PR #1920 完整报告

THUDM/slime

Move fully_async example to main codebase

合并时间：2026-05-18 19:46

原文链接：http://prhub.com.cn/THUDM/slime/pull/1920

执行摘要

- 一句话：将 fully_async rollout 从示例迁移至主代码库，新增 slime.rollout.fully_async_rollout 模块
- 推荐动作：值得精读，展示了如何将实验性代码正式合入主库的标准化流程：代码迁移、类型标注增强、测试配套、文档更新、CI 集成。其中全局 worker 单例和 ABORTED 样本重定向的设计值得借鉴。

功能与动机

PR 将 fully_async rollout 从示例代码迁移到主代码库，使其成为 slime 的标准 rollout 方案之一。用户可通过 --rollout-function-path 直接使用，无需复制示例代码。同时，该模式通过后台线程保持 in-flight 轨迹，减少训练等待，提升训练吞吐。

实现拆解

1. 删除 examples/fully_async/fully_async_rollout.py，将其核心逻辑（AsyncRolloutWorker 类、全局 worker 管理）迁移到 slime/rollout/fully_async_rollout.py，并优化了并发计算（考虑 engine 数量）。
2. 修改 slime/rollout/sglang_rollout.py 中 generate_and_rm_group 的返回类型标注，支持 list[Sample] | list[list[Sample]]，以适应 multi-turn agent rollout 的扇出场景。
3. 新增 CI 测试 tests/test_qwen2.5_0.5B_fully_async_short.py，在 Qwen2.5-0.5B 模型上执行 3 轮 GRPO 以验证 end-to-end 路径。
4. 更新 examples/fully_async/ 下的启动脚本（新增 run-qwen2.5-0.5B-fully_async.sh，删除旧的 run-qwen3-4b-fully_async.sh）和 README，指导用户如何使用新模块。
5. 在 CI 配置文件 .github/workflows/pr-test.yml 和 .jinja2 模板中注册新测试。

关键文件：

- slime/rollout/fully_async_rollout.py（模块 异步 rollout；类别 source；类型 core-logic；符号 _get_global_worker, _stop_global_worker, AsyncRolloutWorker, init）：核心新模块，实现 fully-async rollout 的后台 worker
- slime/rollout/sglang_rollout.py（模块 生成奖励；类别 source；类型 core-logic；符号 generate_and_rm_group）：修改 generate_and_rm_group 的返回类型，支持 multi-turn 扇出

- tests/test_qwen2.5_0.5B_fully_async_short.py（模块 集成测试；类别 test；类型 test-coverage；符号 prepare, execute）：新增 CI 测试，验证 fully-async rollout 端到端路径
- examples/fully_async/fully_async_rollout.py（模块 示例脚本；类别 source；类型 deletion；符号 get_global_worker, stop_global_worker, AsyncRolloutWorker, init）：旧实现，被删除，内容迁移至 slime/rollout/ 中
- examples/fully_async/README.md（模块 文档；类别 docs；类型 documentation）：更新文档，反映新的模块路径和使用方式

关键符号：_get_global_worker, _stop_global_worker, AsyncRolloutWorker, start, stop, get_completed_groups, queue_size, generate_rollout_fully_async, generate_and_rm_group

关键源码片段

slime/rollout/fully_async_rollout.py

核心新模块，实现 fully-async rollout 的后台 worker

```
# 全局 worker 单例，跨 rollout 调用共享，保持后台队列持续运行
_global_worker: AsyncRolloutWorker | None = None
_worker_lock = threading.Lock()


def _get_global_worker(args, data_buffer) -> AsyncRolloutWorker:
    """获取或创建全局异步 worker。

    根据 args 中的 sglang 并发度和引擎数量计算并发度，确保与 sglang_rollout
    的 per-sample 信号量上限匹配。
    """
    global _global_worker
    with _worker_lock:
        if _global_worker is None or not _global_worker.worker_thread.is_alive():
            logger.info("starting fully-async rollout worker")
            num_engines = max(1, args.rollout_num_gpus // args.rollout_num_gpus_per_engine)
            _global_worker = AsyncRolloutWorker(
                args, data_buffer, concurrency=args.sglang_server_concurrency * num_engines
            )
            _global_worker.start()
        return _global_worker


def _stop_global_worker() -> None:
    """停止全局 worker，在进程退出时自动调用。"""
    global _global_worker
    with _worker_lock:
        if _global_worker is not None:
            _global_worker.stop()
            _global_worker = None
```

```
atexit.register(_stop_global_worker)


class AsyncRolloutWorker:
    """后台线程 + asyncio 事件循环，持续从 data_buffer 消费样本并调用 generate_and_rm_
    group。"""

    def __init__(self, args, data_buffer, concurrency: int = 10):
        self.args = args
        self.data_buffer = data_buffer
        self.concurrency = concurrency
        self.running = True
        self.output_queue: queue.Queue[tuple[int, list[Sample]]] = queue.Queue(maxsize=1000)
        self.worker_thread: threading.Thread | None = None
        self.state = GenerateState(args)

    def start(self) -> None:
        if self.worker_thread is None or not self.worker_thread.is_alive():
            self.worker_thread = threading.Thread(target=self._thread_main, name="fully-async-
            rollout", daemon=True)
            self.worker_thread.start()

    def stop(self) -> None:
        self.running = False
        if self.worker_thread and self.worker_thread.is_alive():
            self.worker_thread.join(timeout=5)

    def get_completed_groups(self) -> list[tuple[int, list[Sample]]]:
        completed: list[tuple[int, list[Sample]]] = []
        while True:
            try:
                completed.append(self.output_queue.get_nowait())
            except queue.Empty:
                break
        return completed

    def queue_size(self) -> int:
        return self.output_queue.qsize()

    def _thread_main(self) -> None:
        asyncio.run(self._loop())

    async def _loop(self) -> None:
        # 连续工作循环；具体实现与旧版 continuous_worker_loop 类似，
        # 但使用 self.concurrency 代替 rollout_batch_size 控制最大并发数，
        # 并将 ABORTED 样本重新放回 data_buffer。
        # 完整代码见提交后的文件。
        pass
```

slime/rollout/sglang_rollout.py

修改 generate_and_rm_group 的返回类型，支持 multi-turn 扇出

```
async def generate_and_rm_group(
    args: Namespace, group: list[Sample], sampling_params: dict[str, Any], evaluation: bool =
    False
) -> list[Sample] | list[list[Sample]]:
    # ``generate_and_rm`` may return either a ``Sample`` or a ``list[Sample]``
    # depending on whether the ``--custom-generate-function-path`` callable
    # emits one trainable sample or several (e.g. multi-turn agent rollouts
    # that fan out into multiple prefix-chained samples). The asyncio.gather
    # below preserves whichever shape each task produced, so the group is
    # ``list[Sample]`` for plain rollouts and ``list[list[Sample]]`` for
    # the fan-out case.
    state = GenerateState(args)
    # ... 其余函数体未变
```

tests/test_qwen2.5_0.5B_fully_async_short.py

新增 CI 测试，验证 fully-async rollout 端到端路径

```
def execute():
    ckpt_args = f"--hf-checkpoint /root/models/{MODEL_NAME}/ " f"--ref-load /root/models/
    {MODEL_NAME}/ "

    rollout_args = (
        # 唯一与 test_qwen2.5_0.5B_async_short.py 不同的行:
        # 使用公共的全异步 rollout 函数。
        "--rollout-function-path slime.rollout.fully_async_rollout.generate_rollout_fully_async "
        "--prompt-data /root/datasets/dapo-math-17k/dapo-math-17k.jsonl "
        "--input-key prompt "
        "--label-key label "
        "--apply-chat-template "
        "--rollout-shuffle "
        "--rm-type deepscaler "
        "--num-rollout 3 "
        "--rollout-batch-size 8 "
        "--n-samples-per-prompt 4 "
        "--rollout-max-response-len 8192 "
        "--rollout-temperature 0.8 "
        "--global-batch-size 32 "
        "--balance-data "
    )
    # ... 其他参数组合
    train_args = f"{ckpt_args} {rollout_args} {optimizer_args} {grpo_args} ..."
    U.execute_train(
        train_args=train_args,
        num_gpus_per_node=NUM_GPUS,
        megatron_model_type=MODEL_TYPE,
        train_script="train_async.py",
```

)

评论区精华

无 review 评论，PR 由作者自行合入。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险在于新模块的 worker 生命周期管理：全局 worker 使用线程和 asyncio，若未正确停止可能导致资源泄漏；ABORTED 样本重定向逻辑依赖 data_buffer 的正确行为；并发度计算依赖 sglang 引擎数量，配置错误可能引起性能下降。此外，新增的 CI 测试仅在特定模型下运行，覆盖有限。
- 影响：对用户而言，fully_async rollout 成为一级特性，可通过命令行参数直接使用；旧示例路径（如 fully_async_rollout.generate_rollout_fully_async）不再可用，需改为 slime.rollout.fully_async_rollout.generate_rollout_fully_async。对系统而言，后台 worker 线程在训练周期之间持续运行，可能略微增加显存占用，但显著降低 rollout 等待时间。对团队而言，维护成本增加一个核心模块，但测试覆盖降低了回归风险。
- 风险标记：核心路径变更，并发控制复杂度，测试覆盖有限，生命周期管理

关联脉络

- PR #1890 Add missing metrics to log: 都与 rollout 日志和性能监控相关
- PR #1873 Use Ray ObjectRef await instead of asyncio.to_thread: 都涉及异步 rollout 优化