

PR #54284 完整报告

vllm-project/vllm

[Bugfix][V1] Keep an encoder cache entry until its last occurrence is freed

合并时间: 2026-08-29 13:08

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54284>

执行摘要

- 一句话: 修复多轮对话中重复图像被提前释放引发编码器重算
- 推荐动作: 值得精读: 此 PR 揭示了缓存管理中以请求为单位计数引用的设计缺陷, 修复手法简洁有效。可借鉴在类似缓存计数场景中, 按实际使用位置持有引用, 避免过早释放。

功能与动机

PR body 明确指出: 在多层视觉对话中, 聊天模板会为每一轮内联同一图像, 导致单个请求携带多个共享同一 `mm_hash` 的特征, 而 `cached` 以请求计数引用, 导致第一个出现位置释放后引用即被丢弃。这会使条目过早进入 `freeable`, 进而可能被其他请求驱逐, 后续出现位置需重新编码, 造成不必要的编码器计算和缓存抖动。虽然不影响输出正确性, 但浪费资源。

实现拆解

变更集中在 `vllm/v1/core/encoder_cache_manager.py` 的 `free_encoder_input` 方法, 并在测试文件中添加两个针对性测试。

1. 修改 `free_encoder_input` 引用释放逻辑: 在释放请求对某 `mm_hash` 的引用前, 检查该请求是否还有剩余的缓存输入 `id` 映射到同一 `mm_hash`。若存在, 则跳过此次释放, 保持引用有效; 若不存在, 则按原逻辑释放并可能将条目移入 `freeable`。
2. 新增测试用例:
 - `test_duplicate_mm_hash_stays_referenced_until_last_free`: 验证释放第一个出现位置后引用保持、`freeable` 不变; 释放第二个后条目才被释放。
 - `test_duplicate_mm_hash_is_not_evicted_before_last_use`: 验证条目仍被需要时, 竞争请求无法将其作为可回收容量, 后续出现位置仍能命中缓存, 且不会通知 `worker` 释放。
3. 测试范围: 仅修改 `tests/v1/core/test_encoder_cache_manager.py`, 无配置或部署变动。

关键文件:

- `vllm/v1/core/encoder_cache_manager.py` (模块 缓存管理器; 类别 `source`; 类型 `core-logic`; 符号 `free_encoder_input`): 核心修复文件, 修改了 `free_encoder_input` 中引用释放逻辑, 避免重复多模态项被提前释放。
- `tests/v1/core/test_encoder_cache_manager.py` (模块 缓存管理器; 类别 `test`; 类型 `test-coverage`; 符号 `test_duplicate_mm_hash_stays_referenced_until_last_free`, `test_duplicate_mm_hash_is_not_evicted_before_last_use`): 新增两个测试用例, 验证重

复多模态项在最后一次释放前不会被释放或驱逐，覆盖了修复的核心场景。

关键符号：free_encoder_input

关键源码片段

vllm/v1/core/encoder_cache_manager.py

核心修复文件，修改了 free_encoder_input 中引用释放逻辑，避免重复多模态项被提前释放。

vllm/v1/core/encoder_cache_manager.py (部分)

```
def free_encoder_input(self, request: Request, input_id: int) -> None:
```

```
    """释放请求对某个编码器输入 (mm_data) 的引用。
```

```
    当对应 mm_hash 的引用集合为空时，将其加入 freeable 并更新 num_freeable_slots。
```

```
    条目只有在容量不足时（如 can_allocate 时）才会被真正释放。
```

```
    """
```

```
    req_id = request.request_id
```

```
    mm_hash = request.mm_features[input_id].identifier
```

```
    # 无论 mm_hash 是否已被驱逐，都要清理 request_cached_ids
```

```
    if req_id in self.request_cached_ids:
```

```
        self.request_cached_ids[req_id].discard(input_id)
```

```
        if not self.request_cached_ids[req_id]:
```

```
            del self.request_cached_ids[req_id]
```

```
    # 若 mm_hash 不在缓存中或引用集合为空，直接返回
```

```
    if not self.cached.get(mm_hash, None):
```

```
        return
```

```
    # 关键修复：cached 以请求为单位计数引用，而非按位置计数，
```

```
    # 因此一个请求重复使用同一多模态项（如多轮对话中携带同一图像）
```

```
    # 只会持有一个引用覆盖所有出现位置。
```

```
    # 如果在第一个位置释放时即丢弃引用，条目会过早变为可回收，
```

```
    # 而请求后续位置仍需要它，可能导致编码器重算。
```

```
    # 因此，只有当该请求不再有任何剩余缓存输入映射到同一 mm_hash 时，
```

```
    # 才真正释放引用。
```

```
    if any(
```

```
        request.mm_features[other_id].identifier == mm_hash
```

```
        for other_id in self.request_cached_ids.get(req_id, ())
```

```
    ):
```

```
        return
```

```
    self.cached[mm_hash].discard(req_id)
```

```
    if not self.cached[mm_hash]:
```

```
        num_encoder_embeds = request.get_num_encoder_embeds(input_id)
```

```
        self.freeable[mm_hash] = num_encoder_embeds
```

```
        self.num_freeable_slots += num_encoder_embeds
```

tests/v1/core/test_encoder_cache_manager.py

新增两个测试用例，验证重复多模态项在最后一次释放前不会被释放或驱逐，覆盖了修复的核心场景。

```
# tests/v1/core/test_encoder_cache_manager.py (部分)
```

```
def test_duplicate_mm_hash_stays_referenced_until_last_free():
```

```
    """验证：cached 以请求为单位计数引用，因此释放第一个出现位置时不应释放条目。"""
```

```
    manager = EncoderCacheManager(cache_size=20)
```

```
    req = MockRequest("r1", ["imgA", "imgA"], [4, 4])
```

```
    manager.allocate(req, 0)
```

```
    assert manager.check_and_update_cache(req, 1) # 第二个位置命中缓存
```

```
    manager.free_encoder_input(req, 0) # 释放第一个位置
```

```
    # 引用应仍然存在，且未进入 freeable
```

```
    assert manager.cached["imgA"] == {"r1"}
```

```
    assert "imgA" not in manager.freeable
```

```
    assert manager.num_freeable_slots == 16
```

```
    manager.free_encoder_input(req, 1) # 释放最后一个位置
```

```
    # 此时引用被释放，条目进入 freeable
```

```
    assert not manager.cached["imgA"]
```

```
    assert manager.freeable["imgA"] == 4
```

```
    assert manager.num_freeable_slots == 20
```

```
def test_duplicate_mm_hash_is_not_evicted_before_last_use():
```

```
    """验证：重复项在最后一个位置使用前不应成为可回收容量。"""
```

```
    manager = EncoderCacheManager(cache_size=8)
```

```
    req = MockRequest("r1", ["imgA", "imgA"], [4, 4])
```

```
    other = MockRequest("r2", ["imgB"], [8])
```

```
    manager.allocate(req, 0)
```

```
    assert manager.check_and_update_cache(req, 1)
```

```
    manager.free_encoder_input(req, 0) # 释放第一个位置
```

```
    # 其他请求不应能分配，因为条目仍被占用
```

```
    assert not manager.can_allocate(other, 0, int(1e9), 0)
```

```
    # 后续位置仍能命中缓存，且不会通知 worker 释放
```

```
    assert manager.check_and_update_cache(req, 1)
```

```
    assert manager.get_freed_mm_hashes() == []
```

评论区精华

review 过程较为简洁，主要动作是 CI 触发和批准。PR 作者 Hotragn 描述了问题根因和修复方案，维护者 njhill 表达感谢并批准合并。没有实质性技术讨论或争议。

- CI 触发与批准 (other): 无分歧，直接合并。

风险与影响

- 风险：该变更影响多模态输入的缓存生命周期，主要风险包括：
 - 若 `request_cached_ids` 与 `cached` 状态不一致，可能导致引用计数不准确，使条目无法及时释放，略微增加内存占用。
 - 引入的 `any()` 扫描在请求缓存输入 id 较多时可能带来微小性能开销，但 PR 作者指出该检查仅在释放路径上执行，且缓存 id 数量通常很少，影响可忽略。
 - 未发现明显的正确性风险，测试覆盖了关键场景。
 - 影响：影响范围限于 V1 引擎的多模态缓存管理。修复后，多轮视觉对话中的重复图像不会被过早释放，减少编码器重复计算和缓存抖动，提升此类请求的推理效率。对单次使用图像的请求无影响。团队维护成本低，逻辑小且清晰。
- 风险标记：影响多模态缓存生命周期，引入微小扫描开销

关联脉络

- PR #54162 [Bugfix][MRV2] Release model and KV cache on in-process engine shutdown: 同属 V1 引擎的缓存生命周期管理改进，关注资源释放时机。