

PR #2823 完整报告

radixark/miles

Fail CI runs on crashed async eval points or abnormal eval results

合并时间: 2026-09-04 21:33

原文链接: <http://prhub.com.cn/radixark/miles/pull/2823>

执行摘要

- 一句话: 集中化 CI 评估失败策略, 修复异步评估点崩溃时 CI 仍通过的漏洞。
- 推荐动作: 此 PR 值得精读。它虽然改动文件不多, 但解决了一个影响 CI 可靠性的核心问题。关键的设计决策——在 `report_eval_skip` 中集中失败策略——非常清晰且易于维护。MetricChecker 的扩展逻辑和测试覆盖也做得很好。对于关注 CI/CD、测试基础设施和模型训练质量门控的工程师来说, 这是一个很好的实践案例。

功能与动机

PR body 和 commit 信息指出, 当前 `EvalDispatcher` 在评估失败时会降级为“跳过”状态, 这对训练是合理的, 但让完全异步评估的 E2E 测试即使所有评估点都崩溃也能通过。崩溃只记录为 `eval/skipped_crashed` 指标和警告, 而 CI 保持绿色。这违背了 PR 的核心保障目标: 确保评估管道实际工作。因此, 需要在 `--ci-test` 模式下强制失败, 以暴露潜在的评估问题。

实现拆解

1. 在核心评估跳过路径集中失败: 在 `miles/ray/rollout/rollout_executor.py` 的 `RolloutExecutor.report_eval_skip` 方法中, 于记录跳过原因之后, 增加了一个条件判断: 若 `self.args.ci_test` 为真, 则抛出 `RuntimeError`。这是最关键且最简练的变更, 它将 CI 失败策略集中到了一个所有评估跳过 (包括崩溃、端口繁忙、导出失败、检查点缺失、后端跳过) 都会经过的公共路径。
2. 增强 MetricChecker 以支持精确计数门控: 在 `miles/utils/metric_checker.py` 中, 将内部状态从布尔值 `_exists_check_success` 改为布尔列表 `_check_results`, 以记录每次检查的结果。`dispose` 方法现在支持两种模式: 默认模式 (无 `expect_num`) 要求至少一次成功; 精确模式 (设置了 `ci_metric_checker_expect_num`) 要求恰好 `expected` 次成功检查且所有检查都通过。同时, 在 `miles/utils/arguments.py` 中新增了 `--ci-metric-checker-expect-num` 命令行参数。
3. 新增和更新单元测试: 新增 `tests/fast/ray/rollout/test_eval_dispatch.py` 测试文件, 验证崩溃评估在 CI 模式下会通过 `report_eval_skip` 路径失败, 而在非 CI 模式下仅记录日志。新增 `tests/fast/utils/test_metric_checker.py` 测试文件, 全面覆盖 MetricChecker 的两种期望模式。在 `tests/fast/rollout/test_checkpoint_eval.py` 中新增 `test_report_eval_skip_fails_ci_after_logging` 参数化测试, 确保所有可归因的评估跳过原因在 CI 模式下都会导致失败, 并修复了该测试文件中 `make_args` 缺失 `ci_test` 字段的问题。

4. 配置 E2E 测试以使用新的精确门控：修改 `tests/e2e/megatron/test_qwen3_4b_fully_async_eval.py` 和 `tests/e2e/short/test_qwen2.5_0.5B_fully_async_eval.py`，在它们的 `ci_args` 中添加 `--ci-metric-checker-key eval/gsm8k --ci-metric-checker-threshold 0.4 --ci-metric-checker-expect-num 3`。这要求两次 E2E 运行都必须恰好完成三次评估，且每次评估的 GSM8K 分数都达到 0.4，从而防止训练前的基线评估单独满足门控，掩盖后续评估的失败。
5. 更新文档：在 `docs/developer/debug.md` 中同步更新了关于准确度门控 (MetricChecker) 的说明，反映了新的精确计数选项。

关键文件：

- `miles/ray/rollout/rollout_executor.py` (模块 评估执行器；类别 `source`；类型 `core-logic`；符号 `report_eval_skip`)：核心变更点。在 `report_eval_skip` 方法中实现了 CI 模式下的集中失败策略，这是整个 PR 的关键逻辑。
- `miles/utils/metric_checker.py` (模块 指标检查器；类别 `source`；类型 `core-logic`；符号 `MetricChecker.init`, `MetricChecker.on_eval`, `MetricChecker.dispose`)：实现了可配置的评估精度门控逻辑，支持“任一次成功”和“精确 N 次全部成功”两种模式，是防止基线评估掩盖问题的核心。
- `miles/utils/arguments.py` (模块 参数配置；类别 `source`；类型 `configuration`)：新增 `--ci-metric-checker-expect-num` 命令行参数，用于控制 MetricChecker 的精确计数模式，是配置层面的关键改动。
- `tests/e2e/megatron/test_qwen3_4b_fully_async_eval.py` (模块 端到端测试；类别 `test`；类型 `test-configuration`)：E2E 测试配置。添加了精确评估计数门控参数，是验证新 CI 策略在真实场景下生效的关键。
- `tests/e2e/short/test_qwen2.5_0.5B_fully_async_eval.py` (模块 端到端测试；类别 `test`；类型 `test-configuration`)：另一个关键的 E2E 测试配置，同样添加了精确评估计数门控参数，确保所有完全异步评估测试都受到新的质量检查。
- `tests/fast/ray/rollout/test_eval_dispatch.py` (模块 评估调度器；类别 `test`；类型 `test-coverage`；符号 `FakeRemoteMethod`, `FakeRolloutManager`, `make_dispatcher`, `TestSettle`)：新增的单元测试文件，专门测试 `EvalDispatcher` 在 CI 模式下处理崩溃评估点的行为，是验证核心失败路径的关键。
- `tests/fast/utils/test_metric_checker.py` (模块 指标检查器；类别 `test`；类型 `test-coverage`；符号 `_make_checker`, `TestDefaultExpectation`, `TestExactExpectation`, `test_no_eval_always_fails`)：新增的单元测试文件，全面测试 MetricChecker 的两种模式（默认和精确），是验证新门控逻辑正确性的关键。
- `tests/fast/rollout/test_checkpoint_eval.py` (模块 检查点评估；类别 `test`；类型 `test-coverage`；符号 `test_report_eval_skip_fails_ci_after_logging`)：更新现有测试文件，添加 `test_report_eval_skip_fails_ci_after_logging` 参数化测试，验证所有评估跳过原因在 CI 模式下都会失败，并修复了测试参数。

关键符号：`RolloutExecutor.report_eval_skip`, `MetricChecker.init`, `MetricChecker.on_eval`, `MetricChecker.dispose`

关键源码片段

miles/utils/metric_checker.py

实现了可配置的评估精度门控逻辑，支持“任一次成功”和“精确 N 次全部成功”两种模式，是防止基线评估掩盖问题的核心。

```
class MetricChecker:
    def __init__(self, args):
        self.args = args
        # 存储每次评估检查的成功 / 失败结果
        self._check_results: list[bool] = []

    def on_eval(self, metrics: dict[str, float]):
        actual_value = metrics.get(self.args.ci_metric_checker_key)
        assert actual_value is not None, f"{metrics=} {self.args.ci_metric_checker_key=}"
        check_success = actual_value >= self.args.ci_metric_checker_threshold
        logger.info(f"[MetricChecker] {check_success=} {actual_value=} {self.args.ci_metric_checker_threshold=}")
        self._check_results.append(check_success)

    def dispose(self):
        # 必须至少有一次评估结果
        assert self._check_results, "[MetricChecker] no metrics checked"
        # 根据配置的期望数量选择门控策略
        if (expected := self.args.ci_metric_checker_expect_num) is None:
            # 默认策略：至少有一次评估成功即可通过
            assert any(self._check_results), f"[MetricChecker] accuracy check failed: {self._check_results=}"
        else:
            # 精确策略：要求恰好 expected 次评估，且全部成功
            assert len(self._check_results) == expected and all(self._check_results), (
                f"[MetricChecker] expected exactly {expected} checks and all to succeed: " f"{self._check_results=}"
            )
        logger.info("[MetricChecker] pass dispose check")
```

tests/fast/ray/rollout/test_eval_dispatch.py

新增的单元测试文件，专门测试 `EvalDispatcher` 在 CI 模式下处理崩溃评估点的行为，是验证核心失败路径的关键。

```
class TestSettle:
    async def test_a_crashed_eval_fails_when_reporting_the_ci_skip(self, tmp_path):
        """A crashed eval fails CI through the centralized skipped-point contract."""
        dispatcher = make_dispatcher(ci_test=True)
        exported_dir = str(tmp_path / "step_3")

        # 模拟一个失败的远程方法（代表崩溃的评估点）
        class FailingRemoteMethod(FakeRemoteMethod):
            def remote(self, *args):
                self.log.append((self.name, args))
                fut = asyncio.get_event_loop().create_future()
```

```

# 模拟抛出与 report_eval_skip 相同的错误格式
fut.set_exception(RuntimeError("CI eval 3 skipped: crashed"))
return fut

dispatcher.rollout_manager.report_eval_skip = FailingRemoteMethod(
    dispatcher.rollout_manager.log, "report_eval_skip"
)

# 验证在 CI 模式下, _settle 会传播 RuntimeError
with pytest.raises(RuntimeError, match="CI eval 3 skipped: crashed"):
    await dispatcher._settle(3, failed_ref(RuntimeError("engine gone")), exported_dir)

# 验证 report_eval_skip 被调用, 且快照目录被保留
assert dispatcher.rollout_manager.log == [("report_eval_skip", (3, "crashed"))]
assert dispatcher._exported == [exported_dir]

```

评论区精华

核心讨论围绕“评估被跳过是否应导致 CI 失败”展开。guapisolo 指出, 如果允许训练前的基线评估满足 `MetricChecker`, 那么后续所有评估点都可以被跳过 (`export_failed`, `ckpt_missing`, `EvalSkip`), 而 CI 仍然会通过, 这违背了 PR 的初衷。yueming-yuan 进一步提问: “是否也应该在评估被跳过时失败 CI?” 作者 fzyzcjy 的最终决策是: 在 CI 模式下, 所有评估跳过都必须失败 CI (通过集中化的 `report_eval_skip` 路径), 并且通过 `--ci-metric-checker-expect-num` 精确控制期望的成功评估次数, 以确保评估管道完整运行。讨论中还涉及了对 `EvalDispatcher.drain()` 方法是否会吞没异常的分析, 结论是该行为符合 CI 快速失败的预期。

- 评估跳过是否应导致 CI 失败 (correctness): 作者最终决定, 在 CI 模式下, 所有评估跳过 (包括崩溃、各种失败原因) 都必须导致 CI 失败, 并通过在 `report_eval_skip` 中集中实现来确保这一点。同时, 通过 `--ci-metric-checker-expect-num` 精确控制期望的成功评估次数, 确保评估管道完整运行。
- `report_eval_skip` 的异常是否会被 `EvalDispatcher.drain()` 吞掉 (correctness): 分析认为该异常会正常传播, 导致跳过 `dispose.remote()` 并让进程以非零状态退出。这被视为符合 PR 目标 (在评估问题上让 CI 快速失败) 的预期行为, 而非缺陷。

风险与影响

- 风险:
 1. CI 稳定性风险: 将 `--ci-test` 模式下的评估跳过行为从“记录并继续”改为“立即失败”是一个显著的行为变更。如果某些评估点在 CI 环境中因外部因素 (如瞬时资源争用) 偶尔跳过, 可能导致 CI 频繁失败, 增加维护负担。然而, PR 作者通过设定较宽松的阈值 (0.4) 和精确计数 (3 次), 试图在可靠性和敏感性之间取得平衡。
 2. `MetricChecker` 逻辑正确性: `MetricChecker` 从“任一次成功即通过”变为可配置的“精确 N 次全部成功”。需要确保新逻辑能正确处理零评估情况 (已有测试覆盖) 和边界条件 (如评估次数不匹配)。测试覆盖看起来比较全面。

3. E2E 测试硬编码期望值：E2E 测试中的 `--ci-metric-checker-expect-num 3` 是一个硬编码的期望值。如果未来测试的评估次数安排发生变化（例如，由于配置调整），需要手动更新此值，否则会导致 CI 失败或失去门控效果。这是一个维护点，但在此 PR 的上下文中是合理的。

- 影响：

1. 对 CI 系统的影响：此变更直接提升了 CI 对评估管道健康状况的检测能力。任何评估崩溃、跳过或未达到预期精度的运行都将导致 CI 失败。这可能会在短期内暴露一些之前被掩盖的、间歇性的评估问题，但长期来看会增强对模型训练后质量的信心。
2. 对开发者工作流的影响：开发者在提交 PR 时，需要确保其修改不会导致评估点崩溃或评估结果低于阈值（0.4），否则 CI 将失败。这增加了提交的质量要求，但也提供了更早的反馈。
3. 对系统整体质量的影响：这是一个积极的变更，它强化了训练 - 评估循环的完整性。通过确保评估点不仅被调度，而且成功执行并产生符合标准的结果，减少了交付“评估看似正常但实际失败”的模型的风险。 - 风险标记：CI 行为显著变更，硬编码的 E2E 期望值

关联脉络

- 暂无明显关联 PR