PR #2056 完整报告

THUDM/slime

Use /v1/loads to re-abort server

合并时间：2026-06-11 16:50

原文链接：http://prhub.com.cn/THUDM/slime/pull/2056

执行摘要

- 一句话：使用 /v1/loads 接口重试中止服务器
- 推荐动作：值得精读。该 PR 展示了如何通过轮询负载接口实现可靠的服务器中断，设计简洁清晰。尤其是 num_requests_from_load 的递归解析模式，可复用于其他类似场景。

功能与动机

原始 abort 逻辑只发送一次 abort 请求，服务器可能仍存在 pending 请求，导致后续训练状态不一致。PR 旨在通过轮询负载接口实现可靠的 'abort until idle' 语义。

实现拆解

1. 在 slime/backends/sglang_utils/server_control.py 中新增 5 个函数：
   - num_requests_from_load: 递归解析 /v1/loads 返回的结构，提取 num_reqs、total_reqs 等字段，返回请求总数。
   - _abort_server_once: 向单个服务器发送 abort 请求。
   - _get_server_num_requests: 调用 /v1/loads?include=core 并通过 num_requests_from_load 获得当前请求数。
   - abort_server_until_idle: 核心重试循环：每轮先 abort，再检查负载，若负载 ≤ 0 则退出，否则等待 3 秒后重试。
   - abort_servers_until_idle: 对多个 URL 并发执行 abort_server_until_idle。
2. 在 slime/rollout/sglang_rollout.py 中：
   - 删除原有的 6 行“发送 abort → 收集异常”的代码。
   - 新增导入 abort_servers_until_idle，并用 await abort_servers_until_idle(urls) 替换原逻辑。
   - abort 函数控制流不变：获取 URL 列表 → 调用 abort_servers_until_idle → 等待 pending 任务完成。

关键文件：

- slime/backends/sglang_utils/server_control.py（模块 后端控制；类别 source；类型 core-logic；符号 num_requests_from_load, _abort_server_once, _get_server_num_requests, abort_server_until_idle）：新增的服务器控制模块，实现了基于 /v1/loads 的可靠 abort 逻辑，是本次 PR 的核心变更。

- slime/rollout/sglang_rollout.py（模块 推理调度；类别 source；类型 dependency-wiring）：修改了 abort 函数，用新逻辑替换原有单次 abort，是调用侧的主要变更。

关键符号：num_requests_from_load, _abort_server_once, _get_server_num_requests, abort_server_until_idle, abort_servers_until_idle, abort

关键源码片段

slime/backends/sglang_utils/server_control.py

新增的服务器控制模块，实现了基于 /v1/loads 的可靠 abort 逻辑，是本次 PR 的核心变更。

```
# 递归解析 /v1/loads 响应，提取请求总数
import asyncio
import logging
from typing import Any

from slime.utils.http_utils import get, post

logger = logging.getLogger(__name__)

ABORT_RETRY_INTERVAL_SECONDS = 3 # 重试间隔


def num_requests_from_load(load: Any) -> int:
    """递归从负载结构中提取请求总数。"""
    if isinstance(load, list):
        return sum(num_requests_from_load(item) for item in load)
    if not isinstance(load, dict):
        return 0
    if "loads" in load: # 嵌套的 loads 字段
        return num_requests_from_load(load["loads"])
    # 尝试多个可能的 key
    for key in ("num_reqs", "num_total_reqs", "total_reqs"):
        value = load.get(key)
        if isinstance(value, int):
            return value
    # 回退：用 running + waiting 估算
    running = load.get("num_running_reqs", load.get("total_running_reqs"))
    waiting = load.get("num_waiting_reqs", load.get("total_waiting_reqs"))
    return (running if isinstance(running, int) else 0) + (
        waiting if isinstance(waiting, int) else 0
    )


async def _abort_server_once(url: str) -> None:
    """向单个服务器发送 abort 请求。"""
    try:
        await post(f"{url}/abort_request", {"abort_all": True})
    except Exception as e:
```

```
        logger.warning(f"Failed to abort SGLang server at {url}: {e}")


async def _get_server_num_requests(url: str) -> int:
    """获取服务器当前请求数（通过 /v1/loads）。"""
    return num_requests_from_load(
        await get(f"{url}/v1/loads?include=core")
    )


async def abort_server_until_idle(
    url: str, retry_interval: int = ABORT_RETRY_INTERVAL_SECONDS
) -> None:
    """循环 abort 直至服务器请求数 ≤ 0。"""
    attempt = 1
    while True:
        logger.info(f"Abort request for SGLang server {url}")
        await _abort_server_once(url)

        try:
            num_requests = await _get_server_num_requests(url)
        except Exception as e:
            logger.warning(
                f"Failed to get SGLang server load from {url}: {e}"
            )
            return # 无法获取负载时安全返回

        if num_requests <= 0:
            return # 已空闲

        logger.info(
            f"SGLang server {url} still has {num_requests} requests "
            f"after abort attempt {attempt}; "
            f"retrying in {retry_interval} seconds."
        )
        await asyncio.sleep(retry_interval)
        attempt += 1


async def abort_servers_until_idle(urls: list[str]) -> None:
    """并发中止所有服务器直至空闲。"""
    await asyncio.gather(*(abort_server_until_idle(url) for url in urls))
```

评论区精华

该 PR 无 review 评论和讨论。

- 暂无高价值评论线程

风险与影响

- 风险：无明显的回归风险。新逻辑在无法获取负载时（若 /v1/loads 不可用），会打印 warning 并直接返回，不会阻塞训练。但需注意若服务器持续有请求（如长推理任务），重试可能无限循环，但代码中有重试间隔和日志，提供了可观测性。另外，num_requests_from_load 硬编码了多个 key 名，若 sglang 未来变更 API 响应结构，可能导致负载统计不准确。
- 影响：
  - 对用户：中断训练的可靠性提升，避免因服务器未完全空闲导致的异常。
  - 对系统：新增了对 /v1/loads 接口的依赖，若 sglang 版本不支持该接口，则需回退到原有单次 abort 行为（当前代码没有 fallback）。
  - 对团队：抽象出 server_control.py，便于后续维护 sglang 服务器控制逻辑。
  - 风险标记：缺少 fallback 机制，依赖 sglang 内部 API

关联脉络

- 暂无明显关联 PR