

PR #56766 完整报告

vllm-project/vllm

[CI] Keep OTEL bytecode out of mounted checkouts

合并时间: 2026-09-14 17:52

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56766>

执行摘要

- 一句话: CI OTEL helper 禁用字节码写入, 避免 root 残留阻塞 checkout
- 推荐动作: 值得花 5 分钟左右精读, 但重点不在代码本身, 而在两点: 一是「root 容器 + host 挂载 checkout + Python 默认写 pycache」这条诊断链, 以及把防护收敛到 shell 包装层、同时补齐所有旁路调用点的做法; 二是「预防性修复与一次性运维清理必须分开推进」的处置范例——PR 合并并不代表 CI 立刻恢复, 评论里明确要求暂停 slot 而非浪费重试。如果团队维护 CI 或容器化测试环境, 建议顺手排查是否还有其它 root 写回挂载路径 (评论中的 vllm/logging_utils 已给出反例), 并考虑给新用例补上 GPU 采样分支的断言。

功能与动机

PR body 明确说明: OTEL helper 运行在 root 容器中、作用于宿主机挂载的 checkout, 其 Python 导入会留下 root 所有的 .buildkite/scripts/ci-otel/__pycache__ 文件; 在 h200-ci-6-23 上这个残留文件让下一次 Buildkite checkout 无法清理工作区, 重试时报 Permission denied, 随后是 destination path '.' already exists and is not an empty directory。作者在评论中持续补充证据, 把观测到的失败面从最初的 5 个 build 扩大到 13 个 job / 12 个 build, 并强调 PYTHONDONTWRITEBYTECODE 只作用于 helper 进程, 测试命令的环境与行为保持不变。

实现拆解

1. 定位污染源与调用面: .buildkite/scripts/ci-otel/ci_otel.sh 里有四类 Python 执行点——文件头部的 import ci_otel 可用性探测、统一包装函数 _ci_otel_python()、ci_otel_start() 中后台启动的 ci_gpu.py GPU 采样、退出钩子 _ci_otel_on_exit() 中的 ci_otel.py flush。这些调用都以 root 身份在挂载 checkout 上执行, Python 默认把 __pycache__ 写在模块旁边, 因此必须逐个堵住, 不能只改一处。
2. 收敛防护位置: _ci_otel_python() 是 shell 层共享入口, 但内部有 timeout 存在 / 不存在两条分支, 两边都加了 PYTHONDONTWRITEBYTECODE=1 前缀; 这样后续新增的常规调用只要走包装函数就自动免疫。
3. 补齐绕过包装函数的路径: 可用性探测的 import ci_otel、ci_otel_start() 中的 ci_gpu.py (带 & 的后台进程, 环境变量前缀仍随子进程生效) 以及 _ci_otel_on_exit() flush 的两条分支都单独补上了该变量, 对应 patch 中另外三个 hunk。
4. 限定作用域: 变量只作为 helper 子进程的前缀, 不导出到 shell 全局, 也不影响后续测试命令的环境, 因此对被测代码零影响; OTEL 遥测的采集、记录与 flush 语义完全不变。

5. 测试配套: .buildkite/scripts/ci-otel/tests/test_ci_otel.py 新增

test_shell_wrapper_does_not_write_bytecode_to_helper_tree, 用 shutil.copytree(..., ignore=shutil.ignore_patterns("__pycache__")) 复制出干净 helper 目录, 以 /bin/sh -c 真正 source ci_otel.sh 并执行 ci_otel_start 1 true; ci_otel_finish 0, 通过 CI_INFRA_OTEL_DIR、CI_INFRA_OTEL_SPOOL_DIR 重定向目录, 并设置 CI_INFRA_GPU_SAMPLING=0 关闭 GPU 采样以保证稳定, 最后断言 list(helper_dir.rglob("*.pyc")) == []. 同时新增 import shutil。

关键文件:

- .buildkite/scripts/ci-otel/ci_otel.sh (模块 遥测脚本; 类别 other; 类型 core-logic; 符号 _ci_otel_python, ci_otel_start, _ci_otel_on_exit) : 核心改动文件: 在可用性探测、_ci_otel_python() 包装、ci_otel_start() 的 GPU 采样、_ci_otel_on_exit() flush 共四处调用点补上 PYTHONDONTWRITEBYTECODE=1, 是阻止 root 残留 .pyc 的唯一实质手段。
- .buildkite/scripts/ci-otel/tests/test_ci_otel.py (模块 遥测测试; 类别 test; 类型 test-coverage; 符号 test_shell_wrapper_does_not_write_bytecode_to_helper_tree) : 新增回归测试 test_shell_wrapper_does_not_write_bytecode_to_helper_tree, 在临时干净副本上跑完整 shell 生命周期并断言无 .pyc, 是把「不再污染 checkout」这一约束固化下来的配套测试。

关键符号: _ci_otel_python, ci_otel_start, _ci_otel_on_exit,
test_shell_wrapper_does_not_write_bytecode_to_helper_tree

关键源码片段

.buildkite/scripts/ci-otel/ci_otel.sh

核心改动文件: 在可用性探测、_ci_otel_python() 包装、ci_otel_start() 的 GPU 采样、_ci_otel_on_exit() flush 共四处调用点补上 PYTHONDONTWRITEBYTECODE=1, 是阻止 root 残留 .pyc 的唯一实质手段。

```
# 可用性探测: helper 以 root 身份在挂载 checkout 上 import ci_otel,
# 探测本身也会写 __pycache__, 所以这里同样要带 PYTHONDONTWRITEBYTECODE=1。
if [ -z "${CI_INFRA_OTEL_RUNTIME_DIR}" ] \
  || [ -z "${CI_INFRA_OTEL_PYTHON}" ] \
  || [ ! -f "${CI_INFRA_OTEL_DIR}/ci_otel.py" ] \
  || ! mkdir -p "${CI_INFRA_OTEL_SPOOL_DIR}" \
  || ! PYTHONDONTWRITEBYTECODE=1 \
    PYTHONPATH="${CI_INFRA_OTEL_DIR}${PYTHONPATH:+:${PYTHONPATH}}" \
    "${CI_INFRA_OTEL_PYTHON}" -c "import ci_otel" >/dev/null 2>&1; then
# tracing 不可用时降级为普通执行, 测试命令照常运行
echo "vLLM CI OTel: tracing unavailable; test command will run normally" >&2 || :
fi

# 统一包装函数: 把环境变量收敛在这里, 覆盖 timeout 存在与不存在两条分支,
# 后续常规调用只要走 _ci_otel_python 就自动免疫 .pyc 落盘。
_ci_otel_python() {
  if command -v timeout >/dev/null 2>&1; then
    PYTHONDONTWRITEBYTECODE=1 timeout 2s "${CI_INFRA_OTEL_PYTHON}" "$@"
```

```

else
    PYTHONDONTWRITEBYTECODE=1 "${CI_INFRA_OTEL_PYTHON}" "$@"
fi
}

# GPU 采样绕过了包装函数，直接以后台子进程启动 ci_gpu.py,
# 环境变量前缀会随 & 启动的子进程继承，因此同样不会写字节码。
ci_otel_start() {
    if [ "${CI_INFRA_GPU_SAMPLING:-1}" != "0" ] \
        && [ -f "${CI_INFRA_OTEL_DIR}/ci_gpu.py" ] \
        && command -v nvidia-smi >/dev/null 2>&1; then
        PYTHONDONTWRITEBYTECODE=1 \
            "${CI_INFRA_OTEL_PYTHON}" "${CI_INFRA_OTEL_DIR}/ci_gpu.py" "$$" </dev/null &
        _CI_INFRA_GPU_PID=$!
    fi
}

# 退出钩子里的 flush 是第三类旁路调用，两条分支都要显式加环境变量，
# 否则进程退出时会成为唯一仍会污染 checkout 的路径。
_ci_otel_on_exit() {
    # ...
    trap - 0
    ci_otel_finish "${CI_INFRA_OTEL_EXIT_STATUS}" || :
    if command -v timeout >/dev/null 2>&1; then
        PYTHONDONTWRITEBYTECODE=1 timeout 4s \
            "${CI_INFRA_OTEL_PYTHON}" "${CI_INFRA_OTEL_DIR}/ci_otel.py" flush || :
    else
        PYTHONDONTWRITEBYTECODE=1 \
            "${CI_INFRA_OTEL_PYTHON}" "${CI_INFRA_OTEL_DIR}/ci_otel.py" flush || :
    fi
    # 若 runtime 目录由本进程创建，随后负责清理
}

```

[.buildkite/scripts/ci-otel/tests/test_ci_otel.py](#)

新增回归测试 [test_shell_wrapper_does_not_write_bytecode_to_helper_tree](#)，在临时干净副本上跑完整 shell 生命周期并断言无 .pyc，是把「不再污染 checkout」这一约束固化下来的配套测试。

```

def test_shell_wrapper_does_not_write_bytecode_to_helper_tree(tmp_path):
    # 先把 helper 目录整体复制到临时目录并忽略已有的 __pycache__,
    # 这样「是否新生成 .pyc」才是唯一判定依据，也不会污染真实源码树
    helper_dir = tmp_path / "ci-otel"
    shutil.copytree(
        SCRIPTS_DIR,
        helper_dir,
        ignore=shutil.ignore_patterns("__pycache__"),
    )
    # 真正 source shell 包装脚本，覆盖 ci_otel_start → ci_otel_finish
    # 以及进程退出时 flush 的完整生命周期；只 import Python 模块

```

```

# 无法验证 shell 层是否把环境变量透传给了子进程
shell = (
    f'. "{helper_dir} / "ci_otel.sh"'; '
    f"ci_otel_start 1 {_quoted('true')}; ci_otel_finish 0"
)

result = subprocess.run(
    ["/bin/sh", "-c", shell],
    check=False,
    capture_output=True,
    text=True,
    env={
        **os.environ,
        # 把 helper 目录与 spool 目录都重定向到临时路径
        "CI_INFRA_OTEL_DIR": str(helper_dir),
        # 关闭 GPU 采样，避免依赖 nvidia-smi 造成测试不稳定
        "CI_INFRA_GPU_SAMPLING": "0",
        "CI_INFRA_OTEL_SPOOL_DIR": str(tmp_path / "spans"),
    },
)

assert result.returncode == 0, result.stderr
# 核心断言：整棵 helper 树里不允许出现任何 .pyc
assert list(helper_dir.rglob("*.pyc")) == []

```

评论区精华

本 PR 的 5 条评论全部是作者 khluu 的运维侧进展更新，没有 reviewer 反对意见；唯一的 review 是 claude[bot] 声明因 fork PR 禁用自动审查。讨论的核心不是代码风格，而是「预防性修复无法替代一次性清理」以及失败面不断扩大的事实记录：影响范围从 6 jobs/5 builds 滚动到 13 jobs/12 builds，且出现了第二个与 OTel 无关的 root 所有路径

[vllm/logging_utils/__pycache__/__init__.cpython-312.pyc](#)，作者据此要求「递归清理整个 checkout，而不是只删第一个可见文件」，并在 12:26 UTC 主动暂停 [h200-ci-6-23](#) 12 小时作为运营遏制。作者也坦承修复在目标 slot 上尚无验证证据。

- 预防性修复与一次性清理的边界 (other): 合并代码与恢复 slot 必须分成两步：先 root 递归清理 /chown 整个 checkout 并以 buildkite-agent 验证干净 clone，再把 h200-ci-6-23 放回调度；期间不应继续消耗 job 重试。
- 第二个 root 所有的 stale 路径暴露修复覆盖面不足 (correctness): 本 PR 只闭合了 OTel helper 这一条污染链，非 OTel 的 root 写入源仍未处理，属于遗留问题。
- 失败影响面的滚动统计与运营遏制 (other): 确认是单 slot 级基础设施事故，采取暂停调度作为遏制手段，并要求 resume 前完成 root 清理与验证。
- fork PR 自动审查被禁用 (other): 没有自动化审查覆盖，改动由人工提交者与作者自测（60 个 pytest 用例、shellcheck、pre-commit、sh -n）验证。

风险与影响

- 风险:

1. 预防不等于修复: 作者原话是「PR #56766 prevents new OTel helper bytecode after a clean checkout exists; it cannot repair a checkout that the agent is already unable to delete」, 合并后 13 个 job 的既有失败仍会持续, 直到 h200-ci-6-23 被 root 递归清理或 recycle。
2. 覆盖不完整: 评论中出现的 vllm/logging_utils/__pycache__/__init__.cpython-312.pyc 不在 ci-otel 目录下, 说明除 OTel helper 外还有别的 root 写入源会污染挂载 checkout, 本 PR 只堵住了其中一处, 同类 checkout 清理失败仍可能复发。
3. 测试存在盲区: 新增用例把 CI_INFRA_GPU_SAMPLING=0, 因此 ci_otel_start() 中 ci_gpu.py 那条调用路径并未被真正执行; 可用性探测分支也没有独立断言。若未来有人绕过 _ci_otel_python() 直接调用 Python, 回归测试不一定能拦住。
4. 验证证据缺失: 作者明确表示首个 post-fix main build #88785 未落到 H200 lane, 同 slot 的防复发证据尚待补, 因此该改动在目标环境上属于「理论上正确、尚未实测」。
5. 兼容性风险低: PYTHONDONTWRITEBYTECODE=1 只影响 helper 进程自身的字节码缓存写入, 不改变进程语义与退出码, test_shell_wrapper_preserves_failure_status 等既有失败状态透传测试仍通过。- 影响: 对用户与线上推理服务没有直接影响, 本 PR 只改动 Buildkite 硬件 CI 的遥测 helper 执行环境。对系统而言, 影响集中在 H200 硬件 CI 的 checkout 阶段: 修复后 helper 不再向挂载 checkout 写入 root 所有的 .pyc, checkout 清理与重试可恢复正常。对团队而言, CI 维护者需要完成一次带 root 权限的递归清理 /chown 并验证 buildkite-agent 能干净 clone, 才能让该 slot 重新上线; 所有在该 slot 上被误判为「代码失败」的 PR (如 Rust Frontend Tool Use、V1 Core、V1 Spec Decode 等) 实际都是测试前基础设施失败。长期看, 这次事件暴露出「容器内 root 进程写回宿主主机挂载点」这一类通用 CI 卫生问题, 可能需要更系统的防护策略。- 风险标记: CI 基础设施级阻塞, 预防性修复无法清理已中毒环境, 修复覆盖面不完整 (存在其他 root 写入源), GPU 采样分支未被测试实际执行, 目标 slot 尚无防复发验证证据

关联脉络

- PR #56763 [CI] Update entrypoints CI: 同属 .buildkite CI 配置的集中整治 (test_areas/entrypoints.yaml、test-amd.yaml), 与本 PR 在时间线上相邻, 都是 Buildkite 侧的 CI 卫生与结构调整。
- PR #56671 [CI][Test] Mock CPU backend block sizes in kv_connector unit conftest: 同样是「先让 CI 恢复可跑」的解阻型改动 (修 conftest 解阻 81 个测试), 处置模式与本 PR 刻意区分预防与清理的思路一致。
- PR #54934 [CI][CPU] Add speculative-decoding coverage to CPU CI: 直接修改 .buildkite/hardware_tests 与 .buildkite/scripts 下的硬件 CI 脚本, 与本 PR 同处 Buildkite 硬件 CI 脚本这条演进脉络。