

PR #7762 完整报告

verl-project/verl

[trainer, ckpt] fix: skip V1 async warmup after inflight reissue

合并时间: 2026-09-10 13:52

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7762>

执行摘要

- 一句话: 修复 V1 异步训练恢复后重复提交 warmup 导致的跳样本与窗口溢出
- 推荐动作: 建议精读, 尤其是维护 V1 异步训练与 checkpoint 恢复链路的同学。三个值得关注的设计决策: (1) 门控依据从「reissue 动作返回值」改为「恢复状态占用计数」, 揭示了恢复类逻辑应以状态而非动作结果做门控; (2) 预取窗口的度量单位由 batch 改为 prompt, 并配套 `_add_prompts_to_generate(missing_prompts)`; (3) 恢复池超额时「只保不缩」, 用日志而非裁剪处理异常状态。此外, Begunner 在 review 中指出反例并直接提交修复的协作模式, 以及 PR body 中明确标注 AI 辅助定位、人工逐行复核的做法, 都值得作为流程参考。后续建议补齐两个覆盖缺口: 带 `global_steps` 过滤的计数测试, 以及打通真实 replay buffer / dataloader 的集成测试。

功能与动机

直接动机来自 issue #7759: 在 `resume_mode=auto` 的 V1 异步训练中, `reissue` 之后的额外 warmup 会让 dataloader 多消费并非保存时在途的行 (issue 原文: `prompts are skipped vs. a non-preempted run`), 并使 replay / in-flight window 相对 checkpoint 时的 `TransferQueue` 状态被 overfilled。Issue 也给出了期望行为: If `_reissue_inflight_prompts()` restored at least one in-flight prompt, skip the extra warmup loop; 同时要求 fresh start 或缺少 TQ checkpoint 支持时保留 warmup 回退, `skip.rollout_tq.enable` 继续短路。PR body 还说明这不与 #7401 (SFT epoch 边界 dataloader) 和 #7579 (crash-durable replay buffer RFC) 重复, 是恢复成功后的额外 warmup 问题。

实现拆解

1. 记录恢复态占用: `_count_tq_prompt_groups()` + `_restored_tq_prompt_count` ([verl/trainer/ppo/v1/trainer_base.py](#))

- 新增模块级函数 `_count_tq_prompt_groups(partition_id="train")`, 调用 `tq.kv_list(partition_id)` 遍历分区所有 tag, 只对 `is_prompt=True` 计数, 不区分 pending / running / finished / failure。
- `PPOTrainer.__init__` 新增 `self._restored_tq_prompt_count = 0`, 保证 fresh start 与不支持 TQ checkpoint 的环境下计数为 0, warmup 行为与旧版一致。

- `_load_checkpoint()` 在 `tq.load_checkpoint(tq_ckpt_path)` 成功之后立即写入该计数并打 INFO 日志。影响：warmup 门控依据从「reissue 动作的返回值」切换为「恢复状态的实际占用」，这是本次修复的核心转折。

2. warmup 由整 batch 追加改为差额补齐（同文件）

- 新增 `PPOTrainer._add_async_warmup_batches(num_warmup_batches)`：先短路 `config.skip.rollout_tq.enable` 与 `num_warmup_batches <= 0`；再用 `target_prompts = num_warmup_batches * config.data.train_batch_size` 换算目标窗口，计算 `missing_prompts = max(0, target_prompts - restored_prompts)`；最后通过 `self._add_prompts_to_generate(missing_prompts)` 一次性补齐。
- 与旧实现 `for _ in range(num_warmup_batches): self._add_batch_to_generate()` 相比，度量单位由 batch 变为 prompt，且允许提交非整数 batch 的差额。
- 恢复池达到或超过目标（`missing_prompts == 0`）时不缩减已恢复 prompt，只记录日志，保持「绝不丢弃在途 prompt」的保守策略。

3. 两个异步训练器接入共享 helper（`trainer_colocate_async.py` / `trainer_separate_async.py`）

- `PPOTrainerColocateAsync.on_train_begin` 由 6 行循环收敛为一行 `self._add_async_warmup_batches(self.config.trainer.v1.colocate_async.num_warmup_batches)`，并删除本文件内不再使用的 logging / os 导入与模块级 logger。
- `PPOTrainerSeparateAsync.on_train_begin` 同样收敛为一行调用。
- 影响：skip.rollout_tq 短路逻辑从两个子类上提到基类，未来新增异步模式默认继承修复，不会再出现「一个模式修了、另一个没修」。

4. 清理 fit() 中已失效的注释（`trainer_base.py`）

- 删除 `# SkipManager skips warmup batches in async trainers, so it doesn't conflict with reissue`。这行注释——该注释所描述的假设正是 bug 的认知来源（以为 SkipManager 一定会兜住冲突）。
- `fit()` 调用顺序不变：`self._reissue_inflight_prompts()` → `self.on_train_begin()`，即先恢复在途 prompt，再由 helper 决定是否补齐差额。

5. 测试配套

- 新增 `tests/trainer/ppo/v1/test_v1_async_warmup_on_cpu.py` (+136)：_stub() 用 OmegaConf + MagicMock 构造最小 trainer，覆盖 9 个场景：fresh start 提交整窗（12）、恢复 12 跳过、恢复 6 补 6、恢复 13 不补、num_warmup_batches=0 不提交、skip.rollout_tq=True 不提交、colocate_async/separate_async 两个 on_train_begin 入口的 top-up，以及 `_load_checkpoint` → `on_train_begin` 的计数传递（`global_steps==6`、`_restored_tq_prompt_count==6`、`_add_prompts_to_generate(6)`）。
- 扩展 `tests/trainer/ppo/v1/test_reissue_inflight_on_cpu.py` (+37)：`test_count_tq_prompt_groups_includes_every_prompt_status` 断言四种 prompt 状态全部计入、trajectory 不计入；`test_save_load_terminal_only_counts_prompts_without_reissue` 用真实 save/load 往返证明 terminal-only checkpoint 的 reissue 返回 0 但计数为 2——正是 reviewer 指出的那个反例。

- 无配置项、schema 或部署改动，`_reissue_inflight_prompts()` 的签名与返回值保持兼容。

关键文件：

- `verl/trainer/ppo/v1/trainer_base.py`（模块 训练器基类；类别 source；类型 core-logic；符号 `_count_tq_prompt_groups`, `_add_async_warmup_batches`, `_restored_tq_prompt_count`, `PPOTrainer.init`）：修复主体：新增 `_count_tq_prompt_groups()` 与 `_add_async_warmup_batches()`，在 `__init__` 初始化 `_restored_tq_prompt_count`，在 `_load_checkpoint()` 写入恢复计数，并删除 `fit()` 中已失效的 `SkipManager` 注释。
- `verl/trainer/ppo/v1/trainer_colocate_async.py`（模块 共置训练器；类别 source；类型 dependency-wiring；符号 `PPOTrainerColocateAsync.on_train_begin`）：
`on_train_begin` 由自写 warmup 循环改为调用基类 helper，并删除不再使用的 logging / os 导入与模块 logger，是「逻辑上提」的典型接线改动。
- `tests/trainer/ppo/v1/test_v1_async_warmup_on_cpu.py`（模块 异步预热测试；类别 test；类型 test-coverage；符号 `_stub`, `test_fresh_start_submits_the_whole_prefetch_window`, `test_full_restored_prompt_pool_skips_warmup`, `test_partial_restored_prompt_pool_is_topped_up`）：新增的针对性单测，覆盖 fresh start、整窗恢复、部分恢复、超额恢复、零 warmup、`skip.rollout_tq` 以及两个 `on_train_begin` 入口的 top-up 行为。
- `tests/trainer/ppo/v1/test_reissue_inflight_on_cpu.py`（模块 重发测试；类别 test；类型 test-coverage；符号 `test_count_tq_prompt_groups_includes_every_prompt_status`, `test_save_load_terminal_only_counts_prompts_without_reissue`）：补充 `_count_tq_prompt_groups` 的状态无关性测试与 terminal-only 检查点的端到端 save/load 反例，是评审修正的直接回归保护。
- `verl/trainer/ppo/v1/trainer_separate_async.py`（模块 分离训练器；类别 source；类型 core-logic；符号 `PPOTrainerSeparateAsync.on_train_begin`）：`on_train_begin` 同样改为委托基类 helper，删除了重复的 `skip.rollout_tq` 判断与整 batch 循环。

关键符号：`PPOTrainer._add_async_warmup_batches`, `_count_tq_prompt_groups`, `PPOTrainerColocateAsync.on_train_begin`, `PPOTrainerSeparateAsync.on_train_begin`, `PPOTrainer._load_checkpoint`, `PPOTrainer._reissue_inflight_prompts`, `test_save_load_terminal_only_counts_prompts_without_reissue`

关键源码片段

`verl/trainer/ppo/v1/trainer_base.py`

修复主体：新增 `_count_tq_prompt_groups()` 与 `_add_async_warmup_batches()`，在 `__init__` 初始化 `_restored_tq_prompt_count`，在 `_load_checkpoint()` 写入恢复计数，并删除 `fit()` 中已失效的 `SkipManager` 注释。

```
def _count_tq_prompt_groups(partition_id: str = "train") -> int:
    """Number of prompt groups currently registered in a TransferQueue partition."""
    # 关键点：只按 is_prompt 统计，不区分 pending / running / finished / failure,
    # 因为 terminal 状态的 prompt group 同样占用异步流水线的预取窗口
    data = tq.kv_list(partition_id)
```

```

if not data:
    return 0
return sum(1 for tag in data.get(partition_id, {}).values() if tag.get("is_prompt", False))

```

```

class PPOTrainer(ABC):
    def __init__(self, config: DictConfig):
        # ... (其余初始化逻辑略)
        self.local_trigger_step = 0
        # fresh start 或不支持 TQ checkpoint 时保持 0, 等价于旧的「整窗 warmup」行为
        self._restored_tq_prompt_count = 0

    def _add_async_warmup_batches(self, num_warmup_batches: int) -> None:
        """Fill the async prefetch window without duplicating checkpointed prompt groups."""
        # 保留 skip.rollout_tq 短路与边界保护, 语义与旧子类实现一致
        if self.config.skip.rollout_tq.enable or num_warmup_batches <= 0:
            return

        restored_prompts = self._restored_tq_prompt_count
        # 目标窗口统一以 prompt 计数: num_warmup_batches 个 batch * 每 batch 样本数
        target_prompts = num_warmup_batches * self.config.data.train_batch_size
        missing_prompts = max(0, target_prompts - restored_prompts)
        if missing_prompts == 0:
            # 恢复池已填满甚至超出窗口: 只保不缩, 也不额外提交新 prompt
            logger.info(
                f"Skipping async warmup: {restored_prompts} restored prompt groups already fill "
                f"the "
                f"{target_prompts}-prompt prefetch window"
            )
            return

        # 只补差额, 且按精确 prompt 条数提交 (旧实现是按 num_warmup_batches 循环追加整 batch)
        self._add_prompts_to_generate(missing_prompts)
        logger.info(
            f"Added {missing_prompts} warmup prompts after restoring {restored_prompts} of "
            f"{target_prompts} target prompt groups"
        )

    def _load_checkpoint(self):
        # ... (actor / critic / dataloader 恢复略)
        # 5. restore TransferQueue state (async modes). Re-issuing the restored in-flight prompts
        # is
        # deferred to fit() to use the agent_loop_manager.
        if self.trainer_mode != "sync" and _tq_supports_checkpoint():
            tq_ckpt_path = os.path.join(global_step_folder, "transfer_queue")
            if os.path.exists(tq_ckpt_path):
                logger.info(f"Loading TransferQueue state from {tq_ckpt_path}")
                tq.load_checkpoint(tq_ckpt_path)

```

```

# 在 reissue 之前记录「占用窗口」的总数，而不是 reissue 成功数，
# 这样 terminal-only 的 checkpoint 也能正确抑制 warmup
self._restored_tq_prompt_count = _count_tq_prompt_groups()
logger.info(f"Restored {self._restored_tq_prompt_count} training prompt groups
from TransferQueue")

```

verl/trainer/ppo/v1/trainer_colocate_async.py

`on_train_begin` 由自写 warmup 循环改为调用基类 helper，并删除不再使用的 `logging / os` 导入与模块 `logger`，是「逻辑上提」的典型接线改动。

```

# 两个异步训练器的接线改动完全对称：不再各自判断 skip.rollout_tq,
# 也不再循环调用 _add_batch_to_generate(), 而是统一委托给基类做差额补齐。

```

```

class PPOTrainerColocateAsync(PPOTrainer):
    def on_train_begin(self):
        # 基类 helper 内部读取 self._restored_tq_prompt_count,
        # 只提交 target_prompts - restored_prompts 的差额;
        # skip.rollout_tq 的短路逻辑已上提到基类，子类无需重复判断
        self._add_async_warmup_batches(self.config.trainer.v1.colocate_async.num_warmup_
batches)

```

```

class PPOTrainerSeparateAsync(PPOTrainer):
    def on_train_begin(self):
        # 与 colocate 版本一致，只是读取各自模式下的 num_warmup_batches 配置
        self._add_async_warmup_batches(self.config.trainer.v1.separate_async.num_warmup_
batches)

```

tests/trainer/ppo/v1/test_v1_async_warmup_on_cpu.py

新增的针对性单测，覆盖 fresh start、整窗恢复、部分恢复、超额恢复、零 warmup、`skip.rollout_tq` 以及两个 `on_train_begin` 入口的 top-up 行为。

```

def _stub(*, skip_rollout_tq: bool = False, restored_tq_prompt_count: int = 0,
        num_warmup_batches: int = 2, train_batch_size: int = 4):
    # 最小化 trainer stub: 只保留 warmup 门控用到的配置与计数,
    # 并把真实的 _add_async_warmup_batches 绑定到 stub 上 (其余方法用 MagicMock)
    stub = SimpleNamespace(
        config=OmegaConf.create({
            "skip": {"rollout_tq": {"enable": skip_rollout_tq}},
            "data": {"train_batch_size": train_batch_size},
            "trainer": {"v1": {
                "colocate_async": {"num_warmup_batches": num_warmup_batches},
                "separate_async": {"num_warmup_batches": num_warmup_batches},
            }},
        }),
        _restored_tq_prompt_count=restored_tq_prompt_count,
    )
    stub._add_prompts_to_generate = MagicMock()
    stub._add_async_warmup_batches = PPOTrainer._add_async_warmup_batches.__get__(stub)

```

```
return stub
```

```
def test_partial_restored_prompt_pool_is_topped_up():
    stub = _stub(restored_tq_prompt_count=6) # 已恢复 6 个 prompt group
    stub._add_async_warmup_batches(3) # 目标窗口 = 3 * 4 = 12
    stub._add_prompts_to_generate.assert_called_once_with(6) # 只补 6 个，而非整窗 12 个

@requires_tq_checkpoint
def test_save_load_terminal_only_counts_prompts_without_reissue(tq_init, partition_id, tmp_path)
:
    """A terminal-only checkpoint re-issues nothing, yet its groups still fill the prefetch window,
    so warmup must be gated on the prompt count rather than on the re-issue count."""
    # finished + failure 两个 terminal prompt 经 save/load 往返后:
    # 计数为 2 (占用窗口)，而 reissue 返回 0 (不重发) ——这正是 reviewer 指出的反例
    assert trainer_base._count_tq_prompt_groups(partition_id) == 2
    assert stub._reissue_inflight_prompts(partition_id) == 0
    assert stub.agent_loop_manager.batches == []
```

评论区精华

本 PR 的讨论集中在 issue #7759 的评论区 (`comments_count=2`、`review_comments_count=0`)，由 reviewer Begunner 在半途发现原方案漏掉的关键场景，并直接以两个 fix commit (`ead3e17`、`d2f52f3`) 修正后 APPROVED。

Begunner (issue #7759 评论) : "It is a real bug. Thanks for your contribution! Somehow the current fix misses some cases. `_reissue_inflight_prompts()` only counts pending/running prompts; restored finished/failure groups still occupy pipeline slots but return a reissue count of zero, causing warmup overfill."

YeonwooSung (作者回应) : "Thanks for the fixes :)"

争议焦点与结论：最初方案的隐含门控量是 `_reissue_inflight_prompts()` 的返回值，而该函数只重发 `pending / running` 的 prompt；一旦 checkpoint 里只有 terminal (`finished / failure`) prompt，reissue 返回 0，warmup 仍会整窗提交，反而回到溢出状态。结论是门控必须基于「恢复状态占用了多少窗口位」而非「某个动作成功了多少次」，于是引入状态无关的 `_count_tq_prompt_groups()`，并用 `test_save_load_terminal_only_counts_prompts_without_reissue` 把该反例固化下来。

另一处未形成正式讨论但由 PR body 与测试 docstring 明确的设计选择：预取窗口以 prompt 为度量单位 (`num_warmup_batches * train_batch_size`)，恢复池超额时只保不缩 (only log，不额外提交也不裁剪)。

- reissue 返回 0 的 terminal prompt 是否应计入 warmup 门控 (correctness): 改为使用状态无关的 `_count_tq_prompt_groups()` 统计所有 prompt group 占用，并新增 `test_save_load_terminal_only_counts_prompts_without_reissue` 固化该反例；Begunner 随后 APPROVED。

- warmup 目标以 prompt 而非 batch 计量，恢复池超额时只保不缩 (design): 预取窗口统一以 prompt 计数，`_add_async_warmup_batches()` 成为唯一入口；但「1 个恢复 prompt = 1 个窗口位」的等价性未在真实 replay buffer / dataloader 链路上验证。

风险与影响

- 风险:

1. 计数口径缺少 `global_steps` 过滤: `_count_tq_prompt_groups()` 统计分区内所有 `is_prompt=True` 的 tag，不校验其 `global_steps` 是否属于当前恢复窗口。若 checkpoint 时刻 TQ 中残留更早 step 的 prompt group，计数会偏高，warmup 会少提交甚至完全不提交，把「多提交」的 bug 变成「欠提交」。现有测试所有 prompt 都写在同一 `global_steps` 下，未覆盖混合 step。
2. 「terminal prompt 占用窗口」是一个耦合假设：把 finished / failure 计入占用依赖这些恢复出的 prompt 后续仍能被消费；若开启 DAPO 分组过滤（参见 #7792 对 `verl/trainer/ppo/v1/replay_buffer.py` 的改动）或 `max_off_policy_threshold` 丢弃了部分恢复组，窗口实际可用样本会少于计数值，流水线深度不足。
3. 差额补齐的粒度语义待确认: `_add_prompts_to_generate(missing_prompts)` 允许 `missing_prompts` 不是 `train_batch_size` 的整数倍（如恢复 4、目标 12 时提交 8，见 `test_colocate_async_on_train_begin_tops_up_configured_window`），需要确认下游不会按 batch 对齐或触发重新拉取，否则实际窗口仍是整数 batch，差额被放大。
4. 测试链路偏 mock: `test_v1_async_warmup_on_cpu.py` 用 `MagicMock` 替换 `_add_prompts_to_generate`，只断言调用参数，不验证真实 replay buffer / dataloader 消费后的窗口深度；`_count_tq_prompt_groups` 也只经 monkeypatch 注入。
5. 恢复路径的一次性开销: `_count_tq_prompt_groups()` 对分区做一次全量 `kv_list`，大规模 checkpoint 下是 $O(N)$ 元数据遍历；只在 resume 时调用一次，通常可接受，但值得在大规模场景观察。
6. 可观测性: warmup 日志从子类 logger 迁到基类 logger，新增「Skipping async warmup / Added N warmup prompts」两条 INFO 日志，但没有对应 metric 上报，线上判断是否发生欠提交只能靠日志排查。

回归面相对可控: fresh start (`_restored_tq_prompt_count=0`) 与 TQ 不支持 checkpoint (`_tq_supports_checkpoint()` 为 False 时计数保持 0) 都退化为完整 warmup; `skip.rollout_tq.enable` 短路语义不变，且被测试直接覆盖。

- 影响:

- 用户影响: 仅作用于启用 V1 异步训练 (`trainer.v1.trainer_mode=colocate_async / separate_async`)、以 `resume_mode=auto / resume_path` 恢复、且 `TransferQueue >= 0.1.9` 并在 checkpoint 目录含 `transfer_queue` 的场景。修复后恢复运行消耗的 dataloader 行数与未被抢占的运行对齐，不再跳样本，样本效率与实验可复现性提升；`trainer_mode=sync` 或缺少 TQ checkpoint 支持的环境行为完全不变，无配置迁移成本。
- 系统影响: 无新增配置键、无 API 破坏; `_add_batch_to_generate()` → `_add_prompts_to_generate(n)` 的调用方式变化只发生在 warmup 路径，`_reissue_inflight_prompts()` 保持原签名（其返回值不再被用作门控依据）。

- 团队影响: warmup 门控收敛到 PPOTrainer._add_async_warmup_batches(), 两个异步子类不再各写一份循环, 后续新增异步 trainer 模式默认继承修复; 新增的两组 CPU 测试集中在 tests/trainer/ppo/v1/ 下, 便于长期回归。
- 风险标记: 恢复路径门控口径变更, 计数未按 global_steps 过滤, terminal prompt 占位假设未验证, 测试为 mock 链路, 预热日志缺少 metric

关联脉络

- PR #7722 [perf, tool] fix: Avoid overlapping rollout and training profilers in Trainer V1: 同样修改 verl/trainer/ppo/v1/trainer_base.py, 并调整 V1 异步流程的生命周期钩子 (on_train_begin / on_step 时序), 与本 PR 共享同一份 trainer 基类代码路径。
- PR #7792 [trainer, cfg] feat: filter V1 DAPO groups by canonical reward by default: 改动 V1 replay_buffer 的分组过滤 / 采样语义, 直接关系到本 PR 假设「恢复出的 finished/failure prompt 仍会占用窗口并被后续消费」是否成立。
- PR #7764 [ckpt] fix: drain Mooncake completion slots across weight versions: 同属 [ckpt] 范畴的异步训练检查点正确性修复, 反映仓库近期集中加固 async + checkpoint 恢复语义的同一脉络。
- PR #7579 RFC: crash-durable replay buffer (标题按 PR body 描述填写): 本 PR body 明确将其列为排重对象: 前者是 replay buffer 崩溃可恢复性的 RFC, 本 PR 只处理 TQ 恢复成功后的额外 warmup。
- PR #7401 SFT epoch-boundary dataloader (标题按 PR body 描述填写): 本 PR body 明确将其列为排重对象, 说明本 bug 只涉及 V1 异步恢复路径的 dataloader 消费, 不是 SFT epoch 边界问题。