PR #6129 完整报告

verl-project/verl

[BREAKING][rollout] refactor: move LLMServerManager out of AgentLoopManager

合并时间：2026-04-29 22:24

原文链接：http://prhub.com.cn/verl-project/verl/pull/6129

执行摘要

- 一句话：将 LLM 服务器管理从 AgentLoopManager 抽离为独立模块
- 推荐动作：值得精读。本 PR 体现了良好的架构解耦策略：通过提取公共服务，将变异点（agent 框架）与稳定基础设施（服务器管理）分离。对于需要扩展 agent 框架的开发者，本 PR 的接口设计和代码组织是很好的参考。特别是 GlobalRequestLoadBalancer 和 LLMServerClient 的设计值得学习。

功能与动机

PR body 指出 AgentLoopManager 是一个特定的 agent 框架实现，之前其内部拥有 LLM 服务器副本的启动 / 关闭 / 负载均衡 / 性能分析 /KV-cache 清理，这迫使所有替代 agent 框架要么继承 AgentLoopManager，要么重新实现 rollout 服务器管道。为了支持可插拔的 agent 框架，需要将 LLM 服务器管理抽取为独立模块，使得任何 agent 框架都可以通过 LLMServerClient 复用相同的 rollout 服务器。关联的 RFC Issue #5790 和 #5737 也强调了类似的解耦需求。

实现拆解

1. 创建新模块verl/workers/rollout/llm_server.py，包含 GlobalRequestLoadBalancer（Ray 远程 actor，提供粘性会话和最小负载路由）、LLMServerManager（管理服务器生命周期）和 LLMServerClient（提供给 agent 工作进程使用的客户端代理）。LLMServerClient 依赖于 GlobalRequestLoadBalancer 实现请求分发。
2. 简化 AgentLoopManager：修改 verl/experimental/agent_loop/agent_loop.py，删除原有的 AsyncLLMServerManager、GlobalRequestLoadBalancer 和 Prometheus 配置函数，改为导入并使用 LLMServerClient。AgentLoopManager 不再直接管理服务器，而只负责 agent 循环编排。同时移除对 teacher_loop 和 distillation 的依赖。
3. 迁移 FullyAsyncAgentLoopManager：删除 verl/experimental/fully_async_policy/agent_loop/agent_loop.py，将其中的 FullyAsyncLLMServerManager 和 FullyAsyncAgentLoopWorker 移除。新的 FullyAsyncAgentLoopManager 移到 verl/experimental/fully_async_policy/fully_async_rollouter.py，并继承自 AgentLoopManager，通过 llm_server_manager 获取副本。
4. 移动 Prometheus 配置函数：将 update_prometheus_config、write_config_file、reload_prometheus 从 verl/experimental/agent_loop/prometheus_utils.py 移到 verl/workers/rollout/utils.py，供 LLMServerManager 使用，减少 agent 框架的依赖。

5. 更新多个 trainer 入口：修改 verl/trainer/main_ppo_sync.py、verl/trainer/ppo/ray_trainer.py 等文件，调整导入路径和调用方式以适配新 API。
6. 更新测试与文档：更新 tests/checkpoint_engine/test_special_server_adapter.py、tests/experimental/agent_loop/* 测试文件以及 docs/advance/agent_loop.rst、docs/start/agentic_rl.rst 文档，确保一致性。

关键文件：

- verl/workers/rollout/llm_server.py（模块 工作进程；类别 source；类型 dependency-wiring；符号 GlobalRequestLoadBalancer, init, acquire_server, release_server）：新模块，定义了 LLM 服务器管理的核心抽象：GlobalRequestLoadBalancer（全局负载均衡器）和 LLMServerClient（客户端代理）。是本次重构的核心产出。
- verl/experimental/agent_loop/agent_loop.py（模块 实验模块；类别 source；类型 dependency-wiring；符号 GlobalRequestLoadBalancer, init, acquire_server, release_server）：原接口定义文件，本次大幅简化：删除了服务器管理类和负载均衡器，改为导入 LLMServerClient，使 AgentLoopManager 专注 agent 编排。
- verl/experimental/fully_async_policy/fully_async_rollouter.py（模块 实验模块；类别 source；类型 core-logic；符号 FullyAsyncAgentLoopManager, generate_sequences_single, _select_best_worker, get_rollout_wg）：将 FullyAsyncAgentLoopManager 从 agent_loop 子包迁移至此，并简化为继承 AgentLoopManager，不再管理 LLM 服务器，而是通过 llm_server_manager.get_replicas() 获取副本。

关键符号：GlobalRequestLoadBalancer.acquire_server, GlobalRequestLoadBalancer.release_server, LLMServerClient._acquire_server, LLMServerClient.init, FullyAsyncAgentLoopManager.generate_sequences_single, FullyAsyncAgentLoopManager._select_best_worker

关键源码片段

verl/experimental/fully_async_policy/fully_async_rollouter.py

将 FullyAsyncAgentLoopManager 从 agent_loop 子包迁移至此，并简化为继承 AgentLoopManager，不再管理 LLM 服务器，而是通过 llm_server_manager.get_replicas() 获取副本。

```
# ( 许可证头省略 )
from verl.experimental.agent_loop.agent_loop import AgentLoopManager
from verl.workers.rollout.llm_server import LLMServerManager # 导入新模块

class FullyAsyncAgentLoopManager(AgentLoopManager):
    async def generate_sequences_single(self, prompts: DataProto) -> DataProto:
        """Split input batch and dispatch to agent loop workers."""
        worker = self._select_best_worker()
        output_future = worker.generate_sequences.remote(prompts)
        return await asyncio.wrap_future(output_future.future())
```

```
    def _select_best_worker(self):
        """Select the best worker, simple round-robin load balancing"""
        if not hasattr(self, "_worker_index"):
            self._worker_index = 0
        worker = self.agent_loop_workers[self._worker_index]
        self._worker_index = (self._worker_index + 1) % len(self.agent_loop_workers)
        return worker

    def get_replicas(self):
        """Get rollout worker group"""
        return self.llm_server_manager.get_replicas() # 不再直接维护 roll_replicas
```

评论区精华

Review 主要讨论点：

- 设计评价：gemini-code-assist[bot] 确认重构合理，核心逻辑转移成功，无额外反馈。
- 未来扩展：ArronHZG 认为当前实现不够优雅，计划后续改为传入客户端类初始化；wuxibin89 回应注意到 PR #5990 对 FullyAsyncLLMServerManager 的额外修改，同意未来可能需要子类化 LLMServerClient 并传入额外参数。
- 兼容性确认：PeterSH6 询问旧 AsyncLLMServerManager 的计划，wuxibin89 说明已替换为 LLMServerClient。
- 实现优雅性与未来扩展 (design)：当前实现被接受，但留出了未来通过子类化扩展的接口。
- 旧 AsyncLLMServerManager 的处置 (question)：旧类已移除，使用新导入路径。

风险与影响

- 风险：主要风险为 breaking change：所有从 verl.experimental.agent_loop 导入 AsyncLLMServerManager / FullyAsyncLLMServerManager 的外部代码需要更新。LLMServerClient 接口与旧接口不完全兼容，AgentLoopManager.create(...) 签名已变化。但内部代码已全部更新，测试已通过。Prometheus 配置函数移动后，若其他模块直接导入旧位置会失败。整体风险中等，影响面限于自定义 agent 框架和扩展模块的用户。
- 影响：用户：自定义 agent 框架的使用者需要更新导入和 API 调用；使用 VeRL 内置 agent 的用户无感知。系统：解耦后模块边界清晰，服务器管理可独立演化和测试，支持未来快速集成新的 agent 框架（如 NeMo-Gym、AWS Bedrock AgentCore）。团队：维护职责划分更清楚，agent 框架和 rollout 基础设施可由不同团队独立迭代。
- 风险标记：核心路径变更，公共 API 变更，跨模块重构，Breaking Change

关联脉络

- PR #5990 fully_async: add support for model engine server handle: 讨论中提到此 PR 对 FullyAsyncLLMServerManager 增加了额外字段，影响了本 PR 的扩展设计。
- PR #5787 [RFC] NeMo-Gym integration: PR body 指出本重构有助于 NeMo-Gym 等 agent 框架集成。

- PR #4216 AWS Bedrock AgentCore integration: PR body 提及本重构可支持 AWS Bedrock AgentCore。