PR #2576 完整报告

radixark/miles

fix: skip rollout construction for debug replay

合并时间：2026-08-18 17:58

原文链接：http://prhub.com.cn/radixark/miles/pull/2576

执行摘要

- 一句话：调试回放跳过 rollout 构造，修复崩溃与 FT 误启动
- 推荐动作：值得精读，核心是“构造门控”与“生命周期门控”两个概念的分离：用 load_debug_rollout_data 控制是否构造 rollout fn，用 _rollout_ft_enabled 控制是否启动 rollout 侧的生命周期工作。关注作者对门控选择的对抗性论证（为什么不是 debug_train_only），以及 EvalFleet.pin 中探测委派的语义。建议后续为 RolloutManager 增加 generate_rollout is None 的显式类型保护，避免 None 静默传播。

功能与动机

PR body 指出 release 运行在 stage-c-8-gpu-h200 上因 GenerateState 报 TypeError: unsupported operand type(s) for *: int and NoneType 崩溃；根因是 #2522 让 class-based rollout 成为默认后，RolloutManager.__init__急切构造 InferenceRolloutFn，而 replay 运行（--load-debug-rollout-data，强制 debug_train_only）故意留空 rollout_num_gpus。独立验证评论也确认了同一回归链：GenerateState.__init__ 用 args.sglang_server_concurrency * args.rollout_num_gpus // args.rollout_num_gpus_per_engine 计算信号量，rollout_num_gpus=None 直接触发 TypeError。此外，train-only FT（--ft-components train）仍激活 rollout 生命周期，导致确定性比较出现 10 个 MetricEvent 偏差。

实现拆解

1. 回放构造门控：在 miles/ray/rollout/rollout_manager.py 的 RolloutManager.__init__ 中，把非 legacy 分支改为先判断 args.load_debug_rollout_data is not None；是则直接令 generate_rollout 和 eval_generate_rollout 为 None，不再调用 load_rollout_function，避免构造 GenerateState。同时把 _get_rollout_data 的判定从真值判断收紧为 is not None，保证语义一致。
2. FT 生命周期统一门控：新增实例属性 self._rollout_ft_enabled = self.args.use_fault_tolerance and "rollout" in self.args.ft_components，并将健康监控启动、_ci_fault_injection_pending 初始化、generate() 中的 CI 故障注入条件都改用它；_ci_fault_injection_pending 现在始终先初始化，再在启用 rollout FT 时置为 args.ci_test。
3. EvalFleet 探测委派：在 miles/ray/rollout/eval_fleet.py 的 EvalFleet 中同步计算 _rollout_ft_enabled，用于 pin() 决定是否自行 probe_and_mark_dead()——train-only FT 没有 RolloutHealthMonitor，必须自己探测。

4. 训练端恢复收口：在 miles/ray/actor_group.py 的 RayTrainGroup.update_weights() 中，把 recover_updatable_engines 的调用条件从仅 use_fault_tolerance 收紧为同时要求 "rollout" in ft_components，避免 train-only FT 恢复 rollout 引擎。
5. 测试配套：tests/fast/ray/rollout/real_ray/test_rollout_manager.py 新增参数化测试验证 monitor 数量与故障注入 pending 跟随组件选择，并新增 replay 跳过构造 / train-only 正常构造两个用例；tests/fast/ray/test_actor_group_shared_ppo.py 新增 _RemoteCall 辅助类和 test_train_only_ft_does_not_recover_rollout_engines；tests/fast/ray/rollout/test_eval_fleet.py 新增 test_fleet_probes_with_train_only_fault_tolerance，并调整既有用例的 ft_components 设置。

关键文件：

- miles/ray/rollout/rollout_manager.py（模块 回放管理；类别 source；类型 core-logic；符号 RolloutManager.init, RolloutManager._get_rollout_data, RolloutManager.generate）：核心修复文件：在 __init__中按 load_debug_rollout_data 跳过 class-based rollout 构造，避免 GenerateState 崩溃；并引入 _rollout_ft_enabled 统一门控健康监控、CI 故障注入与 _get_rollout_data 分支语义。
- tests/fast/ray/rollout/real_ray/test_rollout_manager.py（模块 回放测试；类别 test；类型 test-coverage；符号 test_rollout_ft_lifecycle_follows_selected_component, FakeMonitor, test_debug_rollout_replay_skips_class_based_rollout_construction, test_debug_train_only_without_replay_constructs_rollout_function）：覆盖三类回归场景：replay 跳过构造、train-only 不启动 monitor、rollout 组件启用 monitor 与故障注入；是门控行为的主要测试依据。
- miles/ray/rollout/eval_fleet.py（模块 评估集群；类别 source；类型 core-logic；符号 EvalFleet.init, EvalFleet.pin）：EvalFleet.pin 的探测委派逻辑随 _rollout_ft_enabled 调整，train-only FT 下必须自行 probe，否则引擎健康无法保证。
- miles/ray/actor_group.py（模块 训练组；类别 source；类型 core-logic；符号 RayTrainGroup.update_weights）：RayTrainGroup.update_weights 的引擎恢复条件收紧，避免 train-only FT 恢复 rollout 引擎，是生命周期门控在训练侧的关键落点。
- tests/fast/ray/rollout/test_eval_fleet.py（模块 评估测试；类别 test；类型 test-coverage；符号 test_fleet_probes_with_train_only_fault_tolerance）：新增 train-only FT 下 EvalFleet 自行探测的用例，并修正既有用例的 ft_components 设置。
- tests/fast/ray/test_actor_group_shared_ppo.py（模块 训练组测试；类别 test；类型 test-coverage；符号 _RemoteCall, test_train_only_ft_does_not_recover_rollout_engines）：新增 _RemoteCall 辅助类和 train-only FT 不恢复 rollout 引擎的用例，验证 update_weights 门控。

关键符号：RolloutManager.init, RolloutManager._get_rollout_data, RolloutManager.generate, EvalFleet.init, EvalFleet.pin, RayTrainGroup.update_weights, test_rollout_ft_lifecycle_follows_selected_component, test_debug_rollout_replay_skips_class_based_rollout_construction, test_train_only_ft_does_not_recover_rollout_engines, test_fleet_probes_with_train_only_fault_tolerance

关键源码片段

miles/ray/rollout/rollout_manager.py

核心修复文件：在 __init__中按 load_debug_rollout_data 跳过 class-based rollout 构造，避免 GenerateState 崩溃；并引入 _rollout_ft_enabled 统一门控健康监控、CI 故障注入与 _get_rollout_data 分支语义。

```
# 回放模式下完全没有真实 rollout 函数调用，因此：
# - 若配置了 --load-debug-rollout-data，直接跳过 class-based rollout 构造，
# 避免 GenerateState 因 rollout_num_gpus=None 崩溃；
# - 否则才加载 rollout 函数（train-only SFT 也依赖它产出数据）。
self.use_legacy_rollout_v1 = use_legacy_rollout_v1()
if not self.use_legacy_rollout_v1:
    if self.args.load_debug_rollout_data is not None:
        self.generate_rollout = None
        self.eval_generate_rollout = None
    else:
        input = RolloutFnConstructorInput(args=args, data_source=self.data_source)
        self.generate_rollout = load_rollout_function(input, self.args.rollout_function_path)
        if self.args.eval_function_path == self.args.rollout_function_path:
            # eval 与 train 共享同一个有状态实例，避免 FullyAsyncRolloutFn 被构造两次
            self.eval_generate_rollout = self.generate_rollout
        else:
            self.eval_generate_rollout = load_rollout_function(input, self.args.eval_function_path)
else:
    self.generate_rollout = load_function(self.args.rollout_function_path)
    self.eval_generate_rollout = load_function(self.args.eval_function_path)
#  （custom reward / convert 函数加载已省略）

# rollout FT 生命周期只在显式勾选 rollout 组件时激活：
# train-only FT 不应启动健康监控、故障注入或引擎恢复。
self._rollout_ft_enabled = self.args.use_fault_tolerance and "rollout" in self.args.ft_components
self._ci_fault_injection_pending = False
if not self.args.debug_train_only and self._rollout_ft_enabled:
    for srv in self.servers.values():
        for group in srv.server_groups:
            monitor = RolloutHealthMonitor(group, args)
            monitor.start()
            self._health_monitors.append(monitor)
    self._ci_fault_injection_pending = self.args.ci_test
```

miles/ray/rollout/eval_fleet.py

EvalFleet.pin 的探测委派逻辑随 _rollout_ft_enabled 调整，train-only FT 下必须自行 probe，否则引擎健康无法保证。

```
class EvalFleet:
    """专用 in-job 评估引擎（--eval-num-gpus）的权重交付与健康管理。"""

    def __init__(self, args: Namespace, *, srv):
```

```
        self.args = args
        self._srv = srv
        # 只有当 rollout 组件启用故障容忍时，RolloutHealthMonitor 才会负责探测；
        # train-only FT 下没有 monitor，EvalFleet 必须自己 probe。
        self._rollout_ft_enabled = args.use_fault_tolerance and "rollout" in args.ft_components
        self._state = GenerateState(self._fleet_args())

    async def pin(self, checkpoint_dir: str, weight_version: str) -> GenerateState:
        """把快照加载到每个评估引擎，并返回可生成的状态。

        在 manager 事件循环上运行，全程 await 而非阻塞。
        """
        try:
            if not self._rollout_ft_enabled:
                # 没有 monitor 接管时，先探测并标记死引擎，再恢复，再等待全部存活
                await self._srv.probe_and_mark_dead()
            await self._srv.recover()
            await self._srv.wait_all_engines_alive()
        except Exception as e:
            logger.warning(f"Eval fleet unhealthy: {e}")
            raise EvalSkip("unhealthy") from e

        if not await self._pin_fleet(checkpoint_dir, weight_version):
            raise EvalSkip("pin_violation")

        try:
            await self._wait_router_ready()
        except Exception as e:
            logger.warning(f"Eval router not ready: {e}")
            raise EvalSkip("unhealthy") from e

        return self._state
```

评论区精华

作者 guapisolo 在 issue 评论中给出了完整回归链：#2522 使 class-based rollout fn 成为默认后，RolloutManager.__init__ 急切构造 InferenceRolloutFn，replay 又故意留空 rollout_num_gpus，于是构造期即崩溃；并解释了为什么 load_debug_rollout_data 是正确的门控而 debug_train_only 是错的：独立 SFT 运行（如 scripts/run_qwen3_sft.py、examples/geo3k_vlm/run_geo3k_vlm_sft.sh）把 rollout fn 当作实际数据生产者，_get_rollout_data 只绕过 generate_rollout 而不绕过构造。

CI 验证过程被多次更新：初始未验证 → 推送修复后开始验证 → 最终主栈未经 override 的情况下 28 passed / 2 skipped / 0 failed，之前失败的 test_trainer_ft_deterministic_dp2_cp2_real_rollout.py 通过。另有独立验证从 release-cut 失败出发、未参考本 PR 即收敛到同一 gate。

reviewer fzyzcjy 批准："LGTM for trainer-ft related changes, since I refactor a lot in https://github.com/radixark/miles/issues/1837"。

- 回放门控选择：load_debug_rollout_data 而非 debug_train_only (design)：采用 load_debug_rollout_data 作为构造门控，普通 train-only 仍构造 rollout fn。
- train-only FT 不应激活 rollout 生命周期 (correctness)：引入 _rollout_ft_enabled 统一门控，monitor、故障注入、恢复均受控。
- CI 验证状态确认 (testing)：验证完成，PR 可合并。
- 独立验证结论一致 (question)：确认修复方向正确。
- trainer-ft 相关改动批准 (other)：批准本 PR。

风险与影响

- 风险：核心路径变更：RolloutManager.__init__ 是所有训练 / 回放共用的初始化路径，门控改动影响面广；若未来新增调用方直接使用 generate_rollout 而忘记检查回放标志，会触发 None 调用错误。

配置语义收紧：load_debug_rollout_data 从真值判断改为 is not None，若历史配置传空字符串或 0（非 None 但 falsy），行为会反转；不过该参数预期是路径字符串，实际风险低。

FT 组件依赖：_rollout_ft_enabled 依赖 args.ft_components 为有效可迭代且包含 "rollout"；若外部配置未正确初始化ft_components（如None），"rollout" in None会抛TypeError。需要确认参数解析的默认值。

train-only FT 下 EvalFleet 自行 probe 会增加一次探测开销，但每次 pin 才一次，影响很小；RayTrainGroup.update_weights 不再在 train-only FT 时恢复 rollout 引擎，若训练端依赖引擎预先恢复可能漏恢复，但 train-only FT 语义上本就不应触碰 rollout。

现有测试未覆盖 legacy rollout v1 路径，只能依赖环境变量门控保证。

- 影响：对用户：使用 --load-debug-rollout-data 的调试回放不再崩溃；使用 --ft-components train 的确定性 FT 测试不再被 rollout 生命周期污染，MetricEvent 对比可通过。

对系统：修复了 release-branch-cut 的阻塞问题，作者明确表示合并此 PR 是重切 release 的前提。

对团队：FT CI 稳定收敛，后续可在干净的组件门控上继续推进完整 FT 能力；测试新增三组用例，为回放与 train-only 行为提供了长期防护网。

- 风险标记：核心路径变更，配置语义收紧，FT 组件依赖新字段，缺少 legacy 路径测试

关联脉络

- PR #2522 Make the class-based rollout the default and convert legacy path to env var gated: 回归源头：该 PR 使 RolloutManager.__init__急切构造 class-based rollout fn，从而在 replay 时触发 GenerateState TypeError。