PR #34067 完整报告

sgl-project/sglang

[Bugfix] Fix batched KV free aliasing

合并时间：2026-08-08 18:34

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34067

执行摘要

- 一句话：修复延迟释放队列持有可变视图导致的 KV 错放
- 推荐动作：值得精读，核心价值不在代码量（+60/-44）而在根因定位与设计决策：把“入队即拥有”的不变量放在 allocator 边界，而不是让每个调用方自证视图安全，从而同时保留批量释放收益与视图安全性。建议关注两点：_copy_for_free_group 的约定是否会被后续 hardware backend 复用；是否值得用机制（如注册表检查或 debug 断言）替代人工约定，防止未来 allocator 漏掉拷贝导致静默回归。

功能与动机

PR body 明确给出了根因链路：free group 队列持有基于可变 request-to-token 行的 tensor 视图，cache_unfinished_req 在 radix-cache rematching 时会覆写该行，早于 free_group_end() 消费队列，于是队列里残留的是替换后的 cached/tree KV indices 而非请求原有的 KV 槽位。其后果正如 PR 所述："This can make radix-cache eviction accounting diverge from physical allocator state: entries are counted as logically evicted while their SWA mappings remain allocated. Eventually a request can fail allocation even though the cache reports ample evictable capacity." 要解决的不是“能不能批量释放”，而是“批量释放队列必须拥有自己的索引快照”，因此修复原则被定为："The ownership rule lives at the allocator boundary so callers can safely pass views."

实现拆解

修复按以下 4 步落地：

1. 在分配器基类建立所有权规则：python/sglang/srt/mem_cache/allocator/base.py 新增静态方法 _copy_for_free_group(free_index)，实现即 free_index.clone()。这是整个修复的单一入口约定：所有 deferred free 队列在入队时立即拷贝输入，调用方可以放心传入任何视图（例如 req_to_token 行的切片），无需关心该行后续是否被覆写。
2. 将规则推广到全部 allocator 的延迟释放路径，覆盖以下源码文件：

文件	修改点
allocator/token.py	free 的 free-group 分支改为先 _copy_for_free_group 再入队

文件	修改点
allocator/paged.py	free 与 free_segment 的 group 分支均拷贝入队；free_segment 只拷贝页代表元（stride 切片）
allocator/swa.py	free / free_swa（含 PureSWA 子类）的 group 分支拷贝入队
allocator/hisparse.py	两处 free 的 group 分支拷贝入队
multi_ended_allocator.py	三个子分配器 free 的 group 分支拷贝入队
hardware_backend/npu/allocator_npu.py	NPU 分配器 free 的 group 分支拷贝入队

此前这些位置都是 self.free_group.append(free_index) 直接持有视图，统一改为入队即 clone() 后，批量化释放（free group）机制本身保持不变，只是语义变为"入队即拥有"。

1. 恢复 RadixCache.cache_unfinished_req 的原始语义：python/sglang/srt/mem_cache/radix_cache.py 中，该函数此前为避免别名问题，改传 out-of-place 的 values 副本；所有权规则就位后回改为直接传原始 kv_indices[req.cache_protected_len:new_prefix_len] 视图，既消除一次多余拷贝，又保证释放的始终是请求原有的 KV 槽位。
2. 测试配套（3 个测试文件同步加固）：test_radix_cache_unit.py 把原先的 mock DeferredFreeAllocator 换成真实 TokenToKVPoolAllocator，用 available_size() 与 free_pages 断言"释放的确实是原始 request_indices"；test_swa_unittest.py 在 enqueue 后对原索引执行 zero_() 再 free_group_end()，验证队列已持有自己的快照；test_paged_free_segment.py 新增 test_group_owns_deferred_page_representatives，在 free_segment 后把 row 清零再释放，验证页代表元未受行覆写影响。

关键文件：

- python/sglang/srt/mem_cache/allocator/base.py（模块 分配器基类；类别 source；类型 core-logic；符号 _copy_for_free_group, free_group_begin, free_group_end）：修复的核心入口：新增静态方法 _copy_for_free_group，确立"延迟释放队列入队即 clone"的所有权规则，是所有 allocator 修改所依赖的单一约定。
- python/sglang/srt/mem_cache/radix_cache.py（模块 前缀缓存；类别 source；类型 core-logic；符号 cache_unfinished_req）：实际触发别名 bug 的调用方：cache_unfinished_req 恢复为传原始 kv_indices 视图释放旧前缀，依赖新所有权规则才能保证释放的是请求原有 KV 槽位。
- python/sglang/srt/mem_cache/allocator/paged.py（模块 分页分配；类别 source；类型 core-logic；符号 free, free_segment）：free 与 free_segment 两条延迟释放路径都需拷贝入队，其中 free_segment 只拷贝页代表元（stride 切片），是该文件的关键改动点。
- python/sglang/srt/mem_cache/allocator/swa.py（模块 滑窗分配；类别 source；类型 core-logic；符号 free, free_swa）：SWA 分配器的 free / free_swa（含 PureSWA 子类）共 4 处 group 分支需拷贝入队；SWA 映射残留正是 PR 描述的逻辑 / 物理状态分叉的直接

表现。

- python/sglang/srt/mem_cache/multi_ended_allocator.py（模块 多端分配；类别 source；类型 core-logic；符号 free）：多端（unified）分配器内三个子分配器的 free 方法均需遵循同一所有权规则，是改动面最大的单一文件之一。
- python/sglang/srt/mem_cache/allocator/hisparse.py（模块 稀疏分配；类别 source；类型 core-logic；符号 free）：HiSparse 分配器的两处 free 方法同步应用所有权规则，保证其 logical 分配器在延迟释放时同样不受视图覆写影响。
- python/sglang/srt/hardware_backend/npu/allocator_npu.py（模块 NPU 后端；类别 source；类型 core-logic；符号 free）：NPU 后端分配器同步应用所有权规则；该 PR 带有 npu 标签且需通过 NPU CI，说明此路径在 NPU 设备上同样会触发。
- python/sglang/srt/mem_cache/allocator/token.py（模块 Token 分配；类别 source；类型 core-logic；符号 free）：Token 分配器是最基础的 allocator，cache_unfinished_req 正是调用它的 free_segment；其 group 分支也必须遵守所有权规则。
- test/registered/unit/mem_cache/test_radix_cache_unit.py（模块 缓存测试；类别 test；类型 test-coverage；符号 test_cache_unfinished_req_deferred_free_owns_original_indices, TokenToKVPoolAllocator）：核心复现测试：把 mock DeferredFreeAllocator 换成真实 TokenToKVPoolAllocator，断言释放的是原始 request_indices 且 req_to_token 行已更新为 tree_indices，直接验证修复效果。
- test/registered/unit/mem_cache/test_swa_unittest.py（模块 滑窗测试；类别 test；类型 test-coverage；符号 test_free_swa_group_owns_deferred_indices）：SWA 所有权测试：enqueue 后对原索引执行 zero_() 再 free_group_end()，证明 swa_free_group 持有的是自己的快照而非外部视图的别名。
- test/registered/unit/mem_cache/test_paged_free_segment.py（模块 分页释放测试；类别 test；类型 test-coverage；符号 test_group_owns_deferred_page_representatives）：paged free_segment 的所有权测试：free_segment 入队后把 row 清零再 free_group_end()，验证释放的页代表元与预期一致。

关键符号：_copy_for_free_group, free, free_segment, free_swa, cache_unfinished_req, free_group_begin, free_group_end, test_cache_unfinished_req_deferred_free_owns_original_indices, test_free_swa_group_owns_deferred_indices, test_group_owns_deferred_page_representatives

关键源码片段

python/sglang/srt/mem_cache/allocator/base.py

修复的核心入口：新增静态方法 _copy_for_free_group，确立“延迟释放队列入队即 clone”的所有权规则，是所有 allocator 修改所依赖的单一约定。

```
def free_group_begin(self):
    # 进入批量化延迟释放：关闭立即归还，并重置队列
    self.is_not_in_free_group = False
    self.free_group = []

def free_group_end(self):
    # 退出批量模式：把队列里的索引一次性交还底层分配器
```

```
        self.is_not_in_free_group = True
        if self.free_group:
            self.free(torch.cat(self.free_group))


    @staticmethod
    def _copy_for_free_group(free_index: torch.Tensor) -> torch.Tensor:
        # 所有权规则：延迟释放队列入队时立即 clone，这是本 PR 的单一约定。
        # RadixCache 等调用方传入的往往是 req_to_token 行的切片视图，
        # 在 free_group_end() 消费之前该行可能被 radix rematch 覆写；
        # 在 allocator 边界即刻拷贝，可让调用方放心传视图而不破坏批量释放。
        # 注意：任何 allocator 的 free-group 分支都必须经过这里，否则别名 bug 会回归。
        return free_index.clone()
```

python/sglang/srt/mem_cache/radix_cache.py

实际触发别名 bug 的调用方：cache_unfinished_req 恢复为传原始 kv_indices 视图释放旧前缀，依赖新所有权规则才能保证释放的是请求原有 KV 槽位。

```
    # cache_unfinished_req 尾部：radix match 出新前缀后，把被替换的旧前缀 KV 槽
    # 交给分配器延迟释放。
    new_prefix_len = result.prefix_len

    # 修复前这里传的是 out-of-place 的 values 副本（values[req.cache_protected_len : new_prefix_
    len]），
    # 因为 free-group 队列只持有视图，req_to_token 行随后会被 rematch 覆写；
    # 现在 allocator 入队即 clone（_copy_for_free_group），传回原始 kv_indices 视图
    # 同样安全，且省掉一次拷贝、语义更直接：释放的始终是请求原有的 KV 槽位。
    self.token_to_kv_pool_allocator.free_segment(
        kv_indices[req.cache_protected_len : new_prefix_len],
        start_pos=req.cache_protected_len,
    )
```

python/sglang/srt/mem_cache/allocator/paged.py

free 与 free_segment 两条延迟释放路径都需拷贝入队，其中 free_segment 只拷贝页代表元（stride 切片），是该文件的关键改动点。

```
    def free_segment(self, free_index: torch.Tensor, *, start_pos: int):
        # 固定形状的 free()：页内 token 在 kv 行中连续，页代表元是等步长切片；
        # 不用 torch.unique()，因为其输出形状依赖数据，会触发设备同步。
        # 约定：一个组内每个页只能被一个调用释放。
        if free_index.numel() == 0:
            return

        ps = self.page_size
        offset = start_pos % ps
        if offset == 0:
            pieces = (free_index[::ps],)
        else:
            pieces = (free_index[:1], free_index[ps - offset :: ps])
```

```
if self.is_not_in_free_group:
    # 非批量路径：立即把页 id 归还 free_pages / release_pages
    self._release_page_ids(*(p // ps for p in pieces))
else:
    # 批量路径：只拷贝页代表元入队。原 tensor 只是 view，即使后续
    # req_to_token 行被 radix rematch 覆写，队列里持有的拷贝也不受影响，
    # 这正是本 PR 修复的核心场景。
    self.free_page_reps_group.extend(
        self._copy_for_free_group(piece) for piece in pieces
    )
```

评论区精华

该 PR 全程没有产生任何行内评论或讨论线程，reviewer hnyls2002 以空评语直接 APPROVED 并合并。根因分析与修复论证全部沉淀在 PR body：两张 ASCII 图清楚展示了“free_group 队列持有 req_to_token 行视图 → cache rematch 覆写行 → free_group_end() 释放了替换后的索引”的别名链路，并用 unified-cache 复现数据（逻辑淘汰 / 物理释放从 20 / 5 恢复到 20 / 20）佐证修复效果。这也意味着该修复缺少多人交叉审视，正确性论证主要依赖作者自证与新增单元测试。

- 评审结论与讨论情况 (other): 无未解决疑虑，PR 已合并。

风险与影响

- 风险：
  1. 设备端 clone 开销：NPU 等后端在延迟释放路径上会多做一次张量拷贝。free group 属于低频路径（请求结束 / 淘汰时触发），单次影响可忽略，但超大规模 batch 下需留意重复触发时的叠加开销。
  2. 约定依赖人工遵守：_copy_for_free_group 只是基类上的静态方法，没有任何注册表或断言强制未来新增的 allocator 在 group 分支调用它。任何新后端漏掉拷贝，radix_cache.py 回传原始 kv_indices 视图后，老 bug 会静默回归。
  3. 触发条件隐蔽：该 bug 需要“延迟释放期间同一 req_to_token 行被 radix rematch 覆写”这一时序才显现，单元测试能稳定复现，但缺少真实调度循环下的端到端回归测试，CI 覆盖主要靠本次新增的 3 个单元测试。
  4. 回归面较广：改动横扫 8 个 allocator 文件与 radix_cache.py，虽然每处都是同一行模式，但任何一处遗漏都会造成逻辑淘汰会计与物理释放状态不一致的隐性故障。
- 影响：
  - 运行时 / 用户侧：消除“缓存报告充足可淘汰容量但请求分配失败”的偶发故障；SWA 的 full_to_swa_index_mapping 不再残留已逻辑淘汰但物理未释放的映射，radix-cache 淘汰记账与物理分配器状态重新对齐。
  - 系统侧：token、paged、SWA、HiSparse、多端与 NPU 六类 allocator 的延迟释放语义统一为“入队即拥有”，批量化释放的性能收益完整保留；cache_unfinished_req 顺带减少一次 out-of-place 拷贝。

- 团队侧：形成一条新的 API 契约——free-group 分支必须调用 _copy_for_free_group；3 个单元测试把该约定固化为回归护栏，后续改动触碰该语义会立即失败。
- 风险标记：核心 KV 分配路径变更，所有权约定依赖人工遵守，NPU 等设备端新增 clone 开销，缺少端到端回归验证，跨 8 个 allocator 的同步修改

关联脉络

- PR #33404 [Scheduler] Gate SWA eviction on accumulated tokens: 同属 SWA / 滑窗分配与 KV-cache 淘汰记账的正确性主题。该 PR 修复的正是 SWA 逻辑淘汰与物理释放不一致的账实分叉问题，与 #33404 的 SWA 淘汰门控改动形成上下游呼应。
- PR #33888 config: delete the dead get_server_args() bindings across the repo: 同属 mem_cache / radix-cache 子系统的收口工作，触及 flexkv / lmcache radix cache 与调度批次相关路径，与本 PR 在分配器层做所有权收口的方向一致。