PR #51159 完整报告

vllm-project/vllm

[ROCm] Defer `tilelang` import through its import `from vllm.tilelang_utils import tilelang` and relaxed `has_tilelang`

合并时间：2026-08-14 02:12

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51159

执行摘要

- 一句话：ROCm 延迟 TileLang 导入，修复 HIP 符号污染
- 推荐动作：值得精读。设计上展示了如何在不引入 facade 的前提下用装饰器延迟 JIT，以及如何在 import 检测里避免副作用；对后续在 ROCm 上接入其他重依赖库有参考意义。建议关注两个 follow-up：tilelang/TVM 上游符号修复，和 #51162 中 _has_module 的全面改造。

功能与动机

issue #51151 报告：PR #50879 之后，ROCm 上 import tilelang 会向全局进程符号表注入错误 / 损坏的 hipMalloc/hipFree（来自 libhip_stub.so），导致 AITER 加载其 JIT 模块后出现分配与测试失败。本 PR 旨在从 vLLM 侧规避该问题——任何 ROCm 模块导入路径都不应再触发 tilelang 加载；同时呼应 #51162，_has_module 这类探测函数不应真正导入模块。

实现拆解

1. 新增 vllm/tilelang_utils/__init__.py：集中管理 tilelang/tilelang.language 的导入。CUDA 平台在模块导入时即校验并 eager 导入；ROCm 等其它平台先置 None 占位。_ensure_tilelang_imported() 在首次使用时把真正的模块绑定回全局；_get_pass_configs() 缓存按平台生成的 JIT pass 配置（CUDA 额外加 TL_PTXAS_REGISTER_USAGE_LEVEL=10）。
2. 实现平台分叉的 tilelang_jit 装饰器：非 ROCm 路径保持 tilelang.jit(pass_configs=...) 的 eager 装饰；ROCm 路径返回 wrapper，在第一次调用时导入 tilelang、把 tilelang 与 T 重新注入到被装饰函数所在模块的 __globals__（kernel 函数体以无前缀 T 引用 DSL 符号），再编译并缓存 compiled_kernel。
3. 重接 vllm/model_executor/kernels/mhc/tilelang_kernels.py：删除原先的 has_tilelang 检查、直接 import tilelang 和模块级 pass_configs，8 个 MHC kernel 装饰器统一改为 @tilelang_jit；ENABLE_PDL 逻辑保留。
4. 弱化 has_tilelang 副作用：import_utils.py 新增 @cache _has_module_spec()（仅 find_spec），has_tilelang() 改用它，注释明确调用方必须在使用点自己懒加载。
5. 配套防护与测试：tools/pre_commit/check_forbidden_imports.py 新增 tilelang 禁止规则（含正则用例）；vllm/utils/jit_monitor.py 在 ROCm 上跳过 _setup_tilelang_jit_hook()；新增 tests/kernels/test_mhc_tilelang_jit.py 覆盖 ROCm 懒加载与符号表不被

libhip_stub.so 劫持，tests/jit_monitor/test_hooks.py 对 tilelang 测试追加 ROCm skip 标记。

关键文件：

- vllm/tilelang_utils/__init__.py（模块 懒加载门控；类别 source；类型 dependency-wiring；符号 _ensure_tilelang_imported, _get_pass_configs, tilelang_jit, wrapper）：新增的核心模块，集中 TileLang 导入与 JIT 门控，是 ROCm 懒加载方案的主干。
- tests/kernels/test_mhc_tilelang_jit.py（模块 内核测试；类别 test；类型 test-coverage；符号 _PassConfigKey, _install_tilelang_stub, jit, decorate）：验证 CUDA eager / ROCm lazy 行为差异，并用 dlsym/dladdr 守卫 HIP 符号表不被 tilelang 污染。
- vllm/model_executor/kernels/mhc/tilelang_kernels.py（模块 内核层；类别 source；类型 data-contract；符号 mhc_pre_big_fuse_tilelang, mhc_pre_big_fuse_with_norm_tilelang, mhc_pre_big_fuse_broadcast_with_norm_tilelang, mhc_fused_tilelang）：唯一使用 @tilelang.jit 的生产代码；通过切换装饰器彻底移除启动期 tilelang import。
- vllm/utils/import_utils.py（模块 导入工具；类别 source；类型 core-logic；符号 _has_module_spec, has_tilelang）：has_tilelang 从 trial import 降级为 find_spec，消除探测函数的副作用，呼应 issue #51162。
- tools/pre_commit/check_forbidden_imports.py（模块 提交钩子；类别 infra；类型 configuration）：新增 tilelang 禁止导入规则，保证未来代码统一走 vllm.tilelang_utils。
- vllm/utils/jit_monitor.py（模块 监控模块；类别 source；类型 dependency-wiring；符号 activate）：修复 rasmith 指出的残留导入路径：ROCm 上不再安装 TileLang JIT hook。
- tests/jit_monitor/test_hooks.py（模块 监控测试；类别 test；类型 test-coverage）：为 ROCm 上已禁用的 tilelang JIT 监控测试增加 skip 标记，避免误报。
- docs/assets/contributing/dockerfile-stages-dependency.png（模块 文档资源；类别 other；类型 core-logic）：提交过程中被误改的 Dockerfile 阶段依赖图，最终回滚，无净变更。

关键符号：tilelang_jit, _ensure_tilelang_imported, _get_pass_configs, _has_module_spec, has_tilelang, mhc_post_tilelang, test_tilelang_jit_decorator_is_lazy_only_on_rocm, test_deepseek_v4_import_and_jit_monitor_do_not_hijack_hip_symbols

关键源码片段

vllm/tilelang_utils/__init__.py

新增的核心模块，集中 TileLang 导入与 JIT 门控，是 ROCm 懒加载方案的主干。

```
# vllm/tilelang_utils/__init__.py
# TileLang 的统一导入与 JIT 门控：CUDA 保持 eager，ROCm 延迟到首次调用。

from __future__ import annotations

import functools
```

```
from collections.abc import Callable
from functools import cache
from typing import TYPE_CHECKING, Any

from vllm.platforms import current_platform
from vllm.utils.import_utils import has_tilelang

# CUDA 平台在 import vllm.model_executor.kernels.mhc.tilelang_kernels 时就需要
# TileLang，因此这里保持 eager 导入；ROCm 上先置空占位，避免加载 tilelang 自带的
# 破损 TVM / HIP stub 符号（见 issue #51151）。
if TYPE_CHECKING or current_platform.is_cuda():
    if not has_tilelang():
        raise ImportError(
            "tilelang is required for mhc but is not installed. Install it with "
            "`pip install tilelang`."
        )
    import tilelang
    import tilelang.language as T
else:
    tilelang = None # type: ignore[assignment]
    T = None # type: ignore[assignment]


def _ensure_tilelang_imported() -> None:
    """把 `tilelang` 与 `T` 绑定到本模块全局，必要时才真正导入。

    在 ROCm 上该函数在第一次 kernel 调用时才执行，从而把 tilelang 的
    import 副作用挡在启动路径之外。
    """
    global T, tilelang

    if tilelang is not None:
        return
    if not has_tilelang():
        raise ImportError(
            "tilelang is required for mhc but is not installed. Install it with "
            "`pip install tilelang`."
        )
    import tilelang as tilelang_module
    import tilelang.language as tilelang_language

    tilelang = tilelang_module
    T = tilelang_language


@cache
def _get_pass_configs() -> dict[Any, Any]:
    # 保证取配置前 TileLang 已就绪；CUDA 额外设置 PTXAS 寄存器用量
    _ensure_tilelang_imported()
```

```
    pass_configs: dict[Any, Any] = {
        tilelang.PassConfigKey.TL_DISABLE_WARP_SPECIALIZED: True,
        tilelang.PassConfigKey.TL_DISABLE_TMA_LOWER: True,
    }
    if current_platform.is_cuda():
        pass_configs[tilelang.PassConfigKey.TL_PTXAS_REGISTER_USAGE_LEVEL] = 10
    return pass_configs


def tilelang_jit(kernel_function: Callable[..., Any]) -> Callable[..., Any]:
    """`tilelang.jit` 的封装：ROCm 延迟装饰，CUDA 保持原语义。

    TileLang 解析 kernel 函数体时会引用无前缀全局名 `T`，因此延迟路径上
    编译前必须把 `T` / `tilelang` 重新注入到被装饰函数所在模块的 globals。
    """
    if not current_platform.is_rocm():
        _ensure_tilelang_imported()
        return tilelang.jit(pass_configs=_get_pass_configs())(kernel_function)

    compiled_kernel: Callable[..., Any] | None = None

    @functools.wraps(kernel_function)
    def wrapper(*args: Any, **kwargs: Any) -> Any:
        nonlocal compiled_kernel
        if compiled_kernel is None:
            _ensure_tilelang_imported()
            kernel_function.__globals__["tilelang"] = tilelang
            kernel_function.__globals__["T"] = T
            compiled_kernel = tilelang.jit(pass_configs=_get_pass_configs())(
                kernel_function
            )
        return compiled_kernel(*args, **kwargs)

    return wrapper
```

tests/kernels/test_mhc_tilelang_jit.py

验证 CUDA *eager* / ROCm lazy 行为差异，并用 dlsym/dladdr 守卫 HIP 符号表不被 tilelang 污染。

```
# tests/kernels/test_mhc_tilelang_jit.py（节选，桩 + 懒加载用例）
import importlib
import sys
from types import ModuleType
from typing import Any

import pytest

from vllm.platforms import current_platform
from vllm.utils import import_utils
```

```
class _PassConfigKey:
    # 与真实 tilelang.PassConfigKey 对应的桩常量
    TL_DISABLE_WARP_SPECIALIZED = "disable_warp_specialized"
    TL_DISABLE_TMA_LOWER = "disable_tma_lower"
    TL_PTXAS_REGISTER_USAGE_LEVEL = "ptxas_register_usage_level"


def _install_tilelang_stub(monkeypatch: pytest.MonkeyPatch) -> dict[str, int]:
    # 用桩模块替换真实 tilelang，统计 jit 装饰与编译的次数
    calls = {"jit_decorate": 0, "compiled_call": 0}

    tilelang: Any = ModuleType("tilelang")

    def jit(**kwargs: Any) -> Any:
        def decorate(func: Any) -> Any:
            calls["jit_decorate"] += 1

            def compiled(*args: Any, **kw: Any) -> Any:
                calls["compiled_call"] += 1
                return func.__name__

            return compiled

        return decorate

    tilelang.PassConfigKey = _PassConfigKey
    tilelang.jit = jit

    monkeypatch.setattr(import_utils, "has_tilelang", lambda: True)
    monkeypatch.setitem(sys.modules, "tilelang", tilelang)
    monkeypatch.setitem(
        sys.modules, "tilelang.language", ModuleType("tilelang.language")
    )
    monkeypatch.delitem(sys.modules, "vllm.tilelang_utils", raising=False)

    return calls


def test_tilelang_jit_decorator_is_lazy_only_on_rocm(
    monkeypatch: pytest.MonkeyPatch,
) -> None:
    if not (current_platform.is_cuda() or current_platform.is_rocm()):
        pytest.skip("Test requires CUDA or ROCm")

    calls = _install_tilelang_stub(monkeypatch)
    module_name = "vllm.model_executor.kernels.mhc.tilelang_kernels"
    monkeypatch.delitem(sys.modules, module_name, raising=False)
    module = importlib.import_module(module_name)
```

```
    # ROCm 下 import 模块不应触发 JIT 装饰；CUDA 下必须 eager 装饰
    if current_platform.is_rocm():
        assert calls["jit_decorate"] == 0
    else:
        assert calls["jit_decorate"] > 0

    # 首次调用之后，ROCm 只装饰一次并进入 compiled 缓存
    decorated_calls = calls["jit_decorate"]
    assert module.mhc_post_tilelang() == "mhc_post_tilelang"
    if current_platform.is_rocm():
        assert calls["jit_decorate"] == 1
    else:
        assert calls["jit_decorate"] == decorated_calls
    assert calls["compiled_call"] == 1
```

vllm/utils/import_utils.py

has_tilelang 从 trial import 降级为 find_spec，消除探测函数的副作用，呼应 issue #51162。

```
# vllm/utils/import_utils.py 中新增的轻量探测函数
@cache
def _has_module_spec(module_name: str) -> bool:
    """只解析 import spec，不真正导入模块。

    与 `_has_module` 不同，它不做 trial import，因此不会触发重模块的副作用
    （例如 tilelang 注入破损的 HIP stub 符号），代价是无法验证原生依赖
    （共享库等）是否真的可用。
    """
    try:
        return importlib.util.find_spec(module_name) is not None
    except Exception:
        return False


@cache
def has_tilelang() -> bool:
    """TileLang 是否可用：仅检查 import spec。

    tilelang 的导入开销大且可能有符号污染，调用方必须在自己的使用点再懒加载，
    而不是依赖本函数已经完成导入。
    """
    if not _has_module_spec("tilelang"):
        return False
    # ROCm 特有 guard，延迟导入以免在 CUDA 上加载 rocm 相关代码
    from vllm.platforms import current_platform

    if current_platform.is_rocm():
        from vllm.platforms.rocm import on_gfx1250

        # TODO: tilelang 支持 gfx1250 后放开
```

```
        if on_gfx1250():
            return False
    return True
```

评论区精华

- Isotr0py 建议参照 triton_utils 收敛导入："How about making a tilelang_utils similar to triton_utils?" fxmarty-amd 采纳并进一步要求所有代码必须 from vllm.tilelang_utils import tilelang，与 triton 规则对齐。
- Isotr0py 关心 DX：希望用占位 facade 保留 @tilelang.jit 写法，fxmarty-amd 回应这样做需要 hacky 的 _TileLangFacade，且本 PR 的目标更广——ROCm 启动期完全不 import tilelang；最终以 @tilelang_jit 收场，Isotr0py 批准时留话："Anyway, let's land this to fix the broken model first."
- rasmith 指出 jit_monitor.py 的 _setup_tilelang_jit_hook 仍会导入 tilelang 并导致 test_online_quantization 失败；fxmarty-amd 复现后补上 ROCm 门控，并新增 " 确认 tilelang 不被 import、符号不来自 libhip_stub.so" 的测试。
- tjtanaa 与 Fangzhou-Ai 确认：DeepSeek-V4 实际隐藏层大小 4096/7168 均走 AITER，tilelang mHC 路径在真实推理中不会被触发，因此 ROCm 上跳过 TileLang 是安全的。
- 是否参照 triton_utils 引入统一 tilelang 导入模块 (design): 新增 vllm.tilelang_utils 模块，并通过 pre-commit 禁止直接 import tilelang。
- 是否保留 @tilelang.jit 写法的 DX 权衡 (design): 以 @tilelang_jit 统一收口；Isotr0py 批准时表示先落地修复问题。
- jit_monitor 仍会导入 tilelang 导致失败 (correctness): ROCm 下 activate() 不再调用 _setup_tilelang_jit_hook；增加测试确认 tilelang 未被 import 且 hipFree 来自 libamdhip64.so。
- DeepSeek-V4 是否真的会走 tilelang mHC 路径 (question): 真实推理不触发 tilelang mHC，ROCm 上跳过 tilelang 是安全的。

风险与影响

- 风险：
  - ROCm 首次 kernel 调用才编译，可能出现运行时延迟或失败；若 tilelang 本身损坏（如 gfx1250 等），错误从 import 期推迟到调用期，更难定位；测试仅覆盖 MI300/MI350。
  - kernel_function.__globals__ 重绑定会改变被装饰函数模块的全局命名空间；如果同一模块内其它代码在编译前使用 T/tilelang，可能拿到 None。当前仅影响 mhc/tilelang_kernels.py。
  - _get_pass_configs 是 @cache 的，首次调用发生在编译时；若平台判断依赖运行时环境而非 import 时点，配置可能与平台不符（当前逻辑在 import 时判断，风险低）。
  - CUDA/ROCm 行为分叉（eager vs lazy），同一份 kernel 代码在不同后端的行为差异虽有测试覆盖，但其它调用点如果期望 import 后即可调用工厂函数，行为可能改变。
  - jit_monitor 在 ROCm 上不再报告 tilelang JIT，若未来真正启用 tilelang mHC 路径，会失去运行时监控。

    - pre-commit 新规则会拦截所有直接 import tilelang，已有代码若有直接导入会触发 CI 失败，需迁移。
- 影响：
    - ROCm（MI300/MI325/MI350）用户：修复由 tilelang 符号劫持导致的分配失败与 test_online 回归；启动时间缩短（跳过 tilelang 导入）。
    - DeepSeek-V4 MHC 路径：实际推理仍走 AITER，tilelang 代码路径被推迟到调用层；若 tilelang JIT 被真正触发，单次首调编译延迟增加。
    - 开发侧：新增统一的 tilelang 导入门控和 lint 规则，后续 kernel 开发者须遵循 from vllm.tilelang_utils import tilelang, T。
    - 测试与 CI：新增 ROCm 符号表守卫测试，tilelang 相关 jit_monitor 测试在 ROCm 上跳过；AMD CI 覆盖 MI300/MI350 验证。
    - 风险标记：ROCm 首调 JIT 延迟，模块全局重绑定副作用，临时关闭 JIT 监控，平台行为分叉，依赖上游修复

关联脉络

- 暂无明显关联 PR