PR #7027 完整报告

verl-project/verl

[reward] feat: support deterministic reward for user-defined generative RM paths

合并时间：2026-08-11 12:59

原文链接：http://prhub.com.cn/verl-project/verl/pull/7027

执行摘要

- 一句话：补齐奖励路径全确定性，生成式 RM 按位可复现
- 推荐动作：值得精读。四个值得关注的设计决策：(1) 将 Ray actor 从装饰器形式改为 " 普通类 + 实例化时 ray.remote 包装 "，让框架组件可以被继承覆写，是给核心类开扩展点的通用手法；(2) 用 " 提交序 vs 完成序 " 解释 V0/V1 trainer 在确定性上的根本差异——异步流水线里任何依赖完成序的 concat 都无法用 seed 修复，这是设计异步训练引擎的重要教训；(3) flash-attn Triton 交叉熵内核不受 torch.use_deterministic_algorithms 约束这一细节，以及用 env 开关强制降级纯 PyTorch 路径的方案；(4) 路由确定性在 rollout 侧（hash + PYTHONHASHSEED，依赖 env 传播）与 RM 侧（crc32，平台无关）采用两套方案的取舍。

功能与动机

PR body 明确目标："Makes the reward path bitwise-deterministic across two identical PPO runs, closing the remaining gaps for generative RM (GRM) determinism"。此前 verl 已有四层确定性（PyTorch seeds、env 传播、vLLM batch invariance + per-request seed、full_determinism 标志），但 reward 路径仍有两个未被覆盖的缺口：一是浮点确定性——FLASH_ATTENTION_DETERMINISTIC 只约束 attention backward，flash-attn 的 Triton 交叉熵内核与 cuBLAS matmul、NCCL all-reduce 均不受 torch.use_deterministic_algorithms 控制（Triton 自定义 op 不触发 warn_only）；二是路由与串行策略——RM max_num_seqs 被无条件强制为 1，NaiveRouter 没有确定性 tie-break，多副本 reward 路由依赖到达时序。Issue #7216 给出直接证据：E2E 确定性测试从 step 2 起 diff 由 5.96e-08 放大到 0.337，期望 " 如 PR #6572 一样 bitwise-aligned"；verl-omni RFC #111 则列出 GenRM 采样参数硬编码、无浮点确定性等缺口。

实现拆解

第 1 步：浮点确定性原语（必须在 torch/NCCL 初始化前生效）

- verl/workers/engine/utils.py：*enable_full_determinism(seed)* 新增 NCCL_DETERMINISTIC=1、NCCL_ALGO=Ring、NCCL_PROTO=Simple，并用 os.environ.setdefault("VERL_DISABLE_FLASH_ATTN_CE", "1") 关闭 flash-attn 交叉熵内核（setdefault 保留用户显式覆盖）。
- verl/utils/torch_functional.py：logprobs_from_logits 在 VERL_DISABLE_FLASH_ATTN_CE=1 时跳过 logprobs_from_logits_flash_attn，强制走

纯 PyTorch 的 log_softmax+gather 路径。

- verl/trainer/constants_ppo.py：get_ppo_ray_runtime_env() 把 NCCL_DETERMINISTIC、CUBLAS_WORKSPACE_CONFIG、FLASH_ATTENTION_DETERMINISTIC 等 env 通过 Ray runtime_env 转发给 actor 子进程；与 PYTHONHASHSEED/VLLM_BATCH_INVARIANT 不同，这些变量只在显式设置时转发，避免空字符串破坏 vLLM ParallelConfig 的 int 解析。

第 2 步：按 RM 类型决定串行策略

verl/experimental/reward_loop/reward_loop.py 的 RewardLoopManager.__init__ 现在能同时看到 reward_model.rollout 与 custom_reward_function.path：只有 " 无自定义 reward fn（判别式 /classify pooling）" 才强制 max_num_seqs=1 并告警，因为该路径不受 VLLM_BATCH_INVARIANT 覆盖；" 有自定义 reward fn（生成式 /v1/chat/completions）" 不再被强制串行，可依赖 batch invariance + per-request seed 并行推理。配置写入用 with open_dict(self.config) 包裹，防止 Hydra 冻结配置抛 ReadonlyConfigError。原逻辑从 RewardModelManager（reward_model.py）移除，无签名变化。

第 3 步：确定性多副本路由

- verl/workers/rollout/llm_server.py：GlobalRequestLoadBalancer 从 @ray.remote 装饰类改为普通类，在 _init_global_load_balancer() 内用 ray.remote(load_balancer_cls).remote(...) 包装，使调用方能继承并覆写 acquire_server。full_determinism 下 acquire_server 绕开 " 最少负载 "（候选集依赖异步到达时序），改为对全副本池做 hash(request_id) % len(servers) 全哈希路由；配对的 _vllm_request_id() 在 full_determinism 下把调用方稳定 request_id 直接透传给 vLLM，保证 KV-cache 与调度键跨运行一致。LLMServerManager.create(..., load_balancer_cls=...) 与 get_client(client_cls=...) 提供子类注入点，verl-omni 的扩散入口传入 DeterministicGlobalRequestLoadBalancer/DeterministicLLMServerClient。
- verl/experimental/reward_loop/router/naive_router.py：NaiveRouter 通过 VERL_FULL_DETERMINISM 环境变量感知确定性模式，_select_worker(request_id) 在等负载候选间用平台无关的 binascii.crc32(request_id) 做 tie-break（request_id 由请求体派生，不依赖 PYTHONHASHSEED），同一请求跨运行落到同一副本，从而中和副本级浮点差异。
- verl/experimental/teacher_loop/teacher_model.py 同步适配 ray.remote(GlobalRequestLoadBalancer) 的新实例化方式。

第 4 步：测试、CI 与文档配套

- tests/experimental/reward_loop/run_determinism_e2e_with_rm.py 重写为 E2E 判定 gate：subprocess.Popen 捕获训练 stdout，用正则解析 console logger 的 step:N - key:V 行得到 reward 曲线，两次运行 float32 逐位比对，退出码 0/1 作为判定；脚本显式声明 trainer.use_v1=false。该测试命中 vLLM matmul_persistent Triton 内核（需 128KB shared memory，超出 L20 的 99KB 上限），未接入 L20 CI，需在 A100/H100 手动运行。
- .github/workflows/vllm.yml 新增 generation-determinism CI 步骤（复用预烘焙 Qwen2.5-0.5B）；tests/workers/rollout/rollout_vllm/test_vllm_generation_determinism.

py 移除 GRM 跨实例用例（移交 verl-omni）；test_basic_agent_loop.py 等测试适配 LB 新实例化方式；docs/advance/determinism.md 更新 V0 后端要求、RM max_num_seqs策略与 E2E 验证方法。

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.acquire_server, LLMServerClient._vllm_request_id, LLMServerManager._init_global_load_balancer, LLMServerManager.get_client）：核心路由层：GlobalRequestLoadBalancer 改为可继承普通类，full_determinism 下全哈希路由；_vllm_request_id 透传稳定 request_id；LLMServerManager 新增 load_balancer_cls/client_cls 注入点，是本 PR 影响面最大的源码变更。
- verl/experimental/reward_loop/router/naive_router.py（模块 奖励路由；类别 source；类型 entrypoint；符号 NaiveRouter._select_worker, NaiveRouter._make_async_request）：RM 推理 HTTP 路由代理：full_determinism 下用 crc32 对等负载副本做确定性 tie-break，消除多副本 reward 路由对到达时序的依赖。
- verl/experimental/reward_loop/reward_loop.py（模块 奖励循环；类别 source；类型 core-logic；符号 RewardLoopManager.init）：RewardLoopManager 首次能看到 custom_reward_function.path，据此把 max_num_seqs=1 的强制缩小到判别式 RM，生成式 RM 恢复并行，是本 PR 的 reward 语义核心。
- verl/workers/engine/utils.py（模块 确定性原语；类别 source；类型 core-logic；符号 enable_full_determinism）：enable_full_determinism 是全局确定性入口，新增 NCCL 确定性 env 与 VERL_DISABLE_FLASH_ATTN_CE=1，是浮点确定性原语的基础。
- verl/utils/torch_functional.py（模块 工具函数；类别 source；类型 dependency-wiring；符号 logprobs_from_logits）：logprobs_from_logits 在 VERL_DISABLE_FLASH_ATTN_CE=1 时绕过非确定性的 flash-attn Triton 交叉熵内核，直接决定 log_probs 是否可复现。
- verl/trainer/constants_ppo.py（模块 运行环境；类别 source；类型 core-logic；符号 get_ppo_ray_runtime_env）：get_ppo_ray_runtime_env 负责把 NCCL/CUBLAS/FLASH 确定性 env 经 Ray runtime_env 传播给 actor 子进程，并修复空字符串破坏 vLLM 解析的问题。
- verl/experimental/reward_loop/reward_model.py（模块 奖励模型；类别 source；类型 data-contract；符号 RewardModelManager.init）：移除 RewardModelManager 中对 max_num_seqs 的无条件强制，串行策略职责上移到 RewardLoopManager，明确 RM 类型感知的边界。
- tests/experimental/reward_loop/run_determinism_e2e_with_rm.py（模块 端到端测试；类别 test；类型 test-coverage；符号 run_training, parse_reward_curve_from_stdout, _parse_float）：E2E 判定 gate：重写为 stdout 解析 + float32 逐位比对 + 退出码判定，显式要求 V0 trainer 后端，是本 PR 正确性的最终验收工具。

关键符号：enable_full_determinism, logprobs_from_logits, get_ppo_ray_runtime_env, GlobalRequestLoadBalancer.acquire_server, LLMServerClient._vllm_request_id, LLMServerManager._init_global_load_balancer, LLMServerManager.get_client, NaiveRouter._select_worker, RewardLoopManager.init,

parse_reward_curve_from_stdout, _parse_float

关键源码片段

verl/workers/rollout/llm_server.py

核心路由层：GlobalRequestLoadBalancer 改为可继承普通类，full_determinism 下全哈希路由；_vllm_request_id 透传稳定 request_id；LLMServerManager 新增 load_balancer_cls/client_cls 注入点，是本 PR 影响面最大的源码变更。

```
# verl/workers/rollout/llm_server.py
# GlobalRequestLoadBalancer 由 @ray.remote 装饰类改为普通类，实例化时用
# ray.remote(...) 包装，从而允许调用方继承并覆写 acquire_server 后注册为 actor。

class GlobalRequestLoadBalancer:
    def __init__(
        self,
        servers: dict[str, ray.actor.ActorHandle],
        max_cache_size: int = DEFAULT_ROUTING_CACHE_SIZE,
        full_determinism: bool = False,
    ):
        self._servers = dict(servers)
        self._inflight_requests = {sid: 0 for sid in servers} # 每个副本的在途请求数
        self._request_id_to_server = LRUCache(maxsize=max_cache_size) # 粘性会话缓存
        self._full_determinism = full_determinism

    def acquire_server(self, request_id: str) -> tuple[str, ray.actor.ActorHandle]:
        # 先命中粘性会话：同一多轮对话必须路由到同一副本以复用 prefix cache
        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] # 副本已被移除，清理过期缓存

        if not self._inflight_requests:
            raise RuntimeError("No available servers in load balancer")

        if self._full_determinism:
            # 全哈希路由：同一 request_id 跨运行总是落到同一副本。
            # 最少负载选择的候选集依赖异步到达时序，运行间不稳定，故整体绕过。
            server_id = list(self._servers)[hash(request_id) % len(self._servers)]
        else:
            min_count = min(self._inflight_requests.values())
            candidates = [sid for sid, count in self._inflight_requests.items() if count == min_count]
            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]
```

```
# LLMServerClient 侧：默认每轮生成新 uuid，使每次生成都是独立 vLLM 请求；
# full_determinism 下透传调用方的稳定 request_id，保证 vLLM 的 KV-cache 与
# 调度键跨运行一致，是确定性路由的最后一环。
def _vllm_request_id(self, request_id: str) -> str:
    if getattr(self.config.actor_rollout_ref.rollout, "full_determinism", False):
        return request_id
    return uuid4().hex
```

verl/experimental/reward_loop/router/naive_router.py

RM 推理 HTTP 路由代理：full_determinism 下用 crc32 对等负载副本做确定性 tie-break，消除多副本 reward 路由对到达时序的依赖。

```
# verl/experimental/reward_loop/router/naive_router.py
# NaiveRouter 是 RM 推理的 HTTP 路由代理；确定性开启（VERL_FULL_DETERMINISM=1）时，
# 先读取请求体并把它作为 request_id 种子，用平台无关的 crc32 做等负载 tie-break，
# 保证同一请求跨运行路由到同一副本 —— 否则副本间浮点状态差异无法用 seed 抹平。

import binascii

class NaiveRouter:
    def __init__(self, worker_urls: list[str], ...):
        ...
        # 子进程通过环境变量感知确定性模式，独立于 Hydra 配置
        self.full_determinism = os.getenv("VERL_FULL_DETERMINISM", "0") == "1"
        ...

    def _select_worker(self, request_id: bytes | None = None) -> str:
        # 从最少负载的副本集合中选择；request_id 非 None 表示确定性模式
        min_count = min(self.request_counts.values())
        candidates = [url for url, c in self.request_counts.items() if c == min_count]
        if len(candidates) == 1:
            url = candidates[0]
        elif request_id is not None:
            # crc32 不依赖 PYTHONHASHSEED，跨平台结果一致
            url = candidates[binascii.crc32(request_id) % len(candidates)]
        else:
            url = candidates[0]
        self.request_counts[url] += 1
        return url
```

tests/experimental/reward_loop/run_determinism_e2e_with_rm.py

E2E 判定 gate：重写为 stdout 解析 + float32 逐位比对 + 退出码判定，显式要求 V0 trainer 后端，是本 PR 正确性的最终验收工具。

```
# tests/experimental/reward_loop/run_determinism_e2e_with_rm.py
# E2E 判定 gate：两次相同 PPO 运行后比较 float32 reward 曲线。
# 曲线直接从训练子进程 stdout 解析（console logger 输出 step:N - key:V 行），
# 退出码 0= 对齐、1= 未对齐或运行崩溃。

_STEP_LINE_RE = re.compile(r"(?<![\w:])step\s*:\s*(\d+)\s*-?\s*(.*)$")
```

```
def _parse_float(token: str) -> float:
    # 兼容 np.float64(x)、np.int32(n) 等 repr 形式
    token = token.strip()
    m = _NP_WRAP.match(token)
    if m:
        token = m.group(1).strip()
    try:
        return float(token)
    except ValueError:
        m = _NUM_RE.search(token)
        return float(m.group(0)) if m else float("nan")


def parse_reward_curve_from_stdout(stdout: str):
    metrics_by_step = {}
    for line in stdout.splitlines():
        m = _STEP_LINE_RE.search(line)
        if not m:
            continue
        step = int(m.group(1))
        data = {}
        # 值可能自身包含 ':'，先按 " - " 分段，再只对第一个 ':' 切分
        for chunk in m.group(2).split(" - "):
            chunk = chunk.strip()
            if not chunk or ":" not in chunk:
                continue
            key, _, value = chunk.partition(":")
            data[key.strip()] = value.strip()
        if data:
            metrics_by_step[step] = data

    if not metrics_by_step:
        return None, None
    steps = sorted(metrics_by_step)
    rewards = []
    for s in steps:
        key = _pick_reward_key(metrics_by_step[s])
        if key is None:
            print(f"ERROR: No reward key found in step {s}")
            return None, None
        rewards.append(_parse_float(metrics_by_step[s][key]))
    return steps, rewards
```

评论区精华

评审中最有价值的交锋集中在路由抽象与后端选型：

1. wuxibin89 提出设计核心争议："Can we move both DeterministicGlobalRequestLoadBalancer and DeterministicLLMServerClient to

verl-omni, and pass class as argument? Since LLM actually not supportdeterministic mode, keep it here is misleading." —— 最终采纳双轨：默认 LB 保留full_determinism 分支（保证 V0 PPO 直接可用），同时新增load_balancer_cls/client_cls 注入点，确定性子类放在 verl-omni。

2. wuxibin89 两次追问 "Do we still need this?"（acquire_server 的 determinism 分支与 _vllm_request_id），作者 KaisennHu 回应："Still needed. these flag branches are its only routing determinism — removing them breaks V0 PPO." 这解释了保留标志分支的原因：verl 自己的 main_ppo V0 入口仍需可用。
3. gemini-code-assist 的稳定性建议均被采纳：用 with open_dict(self.config) 包裹 DictConfig 修改，以及用 binascii.crc32 替代内置 hash() 做路由 tie-break。
4. E2E 分歧定位：SamitHuang 在 #7216 复测确认 rollout 与 reward 模块已逐位对齐，" The first divergence happens in actor update."；KaisennHu 最终定位并修复："I fixed this by setting trainer.use_v1=false, tq is not supported for full determinism."
5. 技术债提示：wuxibin89 指出 "Since v0 is deprecated and will be removed soon, it's better to add full_determinism in verl-omni's own entrypoint"，并要求清理 FSDP transformer_impl.py 中的 [DET] 调试代码（合入版本已去除）。

- reward_loop.py 直接修改 DictConfig 可能触发 ReadonlyConfigError (correctness): 已采纳，合入版本用 with open_dict(self.config) 保护配置写入。
- NaiveRouter 确定性路由应使用 crc32 替代内置 hash() (correctness): 已采纳，_select_worker 用 binascii.crc32(request_id) % len(candidates) 做等负载 tie-break，不依赖 PYTHONHASHSEED。
- acquire_server 与 _vllm_request_id 的 full_determinism 分支是否还需要 (design): 保留标志分支：确定性子类仅服务 verl-omni 扩散入口，verl 默认 LB 的 full_determinism 分支是 V0 PPO 路由确定性的唯一实现。
- 确定性路由应抽象为子类注入而非内置标志分支 (design): 采纳双轨设计：GlobalRequestLoadBalancer 改为普通类 + 实例化时 ray.remote 包装以支持继承；LLMServerManager 新增 load_balancer_cls/client_cls 注入点，verl-omni 定义并传入确定性子类。
- V0 trainer 已废弃，确定性 env 注入应放 verl-omni entrypoint (design): 入口侧 env 导出与 V0 backend 强制留在 verl-omni entrypoint；verl 的 main_ppo V0 入口标记弃用待移除，文档要求显式 trainer.use_v1=false。
- E2E 复测：首个分歧出现在 actor update，V1 TQ 不支持全确定性 (testing): V1 后端（AgentLoopManagerTQ）的完成序收集在原理上无法 bitwise 复现，E2E gate 与文档显式要求 V0（trainer.use_v1=false）。
- FSDP transformer_impl 中的 [DET] 调试代码清理 (style): 调试日志代码未进入合入版本，问题定位用的 [DET] 插桩被清除。

风险与影响

- 风险：

1. 性能代价：full_determinism 下 rollout 从 " 最少负载 + 粘性 " 退化为全哈希路由，负载可能倾斜，长尾吞吐下降；VERL_DISABLE_FLASH_ATTN_CE=1 使 log_probs 回退到 log_softmax+gather 路径，内存与算力开销上升；判别式 RM 仍强制 max_num_seqs=1 串行。这些是确定性的固有代价，文档已说明。
2. 正确性边界：llm_server.py 的全哈希路由依赖 hash() 与 PYTHONHASHSEED 正确传播（constants_ppo.py 已覆盖 runtime_env，但任何绕过 runtime_env 的启动方式都会破坏确定性）；RM 侧改用 crc32 规避了该依赖，两边路由使用了不同哈希方案，语义需注意。
3. 静默失配风险：reward_loop.py 只在 custom_reward_function.path is None 时强制 max_num_seqs=1；若用户自定义 reward fn 实际内部走 /classify pooling，将不会被强制串行，存在未覆盖的浮点漂移。
4. 兼容性：GlobalRequestLoadBalancer 的 actor 化方式从装饰器改为实例化包装，所有直接 GlobalRequestLoadBalancer.remote(...) 的外部调用方（recipe、第三方代码）需同步适配；本 PR 内 teacher_model.py 与测试已同步。
5. 生命周期：全确定性要求 V0 trainer，而 V0 即将废弃；V1 TQ 的完成序收集问题在合入时未解决，未来把 full_determinism 迁移到 V1 需要重新设计收集语义（提交序对齐）。
6. 测试盲区：E2E gate 未接入 CI（硬件限制），回归只能手动在 A100/H100 验证；test_vllm_generation_determinism.py 移除的 GRM 跨实例用例覆盖转移到了 verl-omni 仓库。 - 影响：用户影响：启用 full_determinism=true 的 PPO 用户（含 verl-omni 自定义生成式 RM 用户）可获得从 rollout 到 reward 全链路的 bitwise 可复现性；代价是必须显式设 trainer.use_v1=false、接受确定性路由带来的负载倾斜与性能开销、判别式 RM 依旧串行。系统影响：路由层引入 " 默认 LB + 标志分支 + 子类注入 " 三层设计，GlobalRequestLoadBalancer 变为可继承的普通类；训练前确定性 env 通过 runtime_env 统一传播，覆盖 actor/ref/rollout/RM 全部子进程。团队影响：verl 与 verl-omni 的确定性职责边界被固化（LLM 层中立、确定性子类在 verl-omni 侧）；CI 新增 generation-determinism 步骤，但 reward E2E gate 因硬件限制暂无法进入 CI。 - 风险标记：核心路由逻辑变更，仅 V0 trainer 支持，E2E 测试未接入 CI，确定性依赖 PYTHONHASHSEED 传播，确定性路由牺牲负载均衡

关联脉络

- PR #6572 full determinism for vLLM rollout and reward model: PR body 明确声明本 PR "Enhances df35e73 (full determinism for vLLM rollout and reward model, PR #6572)"，本 PR 在其基础上补齐 reward 路径的浮点确定性、RM 串行策略与确定性路由。
- PR #7300 [algo] fix: carry running_return through observation spans in REINFORCE++ (#7278): 双方都改动 tests/trainer/ppo/test_reinforce_pp_multiturn_on_cpu.py；本 PR 对该多轮 CPU 测试的配置做了同步适配。
- PR #7337 [ci] chore: add three baselines for npu's nightly ci: 同样修改 test_reinforce_pp_multiturn_on_cpu.py，在 NPU nightly CI 上扩展同一测试文件覆盖。