

PR #54346 完整报告

vllm-project/vllm

[Bugfix][Multimodal] Release Qwen2.5-VL and Qwen3-VL RoPE caches with the model

合并时间: 2026-08-30 11:10

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54346>

执行摘要

- 一句话: Qwen2.5/3-VL RoPE 缓存改实例级, 修复 shutdown 显存驻留
- 推荐动作: 值得精读, 建议按以下视角阅读:
 1. 重点学习“实例方法被类级 lru_cache 装饰导致缓存以 self 为键强引用实例”这一隐蔽生命周期陷阱的定位方法 (PR body 中展示了 AI 辅助引用链分析的完整路径)。
 2. 关注“bounded per-instance LRUCache 替代类级缓存”的通用范式: 用容量上限防止无限增长, 用实例属性保证随所有者释放, 是 Python 缓存生命周期管理的标准解法, 可迁移到其他模型的同类问题。
 3. 参考 review 中“宁删不烂”的测试取舍: 低价值测试 (手动装配对象导致无法捕捉真实回归) 不如删除, 避免虚假的安全感。

阅读顺序建议先看 `qwen3_vl.py` 的 `rot_pos_ids` 改动理解模式, 再看 `qwen2_5_vl.py` 的 `get_rope_by_thw` 确认落地细节。

功能与动机

PR body 明确目标是修复“deterministic GPU memory retention when Qwen2.5-VL or Qwen3-VL is shut down in-process after serving a multimodal request”。根因有二:
`Qwen2_5_VisionTransformer.get_rope_by_thw` 是实例方法却用类级 `functools.lru_cache` 装饰, `self` 进入缓存键使缓存包装器强引用每个处理过请求的 vision tower;
`Qwen3_VisionTransformer.rot_pos_ids` 用类级静态 `lru_cache`, 缓存张量存活到解释器退出。
PR body 还说明其与 #54162 (top-level model 持有)、#54246 (layer-bound KV cache) 是互补修复, 共同覆盖 in-process shutdown 场景的多类内存泄漏源。

实现拆解

1. 修复范围确定: 作者最初只提交 Qwen2.5-VL 的修复, reviewer DarkLight1337 在 issue 评论中提出“Let's fix this for Qwen3-VL too”, 第 2 个 commit 补上 `qwen3_vl.py` 的对称改动。
2. Qwen2.5-VL 改造 (`vllm/model_executor/models/qwen2_5_vl.py`): 删除 `from functools import lru_cache`, 改为 `from functools import partial`, 新增 `from vllm.utils.cache import LRUCache`; 在 `Qwen2_5_VisionTransformer.__init__` 中新增 `self._rope_by_thw_cache = LRUCache(capacity=1024)`; `get_rope_by_thw` 去掉 `@lru_cache(maxsize=1024)` 装饰器, 改为显式缓存键 (t, h, w) 的命中判断、计算、写入

三步。

3. Qwen3-VL 改造 (vllm/model_executor/models/qwen3_vl.py) : 同样引入 LRUCache, 在 Qwen3_VisionTransformer.__init__ 中新增 self._rot_pos_ids_cache = LRUCache(capacity=1024); rot_pos_ids 从 @staticmethod + @lru_cache(maxsize=1024) 改为实例方法, 缓存键 (h, w, spatial_merge_size) 与原静态缓存键一致; 调用方 rot_pos_emb 中 self.rot_pos_ids(...) 的调用方式不变, 对上层完全透明。
4. 行为一致性保证: 缓存键、1024 容量上限、命中返回同一 tensor 对象 (is 判断成立) 等语义均与原来相同; 差异仅在生命周期归属——从“类级 / 解释器级”变为“实例级”, 多实例场景下各实例首次遇到相同键会各自计算一次, 换取模型删除时缓存随之释放。
5. 测试演进与最终取舍: 第 3 个 commit 曾在 tests/models/test_language_model_cache_is_weak.py 新增弱引用回归测试, reviewer 认为通过 object.__new__ 手动装配属性使测试无法捕捉上游初始化回归、且构造过于复杂, 第 4 个 commit 删除全部测试; 验收依靠 PR body 中的端到端 GPU 实验 (A100 双卡、ModelScope 真权重、del + gc.collect() + torch.cuda.empty_cache() 链路)。

关键文件:

- vllm/model_executor/models/qwen3_vl.py (模块 视觉编码器; 类别 source; 类型 core-logic; 符号 rot_pos_ids, _rot_pos_ids_cache, Qwen3_VisionTransformer.init) : Qwen3-VL 视觉编码器的核心修复文件: rot_pos_ids 从 @staticmethod + @lru_cache(maxsize=1024) 改为实例方法, 并在 __init__ 中引入 self._rot_pos_ids_cache = LRUCache(capacity=1024), 让缓存张量随模型实例释放。调用方 rot_pos_emb 无需改动。
- vllm/model_executor/models/qwen2_5_vl.py (模块 视觉编码器; 类别 source; 类型 core-logic; 符号 get_rope_by_thw, _rope_by_thw_cache, Qwen2_5_VisionTransformer.init) : Qwen2.5-VL 视觉编码器的核心修复文件: get_rope_by_thw 移除 @lru_cache(maxsize=1024) 装饰器并改用手动缓存逻辑, __init__ 中新增 self._rope_by_thw_cache = LRUCache(capacity=1024), 解决实例方法经类级缓存被强引用的问题。

关键符号: get_rope_by_thw, rot_pos_ids

关键源码片段

vllm/model_executor/models/qwen3_vl.py

Qwen3-VL 视觉编码器的核心修复文件: rot_pos_ids 从 @staticmethod + @lru_cache(maxsize=1024) 改为实例方法, 并在 __init__ 中引入 self._rot_pos_ids_cache = LRUCache(capacity=1024), 让缓存张量随模型实例释放。调用方 rot_pos_emb 无需改动。

qwen3_vl.py 中的关键改动:

- `\_\_init\_\_` 新增实例级缓存, 替代原类级静态 `lru\_cache`

- `rot\_pos\_ids` 由 `@staticmethod` 改为实例方法, 缓存随实例释放

```
class Qwen3_VisionTransformer(nn.Module):
```

```

def __init__(
    self,
    vision_config: Qwen3VLVisionConfig,
    norm_eps: float = 1e-6,
    quant_config: QuantizationConfig | None = None,
    prefix: str = "",
) -> None:
    super().__init__()
    ...
    self.num_grid_per_side = int(self.num_position_embeddings**0.5)
    # 新增实例级 LRU 缓存：把 RoPE 位置张量的生命周期绑定到模型实例，
    # 替代原类级静态 `lru_cache`，容量保持 1024 项不变
    self._rot_pos_ids_cache = LRUCache(capacity=1024)
    ...

def rot_pos_ids(self, h: int, w: int, spatial_merge_size: int) -> torch.Tensor:
    # 缓存键与原静态缓存完全一致，命中行为（返回同一 tensor 对象）保持不变
    cache_key = (h, w, spatial_merge_size)
    if cache_key in self._rot_pos_ids_cache:
        return self._rot_pos_ids_cache[cache_key]

    # 生成二维旋转位置索引：先广播出行 / 列索引，
    # 再按 `spatial_merge_size` 分块转置，模拟 mRoPE 的二维网格展开
    hpos_ids = np.broadcast_to(np.arange(h).reshape(h, 1), (h, w))
    h_div = h // spatial_merge_size
    w_div = w // spatial_merge_size
    hpos_ids = hpos_ids.reshape(
        h_div,
        spatial_merge_size,
        w_div,
        spatial_merge_size,
    )
    hpos_ids = hpos_ids.transpose(0, 2, 1, 3)
    hpos_ids = hpos_ids.flatten()

    wpos_ids = np.broadcast_to(np.arange(w).reshape(1, w), (h, w))
    wpos_ids = wpos_ids.reshape(
        h_div,
        spatial_merge_size,
        w_div,
        spatial_merge_size,
    )
    wpos_ids = wpos_ids.transpose(0, 2, 1, 3)
    wpos_ids = wpos_ids.flatten()

    # 合并为 `[h * w, 2]` 的位置张量并写入实例级缓存
    result = torch.from_numpy(np.stack([hpos_ids, wpos_ids], axis=-1))
    self._rot_pos_ids_cache[cache_key] = result
    return result

```

```

def rot_pos_emb(self, grid_thw: list[list[int]]):
    # 调用方维持 `self.rot_pos_ids(...)` 原样不变,
    # 对上层语义透明, 无需任何外部适配
    max_grid_size = max(max(h, w) for _, h, w in grid_thw)
    pos_ids = [
        self.rot_pos_ids(h, w, self.spatial_merge_size)
        if t == 1
        else self.rot_pos_ids(h, w, self.spatial_merge_size).repeat(t, 1)
        for t, h, w in grid_thw
    ]
    ...

```

vllm/model_executor/models/qwen2_5_vl.py

Qwen2.5-VL 视觉编码器的核心修复文件: `get_rope_by_thw` 移除

`@lru_cache(maxsize=1024)` 装饰器并改用手动缓存逻辑, `__init__` 中新增

`self._rope_by_thw_cache = LRUCache(capacity=1024)`, 解决实例方法经类级缓存被强引用的问题。

qwen2_5_vl.py 中的关键改动:

- # - 原 ``get_rope_by_thw`` 是实例方法但被类级 ``@lru_cache(maxsize=1024)`` 装饰,
- # ``self`` 进入缓存键导致类缓存强引用每个 vision tower, shutdown 后无法释放
- # - 改为显式检查实例级 LRUCache, 缓存跟着模型走

```

class Qwen2_5_VisionTransformer(nn.Module):

```

```

    def __init__(
        self,
        vision_config: Qwen2_5_VLVisionConfig,
        norm_eps: float = 1e-6,
        quant_config: QuantizationConfig | None = None,
        prefix: str = "",
    ) -> None:
        super().__init__()
        ...
        self.fullatt_block_indexes = vision_config.fullatt_block_indexes
        self.spatial_merge_unit = self.spatial_merge_size**2
        # 新增实例级 LRU 缓存, 容量与原 `@lru_cache(maxsize=1024)` 一致;
        # 原装饰器使 `self` 进入缓存键, 导致类缓存强引用模型实例
        self._rope_by_thw_cache = LRUCache(capacity=1024)
        ...

```

```

    def get_rope_by_thw(self, t, h, w):
        cache_key = (t, h, w)
        if cache_key in self._rope_by_thw_cache:
            return self._rope_by_thw_cache[cache_key]

```

```

        window_index_thw, cu_seqlens_window_thw = self.get_window_index_thw(t, h, w)
        cos_thw, sin_thw = self.rotary_pos_emb_thw(t, h, w)

```

```

# 按 window 索引重排 `cos_thw` 和 `sin_thw`, 并展平为每 token 的 rope 值
window_index_thw_dev = window_index_thw.to(cos_thw.device, non_blocking=True)
cos_thw = cos_thw[window_index_thw_dev, :, :]
cos_thw = cos_thw.flatten(start_dim=0, end_dim=1)
sin_thw = sin_thw[window_index_thw_dev, :, :]
sin_thw = sin_thw.flatten(start_dim=0, end_dim=1)

cu_seqlens_thw = torch.repeat_interleave(
    torch.tensor([h * w], dtype=torch.int32), t
)

# 一次性写入全部中间结果, 保证同一 `(t, h, w)` 命中时返回同一批 tensor
result = (
    cos_thw,
    sin_thw,
    window_index_thw,
    cu_seqlens_window_thw,
    cu_seqlens_thw,
)
self._rope_by_thw_cache[cache_key] = result
return result

```

评论区精华

审查的核心交锋集中在测试价值上：

- DarkLight1337 指出新增的弱引用测试无实际价值：“These tests are not useful as you manually set the attribute anyway. If the upstream modification changes, the test would still pass”。因为测试用手动 `object.__new__` 并直接赋值 `_rope_by_thw_cache`，即使未来实现把缓存改回类级或改名，测试依然通过，属于“假回归保护”。
- 作者回应称已改为正常构造视觉模型，确保缓存初始化变化时测试会失败。
- DarkLight1337 进一步裁定：“I think this is too complicated tbh. Let's just drop the tests”，作者照做并删除全部测试。
- 另外 DarkLight1337 在评论中要求“Let's fix this for Qwen3-VL too”，直接扩大了修复覆盖范围，成为本 PR 包含两个模型的来源。
- 作者还在评论中说明已补齐 DCO sign-off 并请 reviewer 重新启用 auto-merge，属于首次贡献的流程性沟通。
- 弱引用回归测试的有效性质疑 (testing): 作者先回应称已改为正常构造模型，但 reviewer 仍认为复杂度偏高，最终删除全部新增测试。
- 测试删除的取舍 (design): 作者按建议移除测试，PR 最终仅保留两个源文件改动，验收依赖 PR body 中的端到端 GPU 测量。
- 修复范围扩展到 Qwen3-VL (question): 作者在第 2 个 commit 中补充 `qwen3_vl.py` 的对称改动，`rot_pos_ids` 改为实例方法并绑定实例缓存。

风险与影响

- 风险:

1. 无自动化测试兜底: 最终 PR 只改动两个源文件, 没有任何测试文件保留。若未来有人重新引入类级 `lru_cache` (例如为省事加回装饰器), CI 不会失败, 属于纯纪律性回归风险。
2. `LRUCache` 线程安全性未验证: 原 `functools.lru_cache` 底层有锁保护; 替换为 `vllm/utils/cache.py` 的 `LRUCache` 后, `__contains__` 与 `__setitem__` 是分步操作, 若 vision encoder 在多线程中并发调用 `get_rope_by_thw / rot_pos_ids`, 存在 check-then-act 竞态窗口。vLLM 中 encoder 执行路径通常单线程, 但此点未在 PR 中讨论。
3. 多实例重复计算: Qwen3-VL 原来跨实例共享同一 `rot_pos_ids` 张量, 改为实例级缓存后, 每个新实例首次遇到相同 (h, w) 组合都要重算一次位置张量; 多引擎 / 多模型实例场景下有少量额外 CPU 计算。
4. 编译路径未覆盖: `get_rope_by_thw` 位于可能被 `should_torch_compile_mm_encoder` 编译的 encoder 路径 (见 `support_torch_compile` 装饰器), 实例方法化与显式字典式缓存逻辑在 `torch.compile` 下的行为没有测试覆盖。

- 影响:

1. 用户 / 平台侧: 修复的核心场景是 in-process 多次启停 (多模型切换、批处理平台、复用 LLMEngine 等)。实测 Qwen2.5-VL-3B 关闭后保留 CUDA 分配从 1,354,216,448 B 降至 10,617,344 B (缩减 99.22%, 约 1.251 GiB), vision module 不再存活; Qwen3-VL-2B 关闭后保留 279,577,088 B, vision module 与缓存 tensor 均被回收。对长驻服务无运行时影响。
2. 系统侧: 与关联 PR #54246 (layer-bound KV cache 释放)、#54162 (top-level model 持有释放) 共同构成“in-process shutdown 完整释放 GPU 内存”的系列修复, 覆盖三类泄漏源。
3. 团队侧: 改动小、模式清晰、可读性好, 但零测试留存意味着未来依赖 reviewer 的代码审查纪律; 该模式 (类级缓存改实例缓存) 可作为后续类似泄漏修复的参考模板。- 风险标记: 缺少自动化测试, 核心推理路径变更, `LRUCache` 线程安全性未验证, 多实例场景重复计算, `torch.compile` 路径未覆盖

关联脉络

- PR #54246 [Bugfix][MRV2] Release layer-bound KV cache memory in shutdown(): 同一 in-process shutdown 内存释放主题线, PR body 明确说明两者互补: 该 PR 清理 runner 持有的 layer-bound KV cache 绑定, 本 PR 移除 Qwen 类级 RoPE 缓存这一独有泄漏源。
- PR #53808 [Bugfix][Multimodal] Honor modality-scoped mm_processor_kwargs: 同为 Qwen3-VL 多模态正确性修复, 改动同一文件 `vllm/model_executor/models/qwen3_vl.py`, 反映该模型近期多模态路径的持续加固。
- PR #54242 [Frontend] Add video embeds input support: Qwen3-VL 前端 video_embeds 功能扩展, 同样触及 `qwen3_vl.py`, 与本 PR 处于同一模型的功能演进线。
- PR #53529 [Test][Qwen3-VL] Cover compiled DeepStack input contract: Qwen3-VL DeepStack 编译契约测试, 反映 Qwen3-VL 近期活跃的编译与回归保障工作, 与本 PR 的视觉编码器路径改动相关。