

# PR #7837 完整报告

verl-project/verl

[vllm, hardware] fix: patch vllm-ascend on Ascend NPUs

合并时间: 2026-09-15 11:53

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7837>

## 执行摘要

- 一句话: NPU 补丁增加部分唤醒防护, 修复 GLM-5.2 异步训练崩溃
- 推荐动作: 建议精读。本 PR 是典型的“对外部引擎做安全 monkey-patch”的范本: 源码指纹校验、幂等 marker、spawn 入口包装三件事结合得很好, 对维护 vllm-ascend、sglang 等外部组件兼容层的开发者有直接参考价值。值得关注的设计决策是“先验证再替换”和“通过入口函数传播补丁到子进程”这两个模式。

## 功能与动机

PR body 明确指出: 需要在 Ascend NPU 上 patch vllm-ascend, 以支撑 GLM-5.2 Fully Async RL 训练。核心问题是 vLLM 引擎在 partial wake-up (仅调度器唤醒而 model executor 仍睡眠) 期间仍可能调用 `resume_scheduler` 或执行 dummy forward, 触碰睡眠中的显存导致崩溃。因此需要 backport 上游修复并在 verl 的 NPU 兼容补丁中统一管理, 同时保证 spawn 出来的引擎进程也能继承补丁。

## 实现拆解

本 PR 只改动 `verl/utils/vllm/npu_vllm_patch.py` 一个文件, 实现分三步:

1. 新增 spawn 入口包装: 模块级新增 `_glm52_run_engine_core` 与全局变量 `_glm52_original_run_engine_core`。该函数刻意不使用 `@wraps`, 以便 multiprocessing spawn 能按 `verl.utils.vllm.npu_vllm_patch:_glm52_run_engine_core` 路径导入, 在全新子进程中先调用 `patch_vllm_glm52_partial_wake_up()` 重新打补丁, 再执行真正的引擎入口。
2. 实现 `patch_vllm_glm52_partial_wake_up()` 核心逻辑:
  - 从 `vllm.v1.engine.core` 获取 `EngineCore.wake_up` 与 `DPEngineCoreProc.execute_dummy_batch`, 用 `_GLM52_PATCH_MARKER` 属性判断是否已打补丁, 保证幂等。
  - 通过 `_normalized_source` (去除全部空白后的源码) 校验目标实现: 若上游已包含修复模式则保留上游方法; 若签名或源码模式不匹配则抛出 `RuntimeError`, 避免对未知实现盲目打补丁。
  - 安装 `patched_wake_up`: 先剥离 `scheduling tag`, 只有当 `tags` 非空时才真正调用 `model_executor.wake_up(tags)`, 并且仅当 `is_sleeping` 为 `False` 时才 `resume_scheduler()`, 从根上避免部分唤醒期间恢复调度。

- 安装 `patched_dummy_batch`: 当 `model executor` 仍在睡眠时跳过 `dummy forward`, 但保留 `busy loop`、`DP` 集合通信与 `logging` 上下文。
  - 随后包装 `EngineCoreProc.run_engine_core` 为模块级入口; 若两个目标均已被上游修复, 则不再包装, 避免多余间接层。
3. 接入现有补丁流程: 在 `apply_npu_vllm_patches()` 的 GLM-5.2 分支中追加 `patch_vllm_glm52_partial_wake_up()` 调用, 同时把 `_normalized_source` 的错误提示从 `vllm-ascend 0.23.x Python sources` 修正为 `vllm/vllm-ascend 0.23.x Python sources`, 表明校验对象同时涵盖 `vLLM` 与 `vllm-ascend`。
4. 测试与部署配套: 没有新增自动化测试文件, 仅在 PR body 中说明已在 Ascend A3 完成 GLM-5.2 RL fully async 训练实测; 补丁受环境变量门控, 默认不开启, 降低对默认路径的影响。

关键文件:

- `verl/utils/vllm/npu_vllm_patch.py` (模块 兼容补丁; 类别 `source`; 类型 `dependency-wiring`; 符号 `_glm52_run_engine_core`, `patch_vllm_glm52_partial_wake_up`, `patched_wake_up`, `patched_dummy_batch`): 唯一变更文件, 集中实现 GLM-5.2 partial wake-up 防护: 新增 `spawn` 入口包装、源码指纹校验、`wake_up` 与 `dummy batch` 两个方法的 `patch`, 并接入 `apply_npu_vllm_patches`。

关键符号: `_glm52_run_engine_core`, `patch_vllm_glm52_partial_wake_up`, `patched_wake_up`, `patched_dummy_batch`

## 关键源码片段

### `verl/utils/vllm/npu_vllm_patch.py`

唯一变更文件, 集中实现 GLM-5.2 partial wake-up 防护: 新增 `spawn` 入口包装、源码指纹校验、`wake_up` 与 `dummy batch` 两个方法的 `patch`, 并接入 `apply_npu_vllm_patches`。

# `verl/utils/vllm/npu_vllm_patch.py` 核心实现 (整理后)

```
def _glm52_run_engine_core(*args, **kwargs):
    # 保持为模块级函数且不加 @wraps: multiprocessing spawn 需要按模块路径
    # verl.utils.vllm.npu_vllm_patch:_glm52_run_engine_core 导入它,
    # 在全新子进程里先重新打补丁, 再执行真正的引擎入口。
    patch_vllm_glm52_partial_wake_up()
    return _glm52_original_run_engine_core(*args, **kwargs)

def patch_vllm_glm52_partial_wake_up() -> None:
    """Backport vLLM 6f1ada0f91 + 11d60545e2: 部分唤醒期间禁止任何 forward."""
    global _glm52_original_run_engine_core

    from vllm.v1.engine import core

    engine_core = core.EngineCore
    dp_engine_core = core.DPEngineCoreProc
    original_wake_up = engine_core.wake_up
```

```

original_dummy_batch = dp_engine_core.execute_dummy_batch
# 用 marker 判断是否已打过补丁，保证重复调用幂等
wake_up_patched = getattr(original_wake_up, _GLM52_PATCH_MARKER, False)
dummy_batch_patched = getattr(original_dummy_batch, _GLM52_PATCH_MARKER, False)

# 先校验两个目标实现，再决定是否替换；已修复的上游版本直接保留，未知实现显式失败
if not wake_up_patched:
    source = _normalized_source(original_wake_up)
    wake_up_patched = "ifnotself.model_executor.is_sleeping:self.resume_scheduler()" in
    source
    if not wake_up_patched and (
        tuple(inspect.signature(original_wake_up).parameters) != ("self", "tags")
        or 'tags=[tfortintagsift!="scheduling"]' not in source
        or "iftagsisNoneortags:self.model_executor.wake_up(tags)" not in source
        or not source.endswith("self.resume_scheduler()")
    ):
        raise RuntimeError(f"{_GLM52_PATCH_ENV} found an unsupported EngineCore.wake_
        up implementation.")

if not dummy_batch_patched:
    loop_source = _normalized_source(dp_engine_core.run_busy_loop)
    dummy_batch_patched = (
        "elifnotself.model_executor.is_sleeping:withself.log_iteration_details(None):self.execute_
        dummy_batch()"
    ) in loop_source
    if not dummy_batch_patched and (
        "withself.log_iteration_details(None):self.execute_dummy_batch()" not in loop_source
        or _normalized_source(original_dummy_batch)
        != "defexecute_dummy_batch(self):self.model_executor.execute_dummy_batch()"
    ):
        raise RuntimeError(f"{_GLM52_PATCH_ENV} found an unsupported DPEngineCoreProc
        dummy-batch implementation.")

if not wake_up_patched:
    @wraps(original_wake_up)
    def patched_wake_up(self, tags=None):
        # 剥离仅用于唤醒调度器的 scheduling tag：若 tags 只剩空列表，
        # 则不再真正唤醒 model executor，避免部分唤醒时触碰睡眠中的显存。
        if tags is not None and "scheduling" in tags:
            tags = [t for t in tags if t != "scheduling"]
        if tags is None or tags:
            self.model_executor.wake_up(tags)
        # 只有 executor 完全醒着时才恢复调度，否则跳过本轮调度。
        if not self.model_executor.is_sleeping:
            self.resume_scheduler()

    setattr(patched_wake_up, _GLM52_PATCH_MARKER, True)
    engine_core.wake_up = patched_wake_up

```

```

if not dummy_batch_patched:
    @wraps(original_dummy_batch)
    def patched_dummy_batch(self):
        # 保持 busy loop 及其 DP 集合通信不变, logging 上下文照常执行,
        # 但 executor 睡眠时跳过 dummy forward, 避免触碰尚未唤醒的显存。
        if not self.model_executor.is_sleeping:
            return original_dummy_batch(self)

setattr(patched_dummy_batch, _GLM52_PATCH_MARKER, True)
dp_engine_core.execute_dummy_batch = patched_dummy_batch

# 包装 spawn 入口: 子进程会在引擎启动前重新执行本函数, 确保补丁在新进程内生效
entrypoint = core.EngineCoreProc.run_engine_core
if entrypoint is not _glm52_run_engine_core:
    _glm52_original_run_engine_core = entrypoint
    if not (wake_up_patched and dummy_batch_patched):
        core.EngineCoreProc.run_engine_core = staticmethod(_glm52_run_engine_core)

status = "already present" if wake_up_patched and dummy_batch_patched else "applied"
logger.info("GLM-5.2 vLLM partial wake-up patch %s", status)

```

## 评论区精华

该 PR 没有产生任何 review 评论或讨论线程, wucong25 直接 APPROVED。虽然没有公开讨论, 但代码本身体现了三个值得注意的设计决策:

- 先验证后替换: 通过 `_normalized_source` 做源码指纹匹配, 既保留上游已修复的实现, 又对不支持的上游版本显式 fail-fast, 避免黑盒 patch 造成更难排查的运行异常。
- 幂等与多进程传播: 用 marker 属性保证重复调用安全, 并通过包装 `run_engine_core` 入口让 spawn 子进程自动重新打补丁, 解决了 monkey-patch 在多进程场景下易丢失的典型问题。
- 最小行为变更: `patched_dummy_batch` 只跳过 forward, 保留 busy loop 与 DP collectives, 说明作者刻意控制补丁影响面, 避免破坏引擎内部通信协议。
- 暂无高价值评论线程

## 风险与影响

- 风险: 风险点集中在以下几点:
- 上游源码耦合: `_normalized_source` 对源码字符串做精确匹配, vLLM 或 vllm-ascend 一旦调整变量名、换行或注释格式, 即便语义未变也可能触发 `RuntimeError`, 导致启动直接失败。这是 fail-fast 的代价, 属于可用性风险而非静默错误。
- 多进程入口改造: 全局替换 `EngineCoreProc.run_engine_core` 会影响所有开启环境变量后 spawn 的 vLLM 引擎进程; `_glm52_original_run_engine_core` 依赖子进程重新 import 后再次调用 patch 函数来赋值, 若导入路径或调用顺序变化, 可能引用 None 而崩溃。
- 行为语义变化: `patched_dummy_batch` 跳过 dummy forward 后, 依赖 dummy batch 做 profiler 统计或通信长度对齐的路径可能受影响; 虽然注释声称保留 collectives, 但实际效

果仍缺自动化验证。

- 测试覆盖缺失：本 PR 无任何自动化测试，仅依赖单点实机验证，回归风险偏高。
- 影响：影响范围集中在 Ascend NPU + GLM-5.2 + Fully Async RL 的组合场景：
- 用户侧：开启 `VERL_VLLM_ASCEND_GLM52_PATCH=1` 后，GLM-5.2 fully async 训练在 Ascend A3 上不再因部分唤醒期间执行 forward 而崩溃；默认不开启，不影响其他 NPU 用户。
- 系统侧：vLLM 引擎启动路径被包装，spawn 子进程增加一次 patch 调用，性能开销可忽略；但引擎进程的入口函数被替换，可能影响外部对 `EngineCoreProc.run_engine_core` 的引用。
- 团队侧：与 PR 7779 一起构成 Ascend GLM-5.2 支持的功能主线，后续需要补齐针对不同 vLLM 版本组合的自动化回归测试，防止源码指纹误判。
- 风险标记：无自动化测试覆盖，依赖上游源码指纹匹配，多进程入口函数被替换，外部库 monkey-patch

## 关联脉络

- PR #7779 [vllm, hardware] fix: patch vllm-ascend for GLM-5.2 RL training on Ascend NPUs: 同一作者针对同一文件 `verl/utils/vllm/npu_vllm_patch.py` 的此前版本，已完成 RMSNorm、SFA 算子等基础 patch；本 PR 在其之上补充 partial wake-up 防护，属于同一条功能线的递进。
- PR #7762 [trainer, ckpt] fix: skip V1 async warmup after inflight reissue: 同为 Fully Async 训练稳定性修复，说明 `fully_async` 路径近期集中修 bug，本 PR 与其互相印证异步训练在 NPU/ 训练器两侧的稳定投入。
- PR #7722 [perf, tool] fix: Avoid overlapping rollout and training profilers in Trainer V1: 同为 Trainer V1 全异步路径的修复，涉及 rollout 与训练并发问题，与本次引擎侧唤醒时序修复在 `fully_async` 场景下可能共同影响训练稳定性。