PR #1966 完整报告

radixark/miles

[Bug Fix][OPD] Stop gradients through old-policy and teacher scores

合并时间：2026-08-09 23:34

原文链接：http://prhub.com.cn/radixark/miles/pull/1966

执行摘要

- 一句话：OPD 固定打分统一切断梯度，防计算图泄漏
- 推荐动作：值得精读。该 PR 是训练框架中“固定输入边界”的典型实现，适合负责自定义 loss、TIS、rollout 数据流或 OPD 蒸馏的同学阅读；重点关注 compute_advantages_and_returns 的持久化 detach 与 policy_loss_function 的防御性 detach 如何互补，以及自定义 TIS 兼容性的取舍。

功能与动机

PR body 明确指出：OPD 和 PPO 式策略梯度训练把 teacher、reference-policy、pre-update policy、rollout-policy 与 advantage 张量当作固定训练输入；标准打分路径不会产生梯度，但自定义与融合路径可能破坏该契约。若缺少显式边界，一个“活的”旧策略张量可能进入 PPO/GSPO 的 importance ratio，在可微的 current-policy 分支旁保留不必要的计算图。本 PR 的目标就是在训练边界强制该契约。

实现拆解

1. 持久化边界统一切断计算图：在 miles/backends/training_utils/loss.py 新增模块级辅助函数 _detach_rollout_tensor_list(rollout_data, key)，对列表中每个张量调用 detach() 并写回 rollout_data。compute_advantages_and_returns() 在非末级 pipeline 检查之后、KL 与 advantage 估计之前，依次对 log_probs、rollout_log_probs、ref_log_probs、teacher_log_probs 调用该函数，使 rollout 阶段产出的所有打分成为固定训练数据。
2. 损失消费边界防御性 detach：在 miles/backends/training_utils/loss_hub/losses.py 的 policy_loss_function() 中，advantages 在 torch.cat 前逐张 detach；scored_old_log_probs、rollout_old_log_probs、reference_log_probs 均从 batch 中生成 detached 副本，并按 use_rollout_logprobs 选择 old_log_probs，同时补充明确 assert，缺失必备输入时直接报错。这样即使调用方绕过持久化边界，也不会把图带进 importance ratio。
3. OPD 局部设备对齐与 detach：loss_hub/opd.py 的 apply_opd_kl_to_advantages() 中，teacher 打分先 detach 并写回 rollout_data 保持原始设备，仅在做 reverse-KL 前临时搬到 student 设备；预计算 opd_reverse_kl 分支同样先 detach 再 to(device)，防止直接调用者绕过 compute_advantages_and_returns 时泄漏梯度。
4. 消费者适配与校验收紧：TIS 分支区分自定义与内置实现——自定义 TIS 允许 train_log_probs 缺失，内置 vanilla_tis_function 则严格断言 scored_old_log_probs 与

rollout_old_log_probs 都存在；use_kl_loss 分支断言 ref_log_probs 存在；debug dump、mismatch metrics 与 train_rollout_logprob_abs_diff 全部改用 detached 副本。

5. 测试配套：test_true_on_policy_loss_metrics.py 新增 3 个 backward 级测试，验证 policy loss 只通过 current-policy logits 获得梯度（覆盖 use_rollout_logprobs 两种取值）、reference KL 路径不回传梯度、自定义 TIS 可容忍缺失 trainer-scored log-probs；test_opd.py 新增持久化 rollout 数据全字段 detached 断言与预计算 reverse-KL 不回流梯度的测试。

关键文件：

- miles/backends/training_utils/loss.py（模块 损失模块；类别 source；类型 core-logic；符号 _detach_rollout_tensor_list, compute_advantages_and_returns）：核心源码主路径：新增 _detach_rollout_tensor_list，并在 compute_advantages_and_returns 入口统一 detach 四类固定打分张量，是本次修复的持久化边界。
- miles/backends/training_utils/loss_hub/losses.py（模块 损失中心；类别 source；类型 core-logic；符号 policy_loss_function）：核心消费边界：policy_loss_function 对 advantages 与各类固定打分做防御性 detach，并调整 TIS、KL loss 分支的输入校验，是梯度边界的第二道防线。
- miles/backends/training_utils/loss_hub/opd.py（模块 OPD 模块；类别 source；类型 core-logic；符号 apply_opd_kl_to_advantages）：OPD 蒸馏的局部实现：teacher 打分与预计算 reverse-KL 增加 detach，并保持持久数据的原始设备，是 OPD 路径梯度泄漏的修复点。
- tests/fast/backends/training_utils/test_true_on_policy_loss_metrics.py（模块 损失测试；类别 test；类型 test-coverage；符号 test_policy_loss_only_backpropagates_through_current_policy, test_kl_loss_does_not_backpropagate_through_reference_scores, test_custom_tis_can_ignore_missing_trainer_scored_log_probs）：新增 backward 级回归测试：验证 policy loss 只通过 current-policy logits 反向传播、reference KL 路径不回传梯度、自定义 TIS 可容忍缺失 trainer-scored log-probs。
- tests/fast/backends/training_utils/loss/test_opd.py（模块 OPD 测试；类别 test；类型 test-coverage；符号 test_precomputed_reverse_kl_is_detached_before_weighting_advantages, test_fixed_opd_inputs_are_detached_in_persistent_rollout_data）：新增 OPD 固定输入 detached 断言与预计算 reverse-KL 不回流梯度的测试，并增强原有反向 KL 测试的梯度验证。

关键符号：_detach_rollout_tensor_list, compute_advantages_and_returns, policy_loss_function, apply_opd_kl_to_advantages

关键源码片段

miles/backends/training_utils/loss.py

核心源码主路径：新增 _detach_rollout_tensor_list，并在 compute_advantages_and_returns 入口统一 detach 四类固定打分张量，是本次修复的持久化边界。

```
# 持久化边界统一切断计算图：rollout 阶段产出的打分是固定训练数据，
```

```
# 不应携带任何 autograd graph 进入策略更新阶段。
def _detach_rollout_tensor_list(rollout_data: RolloutBatch, key: str) -> list[torch.Tensor] | None:
    tensors = rollout_data.get(key)
    if tensors is None:
        return None
    detached_tensors = [tensor.detach() for tensor in tensors]
    rollout_data[key] = detached_tensors
    return detached_tensors


def compute_advantages_and_returns(args: Namespace, rollout_data: RolloutBatch) -> None:
    log_probs_key = "rollout_log_probs" if args.use_rollout_logprobs else "log_probs"
    log_probs: list[torch.Tensor] = rollout_data.get(log_probs_key)
    values = rollout_data.get("values")

    # 非最后一个 pipeline 阶段没有完整打分，直接返回，不触发 detach。
    if log_probs is None and values is None:
        return

    # 权威持久化边界：先切断旧策略、rollout 策略、参考策略与 teacher 打分的
    # 计算图，再计算 KL 与 advantage，保证固定输入契约对所有打分路径生效。
    _detach_rollout_tensor_list(rollout_data, "log_probs")
    _detach_rollout_tensor_list(rollout_data, "rollout_log_probs")
    _detach_rollout_tensor_list(rollout_data, "ref_log_probs")
    _detach_rollout_tensor_list(rollout_data, "teacher_log_probs")

    # detach 之后重新读取，后续 KL 与 advantage 计算都基于固定输入。
    log_probs = rollout_data.get(log_probs_key)
    ref_log_probs = rollout_data.get("ref_log_probs")

    if args.kl_coef == 0 or not log_probs:
        xs = log_probs if log_probs is not None else 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))
        ]

    advantages, returns = compute_advantages(
        args=args,
        kl=kl,
        rewards=rollout_data.get("rewards"),
        log_probs=log_probs,
        loss_masks=rollout_data.get("loss_masks"),
        total_lengths=rollout_data.get("total_lengths"),
        response_lengths=rollout_data.get("response_lengths"),
        max_seq_lens=rollout_data.get("max_seq_lens", None),
        values=values,
```

```
    )
```

miles/backends/training_utils/loss_hub/losses.py

核心消费边界：policy_loss_function 对 advantages 与各类固定打分做防御性 detach，并调整 TIS、KL loss 分支的输入校验，是梯度边界的第二道防线。

```
# 消费边界防御：即使调用方没有在持久化边界 detach，
# 这里也强制 advantages 与各类固定打分脱离计算图。
def policy_loss_function(
    args: Namespace,
    batch: RolloutBatch,
    logits: torch.Tensor,
    sum_of_sample_mean: Callable[[torch.Tensor], torch.Tensor],
) -> tuple[torch.Tensor, dict[str, torch.Tensor]]:
    parallel_state = get_parallel_state()
    advantages_list = [advantage.detach() for advantage in batch["advantages"]]
    advantages = torch.cat(advantages_list, dim=0)
    scored_old_log_probs = (
        [log_prob.detach() for log_prob in batch["log_probs"]] if batch.get("log_probs") is not None
        else None
    )
    rollout_old_log_probs = (
        [log_prob.detach() for log_prob in batch["rollout_log_probs"]]
        if batch.get("rollout_log_probs") is not None
        else None
    )
    reference_log_probs = (
        [log_prob.detach() for log_prob in batch["ref_log_probs"]] if batch.get("ref_log_probs") is
        not None else None
    )

    # 按配置选择 importance ratio 的旧策略打分，缺输入时直接报错而不是静默失败。
    if args.use_rollout_logprobs:
        assert rollout_old_log_probs is not None, "rollout_log_probs must be provided when --use-
        rollout-logprobs is set"
        old_log_probs = rollout_old_log_probs
    else:
        assert scored_old_log_probs is not None, "log_probs must be provided for policy loss"
        old_log_probs = scored_old_log_probs

    # current-policy 的 log_probs 仍由可微 logits 计算，梯度只应流经这一条分支；
    # 后续 TIS、KL loss、debug dump 与 mismatch metrics 全部使用上述 detached 副本。
    log_probs_and_entropy = get_log_probs_and_entropy(
        logits,
        args=args,
        unconcat_tokens=batch["unconcat_tokens"],
        total_lengths=batch["total_lengths"],
        response_lengths=batch["response_lengths"],
        with_entropy=args.entropy_coef != 0 or args.observe_training_entropy,
```

```
        entropy_requires_grad=args.entropy_coef != 0,
        max_seq_lens=batch.get("max_seq_lens", None),
    )
    log_probs = log_probs_and_entropy["log_probs"]
```

miles/backends/training_utils/loss_hub/opd.py

OPD 蒸馏的局部实现：teacher 打分与预计算 reverse-KL 增加 detach，并保持持久数据的原始设备，是 OPD 路径梯度泄漏的修复点。

```
# OPD 的 reverse-KL 惩罚：student 与 teacher 打分都是固定输入，
# 这里对直接调用者做防御性 detach，避免绕过 compute_advantages_and_returns 时泄漏梯度。
def apply_opd_kl_to_advantages(
    args: Namespace,
    rollout_data: RolloutBatch,
    advantages: list[torch.Tensor],
    student_log_probs: list[torch.Tensor] | None,
) -> None:
    if student_log_probs is None:
        return

    precomputed_reverse_kls = rollout_data.get("opd_reverse_kl")
    if precomputed_reverse_kls is not None:
        reverse_kls = []
        for i, adv in enumerate(advantages):
            reverse_kl = precomputed_reverse_kls[i]
            if not torch.is_tensor(reverse_kl):
                reverse_kl = torch.tensor(reverse_kl, dtype=torch.float32)
            # 防御性消费边界：预计算 reverse-KL 也必须脱离计算图。
            reverse_kl = reverse_kl.detach().to(device=adv.device)
            advantages[i] = adv - args.opd_kl_coef * reverse_kl
            reverse_kls.append(reverse_kl)
        rollout_data["opd_reverse_kl"] = reverse_kls
        return

    teacher_log_probs = rollout_data.get("teacher_log_probs")
    if teacher_log_probs is None:
        raise ValueError(f"OPD with opd_type='{args.opd_type}' requires teacher_log_probs, but it
        is missing.")

    # teacher 打分保持原设备写回持久数据，仅 OPD 计算前临时搬到 student 设备，
    # 避免设备迁移副作用污染持久化数据。
    device = student_log_probs[0].device
    detached_teacher_log_probs = [t.detach() for t in teacher_log_probs]
    rollout_data["teacher_log_probs"] = detached_teacher_log_probs
    teacher_log_probs = [t.to(device=device) for t in detached_teacher_log_probs]

    reverse_kls = []
    for i, adv in enumerate(advantages):
        # 旧 student 打分同样视为固定输入，reverse-KL 不能反向传播到打分源。
```

```
            old_student_log_prob = student_log_probs[i].detach()
            reverse_kl = old_student_log_prob - teacher_log_probs[i]
            advantages[i] = adv - args.opd_kl_coef * reverse_kl
            reverse_kls.append(reverse_kl)
        rollout_data["opd_reverse_kl"] = reverse_kls
```

评论区精华

本 PR 无实质 review 技术讨论。两位 reviewer 均 APPROVED：Shi-Dong 给出“LGTM”，maocheng23 无备注；两个 gemini-code-assist 机器人评论仅提示 Gemini Code Assist 消费者版本已停用。PR 作者在 body 中说明了关键设计权衡：teacher 打分在持久数据中保持原设备、设备对齐局部化到 OPD 计算，以最小化对既有数据流的影响；自定义 TIS 保留不消费 trainer-scored log-probs 的兼容性，而内置 TIS 保留严格输入校验。无未解决疑虑。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 梯度语义变更：任何非标准地依赖 old-policy / teacher 打分回传梯度的自定义 loss 将不再获得梯度，属预期 breaking change，需在文档与升级说明中强调。
  - 自定义 TIS 兼容性：assert "rollout_log_probs" in batch 被替换为分支内校验，自定义 TIS 不再强制要求 train_log_probs；若某个自定义 TIS 隐式依赖该键存在，可能从显式报错变为静默拿到 None。
  - 设备迁移语义：持久化 teacher_log_probs 保持在原设备，OPD 内临时迁移；依赖调用后 rollout_data["teacher_log_probs"] 已被搬到 student 设备的旧代码，行为会发生变化。
  - 回归风险：核心 loss 路径变更影响 PPO/GSPO/TIS/KL 全链路，测试覆盖了常见路径，但真实多卡 / pipeline 并行拓扑下的验证仅靠手工运行官方 Qwen3-8B / 32B OPD recipe。
- 影响：
  - 用户：所有使用 OPD 蒸馏或 PPO/GSPO 策略梯度训练的团队；修复后 loss 反向传播只经过 current-policy 分支，数值语义不变但梯度来源更干净。
  - 系统：切断多余计算图可减少反向传播时间和显存占用，尤其在长序列 + MoE 大规模训练中收益更明显。
  - 团队：确立“持久化边界 + 消费边界”双层 detach 契约，后续新增自定义 scorer、融合打分路径或自定义 TIS 时都有明确的安全网。
  - 影响程度：中高——代码集中在训练核心路径，但改动量小、行为语义（OPD 目标、系数、裁剪）保持不变。
  - 风险标记：核心训练路径变更，自定义 TIS 行为变化，设备迁移语义变化

关联脉络

- PR #1967 [Bug Fix] Always release rollout-engine broadcast lock on failure: 同为 rollout 边界健壮性 Bug Fix，与本 PR 一起构成 rollout 数据与资源边界契约的加固。
- PR #2030 [async] async data buffer: unified filters and better observability: 重构 fully_async 数据缓冲与 rollout_data 生命周期，本 PR 的持久化 detach 正落在这条数据流上。
- PR #2244 pass the engine weight version from the trainer instead of polling the router: 同属训练器与 rollout 引擎之间边界契约收紧，减少隐式依赖。