PR #6572 完整报告

verl-project/verl

[rollout, reward] feat: add full determinism support for vLLM rollout and reward model

合并时间：2026-06-23 13:25

原文链接：http://prhub.com.cn/verl-project/verl/pull/6572

执行摘要

- 一句话：为 vLLM rollout 和 reward model 增加完全确定性支持，实现端到端可重现训练
- 推荐动作：该 PR 值得精读，尤其是以下几点设计决策：1）确定性路由使用 hash(request_id) 而非 random 选择，确保跨运行时一致；2）以 priority 作为 request_id 的基础，避免 uuid4() 引入的随机性；3）RM 序列化作为 vLLM classify 路径不支持的妥协方案。建议团队在文档中明确说明确定性训练的限制（仅 single-turn Agent Loop、RM max_num_seqs=1、性能开销大），并在启用时使用专用的配置 profile。

功能与动机

训练引擎（FSDP 等）已通过 EngineConfig.full_determinism 支持可重现运行，但 rollout 引擎（vLLM/SGLang 等）缺乏确定性控制，RolloutConfig 没有 full_determinism 和 seed 字段，vLLM 服务器 seed 默认使用 replica_rank+0 的隐式回退，且从未对 rollout worker 调用 enable_full_determinism，导致端到端训练不可重现（来自 Issue #6570）。

实现拆解

1. 配置定义：在 verl/workers/config/rollout.py 的 RolloutConfig 中新增 full_determinism: bool = False 和 seed: int = 42 字段；在对应的 rollout.yaml 和 reward.yaml 中添加默认值。
2. 环境变量传播：在 verl/trainer/main_ppo.py 的 run_ppo 中根据配置设置 PYTHONHASHSEED、VERL_FULL_DETERMINISM、VLLM_BATCH_INVARIANT，并通过 PPO_RAY_RUNTIME_ENV 传递给所有 Ray actor（修改 verl/trainer/constants_ppo.py）。确保哈希和 vLLM 批不变性在子进程中生效。
3. 确定性初始化：在 verl/workers/rollout/vllm_rollout/vllm_async_server.py 中，当 full_determinism=True 时调用 enable_full_determinism(seed)，并导出 VERL_FULL_DETERMINISM、VERL_SEED 供 vLLM worker 子进程继承（通过 utils.py 的 vLLMColocateWorkerExtension）。
4. 请求级种子与优先级：在 generate() 方法中向 SamplingParams 注入 seed = replica_rank + config.seed，确保每次生成的 RNG 状态重置。同时通过 priority 参数传递排序信息（verl/experimental/agent_loop/single_turn_agent_loop.py），并将 request_id 设为 f"det-{priority}" 替代 uuid4()，保证路由一致性。

5. 确定性路由与负载均衡：在 verl/workers/rollout/llm_server.py 的 GlobalRequestLoadBalancer.acquire_server 中，当 full_determinism=True 且存在多个等负载候选时，使用 hash(request_id) % len(candidates) 进行平局决胜，确保同一请求始终路由到同一副本。
6. RM 序列化：在 verl/experimental/reward_loop/reward_model.py 中，当 full_determinism=True 时强制 max_num_seqs=1，因为 vLLM 的 /classify 路径不支持 priority 调度和 batch invariance，只能通过序列化推理保证 RM 分数 bitwise 可重现。
7. 测试验证：新增 test_vllm_generation_determinism.py（三个测试：相同实例、跨实例 vLLM、跨实例 AgentLoop）和 run_determinism_e2e_with_rm.py（运行两次 PPO 训练并验证 reward 曲线 bitwise 一致）。

关键文件：

- verl/workers/rollout/llm_server.py（模块 请求路由；类别 source；类型 core-logic；符号 GlobalRequestLoadBalancer.init, GlobalRequestLoadBalancer.acquire_server）：在 GlobalRequestLoadBalancer 中实现确定性路由，通过 hash(request_id) 进行平局决胜，是跨实例确定性的关键
- verl/experimental/agent_loop/single_turn_agent_loop.py（模块 Agent 循环；类别 source；类型 core-logic；符号 SingleTurnAgentLoop.run）：在 run 方法中用基于 priority 的确定性 request_id 替换 uuid4()，确保多副本场景下请求路由一致
- verl/experimental/reward_loop/reward_model.py（模块 奖励模型；类别 source；类型 data-contract；符号 RewardModelManager.init）：强制 max_num_seqs=1 以序列化 RM 推理，避免 vLLM classify 路径无法保证确定性的问题
- verl/workers/rollout/vllm_rollout/vllm_async_server.py（模块 Rollout 服务器；类别 source；类型 dependency-wiring；符号 VLLMHttpServer.init, VLLMHttpServer.generate）：在 vLLM 服务器初始化时调用 enable_full_determinism 并设置环境变量，同时注入 per-request seed
- verl/trainer/main_ppo.py（模块 训练入口；类别 source；类型 core-logic；符号 run_ppo）：在 run_ppo 中从配置读取 full_determinism 和 seed，并设置相关环境变量以确保传播到所有 Ray actor
- tests/experimental/reward_loop/run_determinism_e2e_with_rm.py（模块 E2E 测试；类别 test；类型 test-coverage；符号 run_training, read_metrics_from_jsonl, _pick_reward_key, extract_reward_curve）：新增 E2E 训练确定性验证脚本，运行两次 PPO 训练并比较 reward 曲线是否 bitwise 一致，并生成对比图
- tests/workers/rollout/rollout_vllm/test_vllm_generation_determinism.py（模块 Rollout 测试；类别 test；类型 test-coverage；符号 _get_config_dir, _pearson_correlation, _make_rollout_config, _make_e2e_config）：新增三个 rollout 确定性测试：相同实例、跨实例 vLLM、跨实例 AgentLoop，验证 logprob 的 Pearson 相关系数
- verl/workers/config/rollout.py（模块 Rollout 配置；类别 source；类型 core-logic；符号 RolloutConfig）：在 RolloutConfig 中添加 full_determinism 和 seed 字段，作为配置入口

关键符号：enable_full_determinism, GlobalRequestLoadBalancer.acquire_server, SingleTurnAgentLoop.run, RewardModelManager.init, VLLMHttpServer.init,

VLLMHttpServer.generate, run_ppo, run_training, _compare_logprobs, verify_bitwise_alignment

关键源码片段

verl/workers/rollout/llm_server.py

在 GlobalRequestLoadBalancer 中实现确定性路由，通过 hash(request_id) 进行平局决胜，是跨实例确定性的关键

```
# verl/workers/rollout/llm_server.py
# GlobalRequestLoadBalancer.acquire_server 的确定性路由逻辑

class GlobalRequestLoadBalancer:
    def acquire_server(self, request_id: str) -> tuple[str, ray.actor.ActorHandle]:
        # 先尝试粘滞会话缓存
        if request_id in self._request_id_to_server:
            server_id = self._request_id_to_server[request_id]
            if server_id in self._inflight_requests:
                self._inflight_requests[server_id] += 1
                return server_id, self._servers[server_id]
            del self._request_id_to_server[request_id]

        # 无可用服务器时出错
        if not self._inflight_requests:
            raise RuntimeError("No available servers in load balancer")

        # 找到最小负载的副本集合
        min_count = min(self._inflight_requests.values())
        candidates = [sid for sid, count in self._inflight_requests.items() if count == min_count]

        if len(candidates) == 1:
            server_id = candidates[0]
        elif self._full_determinism:
            # 确定性平局决胜：相同 request_id 永远选同一副本
            server_id = candidates[hash(request_id) % len(candidates)]
        else:
            # 非确定性时直接取第一个（仍比随机更稳定）
            server_id = candidates[0]

        self._request_id_to_server[request_id] = server_id
        self._inflight_requests[server_id] += 1
        return server_id, self._servers[server_id]
```

verl/experimental/agent_loop/single_turn_agent_loop.py

在 run 方法中用基于 priority 的确定性 request_id 替换 uuid4()，确保多副本场景下请求路由一致

```
# verl/experimental/agent_loop/single_turn_agent_loop.py
# SingleTurnAgentLoop.run 的 request_id 确定性生成逻辑
```

```
class SingleTurnAgentLoop(AgentLoopBase):
    async def run(self, sampling_params: dict[str, Any], priority: int = 0, **kwargs) ->
    AgentLoopOutput:
        # priority 可能来自非张量 batch，统一转为 Python int
        priority = int(priority)
        messages = list(kwargs["raw_prompt"])
        # ... 省略多模态处理和 tokenize ...

        # 生成阶段：根据是否启用确定性选择 request_id 生成方式
        with simple_timer("generate_sequences", metrics):
            if getattr(self.rollout_config, "full_determinism", False):
                # 确定性模式：以 priority 为基础，保证相同优先级请求 id 跨运行时一致
                request_id = f"det-{priority}"
            else:
                request_id = uuid4().hex

            output: TokenOutput = await self.server_manager.generate(
                request_id=request_id,
                prompt_ids=prompt_ids,
                sampling_params=sampling_params,
                priority=priority, # 传递优先级给后端服务器
                # ... 省略其他参数 ...
            )
        # ... 省略后续处理 ...
```

verl/experimental/reward_loop/reward_model.py

强制 max_num_seqs=1 以序列化 RM 推理，避免 vLLM classify 路径无法保证确定性的问题

```
# verl/experimental/reward_loop/reward_model.py
# RewardModelManager 初始化中的确定性序列化逻辑

class RewardModelManager:
    def __init__(self, config, resource_pool):
        self.config = config
        self.resource_pool = resource_pool

        # vLLM /classify（pooling 路径）不支持 priority 调度和 VLLM_BATCH_INVARIANT，
        # 因此 co-batched RM 前向会破坏 bitwise 可重现性。
        # 作为 workaround，当 full_determinism 开启时强制 max_num_seqs=1，
        # 让 RM 推理完全序列化。
        if self.config.rollout.full_determinism and self.config.rollout.max_num_seqs != 1:
            logger.warning(
                "[reward_model] full_determinism=True: forcing rollout.max_num_seqs "
                "from %d to 1. vLLM pooling/classify does not support priority "
                "scheduling or batch invariance; serializing RM inference is "
                "currently the only way to keep RM scores bitwise reproducible.",
                self.config.rollout.max_num_seqs,
            )
            self.config.rollout.max_num_seqs = 1
```

```
        self._initialize_llm_servers()
        self._initialize_router()
```

评论区精华

1. 环境变量传播不足：*gemini-code-assist*[bot] 指出仅 *enable_full_determinism* 在父进程调用不够，因为 vLLM worker 是子进程，不会继承 PyTorch/C++ 确定性设置。建议设置 VERL_FULL_DETERMINISM 和 VERL_SEED 环境变量。作者在 *vllm_async_server.py* 和 *utils.py* 中添加了环境变量导出和子进程继承逻辑。
2. PYTHONHASHSEED 静态快照问题：Luosuu 指出 *constants_ppo.py* 中 PYTHONHASHSEED 是导入时读取的静态值，若 *main_ppo.py* 后续才设置环境变量，Ray runtime_env 中仍然是旧值。作者修复为在 *run_ppo* 中动态设置环境变量后再启动 Ray。
3. AgentLoop API 修改：*wuxibin89* 建议不要修改 AgentLoopWorker 的 API（增加 global_priority_offset 参数），而是像 uid 一样将 priority 放在 batch 的 DataProto 中。作者采纳，改为通过 batch 传递 priority。
4. *image_data* 参数被误删除：*wuxibin89* 发现 *single_turn_agent_loop.py* 中移除了 *image_data*、*video_data* 等参数。作者确认是误删并立即修复。
5. *request_id* 确定性修复：Luosuu 询问 uuid4() 是否影响确定性。作者最初认为单副本不影响，后确认多副本场景下 hash(*request_id*) 会因 *request_id* 随机而破坏平局决胜，因此改用基于 priority 的 *det-{priority}* 格式，仅在 *full_determinism=True* 时生效。
6. RM 序列化必要性：Luosuu 问 *reward_loop.py* 中 priority 注入是否有效，作者解释当前使用 *max_num_seqs=1* 保证 RM 确定性，因为 vLLM *classify* 路径不支持 priority 和 batch invariance。
7. 仅支持 single-turn：*wuxibin89* 要求明确文档指出 *full_determinism* 仅支持 single-turn Agent Loop，作者已补充。

- 环境变量传播不足以实现 worker 进程确定性 (*correctness*): 作者在 *vllm_async_server.py* 中添加了环境变量 *export*，并在 *utils.py* 中让子进程动态调用 *enable_full_determinism*
- PYTHONHASHSEED 在 *constants_ppo.py* 中为静态快照导致传播失效 (*correctness*): 作者修复：不在 *constants_ppo.py* 中硬编码，而是由 *main_ppo.run_ppo* 在启动 Ray 前动态设置环境变量
- AgentLoop API 应避免修改 *generate_sequences* 签名 (*design*): 作者采纳，改为通过 batch 传递 priority，不修改 *generate_sequences* 签名
- *image_data* 和 *video_data* 等参数被误删除 (*correctness*): 作者确认为误删并立即修复
- uuid4() 导致确定性路由在多副本场景下失效 (*correctness*): 作者将 *request_id* 改为基于 priority 的 *det-{priority}* 格式（*full_determinism=True* 时），确保相同请求 id 跨运行时一致
- RM 确定性需要序列化推理而非 priority (*design*): 作者保留 *max_num_seqs=1* 方案，移除 *reward_loop.py* 中 priority 注入，并在 *reward_model.py* 中添加强制覆盖逻辑
- *full_determinism* 仅支持 single-turn Agent Loop 需文档说明 (*documentation*): 作者补充了文档说明

风险与影响

- 风险：性能退化：开启 full_determinism 后，VLLM_BATCH_INVARIANT=1 会禁用 vLLM 的批处理优化；priority 调度可能降低服务器吞吐；RM 强制 max_num_seqs=1 导致推理序列化，吞吐显著下降。故该功能仅建议在调试或回归测试中启用。 环境变量传播遗漏：PYTHONHASHSEED 必须在 Ray 启动前设置，若使用自定义启动器（如 srun）可能未正确设置，导致确定性失效。VERL_FULL_DETERMINISM 和 VERL_SEED 通过 Ray runtime_env 传播，但需确保所有 actor 创建时已生效。 多副本 request_id 唯一性：确定性 request_id 依赖 priority 的全局唯一性，若 priority 重复或未正确传递，可能导致路由冲突和死锁。当前实现由 AgentLoopManager.generate_sequences 按 sample_index 生成优先级，在多副本场景下通过 global_priority_offset 保证唯一。 vLLM 版本依赖：VLLM_BATCH_INVARIANT 需要 vLLM 版本 ≥ 0.4.3（实际 commit 中写在 env name 中），早期版本忽略此环境变量，无法保证批不变性。 Agent Loop 限制：当前确定性仅实现在 SingleTurnAgentLoop，未覆盖 ToolAgentLoop 等多轮循环，文档已说明限制。
- 影响：用户影响：用户可在配置中设置 actor_rollout_ref.rollout.full_determinism=true 和 seed 来启用确定性训练，配合已存在的 actor.fsdp_config.full_determinism 实现端到端可重现。默认配置下行为完全不变。 系统影响：开启后 vLLM 服务器吞吐下降 1~2 个数量级（尤其是 RM 序列化），但获得 bitwise 可重现性。其他后端（SGLang、TRT-LLM）未改动，不受影响。 团队影响：需维护五层确定性实现的协同：PyTorch 层、环境变量层、vLLM 批不变层、request seed 层、priority 路由与 RM 序列化层。任何一层被绕过或修改都会破坏整体确定性。新增的测试脚本（3 个单元测试 + 1 个 E2E 脚本）为 CI 提供了确定性回归检测。
- 风险标记：环境变量传播遗漏风险，RM 序列化性能开销，多副本 request_id 唯一性依赖 priority，vLLM batch invariance 版本兼容性，仅支持 single-turn Agent Loop 确定性

关联脉络

- PR #6779 [rollout] feat: add Continuous Token for Agentic Rollout: 修改了相同的 agent_loop 文件（agent_loop.py、single_turn_agent_loop.py），与本 PR 在 AgentLoop 的重叠区域有代码交互
- PR #6790 [trainer] feat: A runnable separate async trainer: 修改了 verl/workers/rollout/llm_server.py，与本 PR 在 GlobalRequestLoadBalancer 的改动可能导致合并冲突或功能协同