裁剪比率（停止梯度）* 优势 * log_prob，梯度只传回 log_prob
    pg_losses = -ratio_truncated.detach() * advantages * log_probs
```

```
    # clipfrac：标记哪些 token 的原始比率超出了 band（非 PPO 的 loss 影响裁剪）
    clipfrac = (ratio_truncated != ratio).float()
    return pg_losses, clipfrac
```

tests/test_cispo_loss.py

新增 CPU 测试，验证 CISPO 损失闭形式匹配和梯度路由正确性

```
# tests/test_cispo_loss.py（新增）

"""CPU 测试 for compute_cispo_loss (MiniMax-M1, https://arxiv.org/abs/2506.13585)."""

import math

import pytest
import torch

from slime.utils.ppo_utils import compute_cispo_loss

NUM_GPUS = 0 # 标记仅 CPU 测试

# 固定数据
ADVANTAGES = torch.tensor([1.0, -0.5, 2.0, -1.0])
LOG_PROBS = torch.tensor([-0.7, -1.2, -0.4, -2.1])

# 测试用例：(eps_clip, eps_clip_high, 原始 IS ratios, 裁剪后 ratios)
CLIP_CASES = [
    pytest.param(0.2, 0.28, [1.0, 1.14, 1.56, 0.4], [1.0, 1.14, 1.28, 0.8], id="ppo_band"),
    pytest.param(1.0, 4.0, [1.0, 3.0, 9.0, 0.4], [1.0, 3.0, 5.0, 0.4], id="wide_minimax_band"),
]


@pytest.mark.parametrize("eps_clip, eps_clip_high, ratios, clamped", CLIP_CASES)
def test_compute_cispo_loss_matches_closed_form_surrogate(eps_clip, eps_clip_high, ratios,
clamped):
    # 构造 ppo_kl = -log(ratio)
    ppo_kl = -torch.tensor([math.log(r) for r in ratios])
    pg_losses, clipfrac = compute_cispo_loss(ppo_kl, LOG_PROBS, ADVANTAGES, eps_clip, eps_
    clip_high)
    # 期望损失：-clamped * advantages * log_probs
    expected_losses = -torch.tensor(clamped) * ADVANTAGES * LOG_PROBS
    torch.testing.assert_close(pg_losses, expected_losses, rtol=1e-6, atol=1e-6)
    # clipfrac 应标记比率离开 band 的位置
    expected_clipfrac = torch.tensor([float(c != r) for c, r in zip(clamped, ratios, strict=True)])
    torch.testing.assert_close(clipfrac, expected_clipfrac)


@pytest.mark.parametrize("eps_clip, eps_clip_high, ratios, clamped", CLIP_CASES)
def test_compute_cispo_loss_gradient_flows_only_through_log_probs(eps_clip, eps_clip_high,
ratios, clamped):
```

```
    # 验证梯度只流经 log_probs，而非 IS 比率路径
    log_ratios = torch.tensor([math.log(r) for r in ratios], requires_grad=True)
    ppo_kl = -log_ratios
    log_probs = LOG_PROBS.clone().requires_grad_()
    pg_losses, _ = compute_cispo_loss(ppo_kl, log_probs, ADVANTAGES, eps_clip, eps_clip_high)
    pg_losses.sum().backward()
    # log_probs 的梯度应为 -clamped * advantages
    torch.testing.assert_close(log_probs.grad, -torch.tensor(clamped) * ADVANTAGES, rtol=1e-6,
    atol=1e-6)
    # log_ratios 的梯度应为 0（因为 stop-gradient）
    assert log_ratios.grad is None or torch.all(
        log_ratios.grad == 0
    ), f"CISPO 必须 stop-gradient IS 比率; log_ratios.grad = {log_ratios.grad}"
```

评论区精华

本 PR 无公开的 review 讨论。但从 commit body 和代码结构可以看出设计上的关键决策：复用现有 `--eps-clip` / `--eps-clip-high` 标志，与 GSPO 保持一致的 band 约定；`pg_clipfrac` 在 CISPO 下定义为 band 退出率（非 PPO 的 loss 影响裁剪率）；`slime_validate_args` 对 `eps_clip < 1.0` 给出警告，建议 canonical 单边设置 `--eps-clip 1.0 --eps-clip-high 4.0`。

- 暂无高价值评论线程

风险与影响

- 风险：回归风险：新损失函数仅在 --advantage-estimator cispo 时激活，默认不变；但 rollout.py 和 arguments.py 中的条件列表扩展可能遗漏与其他 estimator 的兼容性测试。性能：compute_cispo_loss 使用 @torch.compile(dynamic=True)，开销极低。正确性：stop-gradient 实现依赖 ratio_truncated.detach()，测试覆盖了梯度路由。配置风险：eps_clip 默认 0.2 会保留下界裁剪，与 canonical CISPO 单边行为不同，警告可避免误用。
- 影响：用户：研究者和工程师可以通过 --advantage-estimator cispo 直接使用新算法，无需额外代码。系统：无破坏性变更，不影响现有训练脚本。团队：维护成本低，代码集中在损失函数和配置层，测试覆盖关键不变性。
- 风险标记：新损失函数分支，配置警告，梯度路由验证，默认行为不变

关联脉络

- 暂无明显关联 PR