PR #34870 完整报告

sgl-project/sglang

Fix swa eviction frontier for bigram keys

合并时间：2026-08-16 11:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34870

执行摘要

- 一句话：修复 EAGLE bigram key 下 SWA 驱逐边界偏移导致的池泄漏
- 推荐动作：值得精读。这是一个典型的 ' 同一长度在两个坐标系下差一 ' 的根因修复：raw token 数与 bigram key 长度错一页，级联触发驱逐边界偏高、match 拒绝、保护长度停滞、KV 误释放。修复方式克制——提前构造 RadixKey、以 len(radix_key) - 1 作为唯一度量，并明确 'raw path 不动 '。建议重点阅读 cache_unfinished_req 中 key 构造时机调整，以及 TestSWAWindowUnderBigramKey 对页对齐形态的构造（seq_len = 4 * page_size），这是复现该 bug 的最小形态。

功能与动机

34653 将 SGLANG_OPT_UNIFIED_CACHE_FREE_OUT_OF_WINDOW_SLOTS 翻转为默认开启后，混合 SWA 模型 + EAGLE 推测解码在统一 radix cache 下触发池不变量检查崩溃（pool memory leak detected），所有 TP rank 死亡。#34823 已定位出 split_pos = swa_evicted_seqlen - result.prefix_len 的单元混用，并临时以 is_eagle gate 关闭 EAGLE 的 OOW 释放。本 PR 要解决的是同一崩溃的真正根因：SWA 驱逐 frontier 按 raw token 数计算，而 EAGLE 的 bigram key 比 token 少一项，leaf 实际停止位置低一页，使得插入后 leaf 保留的 live SWA 不足一个滑动窗口，后续 match 拒绝该 leaf，cache_protected_len 停滞，下一次插入把树已拥有的 KV 当重复释放。作者在 PR body 中明确：'a bigram key holds one entry less than the tokens it spans, so the leaf actually stops at page_floor(seq_len - 1), and the call site was still passing the token count'。

实现拆解

1. 定位根因：在 python/sglang/srt/mem_cache/unified_radix_cache.py 的 cache_unfinished_req 中，free_out_of_window_slots(req, effective_cache_len - 1, ...) 以原始 token 数作为驱逐边界，而 #29860 的边界约定是 page_floor(pre_len + 1)——即插入停止位置。EAGLE 下 RadixKey 是 bigram key，长度比 token 少 1，插入实际停止在 page_floor(len(radix_key))，比 token 坐标系低一页；当 seq_len 恰好页对齐（如 1024、page_size 64）时，frontier 从 896 算到 960，新 leaf 只剩 64 个 live SWA token，不足 127 的滑动窗口。
2. 修改释放边界调用：把 RadixKey 的构造从插入阶段提前到 SWA 逐组件释放之前，free_out_of_window_slots 改传 len(radix_key) - 1；随后插入阶段复用该 key 并 .page_aligned(self.page_size)，不改变后续 page-aligned 语义。raw key 路径（非 EAGLE）下 len(radix_key) == effective_cache_len，行为不变。
3. 撤销临时 gate：删除 #34823 引入的 and not self.tree_core.is_eagle 条件，EAGLE + 混合 SWA 模型重新默认启用 OOW 释放；DSpark、DFlash、非推测与混合 SSM 路径保持原有行为。
4. 新增回归测试：在 test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py 中新增 TestSWAWindowUnderBigramKey，以 page_size=4、sliding_window_size=7、is_eagle=True、seq_len=16 驱动 cache_unfinished_req，断言 boundary - swa_evicted_seqlen >= window 且 cache_protected_len == boundary；该测试在 main 上失败（4 个 live SWA token vs 7 窗口），修复后通过。
5. 验证配套：8xH200 TP8 gpt-oss-120b + EAGLE3 的 gsm8k 400 题准确率 OOW=1 为 0.853、无泄漏，OOW=0 为 0.845，无 spec 为 0.840；test_unified_radix_cache_unittest.py 与 test_swa_eviction_boundary.py 合计 1048 项全部通过；CI 上 8-gpu H200/B200 的 test_gpt_oss_120b.py 多次 rerun 均通过。

关键文件：

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 cache_unfinished_req）：核心修复文件：cache_unfinished_req 中 SWA 驱逐边界从 token 数改为 RadixKey 长度，并移除 EAGLE gate，是崩溃根因所在。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 缓存测试；类别 test；类型 test-coverage；符号 TestSWAWindowUnderBigramKey, _alloc_paged, test_match_after_insert_reaches_the_new_leaf）：新增 TestSWAWindowUnderBigramKey 回归测试，精确复现 bigram key 下 leaf 保留窗口不足导致 match 拒绝的场景，是修复有效性的关键证据。

关键符号：cache_unfinished_req, free_out_of_window_slots, test_match_after_insert_reaches_the_new_leaf, _alloc_paged

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

核心修复文件：cache_unfinished_req 中 SWA 驱逐边界从 token 数改为 RadixKey 长度，并移除 EAGLE gate，是崩溃根因所在。

```
# python/sglang/srt/mem_cache/unified_radix_cache.py
```

```
# cache_unfinished_req 中与 SWA 驱逐边界相关的核心片段（修复后）

        # components prepare insert data + return effective cache_len
        insert_params = InsertParams(
            prev_prefix_len=req.cache_protected_len,
            chunked=chunked,
            priority=getattr(req, "priority", 0) or 0,
        )
        effective_cache_len = len(token_ids)
        for comp in self._components_tuple:
            cl = comp.prepare_for_caching_req(
                req=req,
                insert_params=insert_params,
                token_ids_len=len(token_ids),
                is_finished=False,
            )
            if cl is not None:
                effective_cache_len = min(effective_cache_len, cl)

        # 先构造未对齐的 RadixKey：EAGLE 下 bigram key 比 token 少一项，
        # 因此 `len(radix_key)` 才是插入实际停止位置的长度单位。
        radix_key = RadixKey(
            token_ids[:effective_cache_len],
            req.extra_key,
            is_bigram=self.tree_core.is_eagle,
            cache_salt=req.cache_salt,
        )

        if envs.SGLANG_OPT_UNIFIED_CACHE_FREE_OUT_OF_WINDOW_SLOTS.get():
            # 驱逐边界必须落在插入停止位置的前一页之下（#29860 的约定）。
            # 原来传 `effective_cache_len - 1`，在 EAGLE bigram 下偏高一个 page，
            # 导致新 leaf 保留的 live SWA 不足一个滑动窗口，后续 match 拒绝该
            # leaf，`cache_protected_len` 不再前进，下一次插入把树已拥有的 KV
            # 当作重复释放。现在统一以 `len(radix_key) - 1` 度量。
            for comp in self._components_tuple:
                comp.free_out_of_window_slots(req, len(radix_key) - 1, insert_params)

        if effective_cache_len <= 0:
            req.prefix_indices = kv_indices_orig.to(dtype=torch.int64, copy=True)
            for comp in self._components_tuple:
                comp.cleanup_after_caching_req(
                    req, is_finished=False, insert_params=insert_params
                )
            return

        kv_indices = kv_indices_orig[:effective_cache_len]

        # 复用已构造的 key 做页对齐，插入与 match 逻辑保持不变
        radix_key = radix_key.page_aligned(self.page_size)
```

```
        page_aligned_len = len(radix_key)
        values = kv_indices[:page_aligned_len].to(dtype=torch.int64, copy=True)

        insert_params.key = radix_key
        insert_params.value = values
        result = self.insert(insert_params)
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增 TestSWAWindowUnderBigramKey 回归测试，精确复现 bigram key 下 leaf 保留窗口不足导致 match 拒绝的场景，是修复有效性的关键证据。

```
# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py
# 新增回归测试：EAGLE bigram key 下，插入后 leaf 必须保留完整滑动窗口

class TestSWAWindowUnderBigramKey(CustomTestCase):
    """EAGLE bigram key 比 token 少一项，leaf 停止在 page_floor(len(key))，
    驱逐边界必须与 key 对齐，否则 match 拒绝新 leaf，KV 被误释放。"""

    cfg = CacheConfig(
        page_size=4,
        components=(ComponentType.FULL, ComponentType.SWA),
        sliding_window_size=7,
        is_eagle=True,
        kv_size=256,
        max_context_len=64,
    )

    def _alloc_paged(self, allocator, need_size):
        # 同时从 full 与 swa 池分配页并对齐映射，模拟真实插入前的 KV 布局
        ps = self.cfg.page_size
        aligned = ((need_size + ps - 1) // ps) * ps
        full_indices = allocator.full_attn_allocator.alloc(aligned)
        swa_indices = allocator.swa_attn_allocator.alloc(aligned)
        self.assertIsNotNone(full_indices)
        self.assertIsNotNone(swa_indices)
        allocator.full_to_swa_index_mapping[full_indices] = swa_indices
        return full_indices[:need_size]

    def test_match_after_insert_reaches_the_new_leaf(self):
        cache, allocator, req_to_token_pool = build_fixture(self.cfg)
        page_size = self.cfg.page_size
        # 页对齐长度是最容易让 bigram key 少一页的形态
        seq_len = 4 * page_size

        # 构造完整请求状态：填入 token、KV 索引、初始 last_node 与保护长度
        req = Req(
            rid=0,
            origin_input_text="",
            origin_input_ids=array("q"),
```

```
        sampling_params=SamplingParams(temperature=0, max_new_tokens=1),
    )
    req_to_token_pool.alloc([req])
    tokens = list(range(1, seq_len + 1))
    req.origin_input_ids = tokens
    req.output_ids = []
    req.full_untruncated_fill_ids = array("q", tokens)
    req.set_extend_range(0, len(req.full_untruncated_fill_ids))
    kv_indices = self._alloc_paged(allocator, seq_len)
    req_to_token_pool.write((req.req_pool_idx, slice(0, seq_len)), kv_indices)
    req.kv_committed_len = seq_len
    req.last_node = cache.root_node.id
    req.cache_protected_len = 0
    req.swa_uuid_for_lock = None
    req.extra_key = None
    req.kv = ReqKvInfo(kv_allocated_len=0, swa_evicted_seqlen=0)

    with envs.SGLANG_OPT_UNIFIED_CACHE_FREE_OUT_OF_WINDOW_SLOTS.override(True):
        cache.cache_unfinished_req(req)

    # leaf 实际停止在 page_floor(seq_len - 1)，它到驱逐前沿之间必须
    # 仍保留至少一个滑动窗口的 live SWA，否则后续 match 会拒绝该 leaf
    boundary = (seq_len - 1) // page_size * page_size
    self.assertGreaterEqual(
        boundary - req.kv.swa_evicted_seqlen,
        self.cfg.sliding_window_size,
        f"leaf ending at {boundary} keeps only "
        f"{boundary - req.kv.swa_evicted_seqlen} live SWA tokens against a "
        f"{self.cfg.sliding_window_size} window",
    )
    # 匹配必须到达插入刚创建的 leaf，cache_protected_len 随之推进
    self.assertEqual(
        req.cache_protected_len,
        boundary,
        "the match after the insert must reach the leaf the insert created",
    )

    cache.dec_lock_ref(
        req.last_node,
        DecLockRefParams(swa_uuid_for_lock=getattr(req, "swa_uuid_for_lock", None)),
    )
    cache.sanity_check()
```

评论区精华

该 PR 没有 reviewer 评论（review_comments = 0），8 条 issue 评论均为 CI rerun 指令与结果回调。设计权衡主要体现在 7 个 commit 的迭代中：最初 commit 'accept swa tombstone as match boundary' 尝试放宽 match 规则、接受 tombstone 作为边界；随后 '

take post-insert indices from insert path, revert validator change' 转向修正插入路径的索引来源；最终 commit 'measure the frontier against the radix key, not the tokens' 确定根因在驱逐边界的度量单位，而非 match 语义。作者在 PR body 中给出关键对比数据：修复前后 swa_ev=896→832、tail=64→128、ok=False→True，并强调 'nothing about the raw path moves'。

- 修复方向：放松 match 边界 vs 修正驱逐边界度量 (design): 选择保持 match 语义不变，将释放边界统一到 RadixKey 长度；raw key 路径行为不变。
- 是否移除 #34823 的 EAGLE gate (correctness): 移除 gate，EAGLE 系列重新默认启用 OOW 释放；以新测试与 8xH200 精度 / 泄漏验证背书。

风险与影响

- 风险：
  1. 核心缓存路径变更：cache_unfinished_req 是所有走统一 radix cache 请求的必经入口，本次前置了 RadixKey 构造并改动释放边界入参；非 EAGLE 路径参数等价（len(radix_key) == effective_cache_len），但仍有回归扩散面。
  2. EAGLE 路径重新默认启用 OOW 释放：移除 gate 后，EAGLE + SWA 部署恢复 #34653 的默认行为。PR 只对 hybrid SWA + EAGLE（gpt-oss-120b TP8）做了 e2e 验证，DSpark、DFlash 以及 SWA-only、mamba-only 配置没有单独验证（#34653 也自述只测过 hybrid SWA + mamba）。
  3. 页对齐假设：修复依赖 bigram key 与 page_floor 的一一对应；page_size=1 时页对齐路径被跳过，边界行为不同，已有测试未覆盖。
  4. 提前构造 key 的副作用面：effective_cache_len <= 0 的早退分支现在会先构造一个空 radix_key，当前无实际影响，但该代码位置未来若新增副作用需注意。
  5. 正确性收益：修复后 swa_evicted_seqlen 与树内记账一致，根治了 #34823 观察到的池泄漏崩溃，且不牺牲前缀复用（#34653 的 cached_tokens 断言仍通过）。
- 影响：
  - 用户侧：所有默认配置下使用 EAGLE 推测解码 + 混合 SWA 模型的部署原会每个 TP rank 崩溃退出，本修复恢复可用，并使 OOW 释放优化重新生效，释放已滑出窗口的 SWA slot、降低缓存占用。
  - 系统侧：统一 radix cache 的 SWA 驱逐边界回到 #29860 的设计约定，插入、match、cache_protected_len 推进与 KV 释放恢复一致；#34823 的临时 gate 被撤销，代码路径简化。
  - 团队侧：新增的 TestSWAWindowUnderBigramKey 明确了 ' 度量单位必须与 key 对齐 ' 的约束，可防止后续重构再引入同类回归；8GPU 与单测双覆盖保障了多配置下的信心。
  - 风险标记：核心路径变更，默认路径重新启用，配置面覆盖有限，依赖页对齐假设

关联脉络

- PR #34823 Skip oow slot freeing under eagle: 本 PR 是其根因修复：移除其引入的 is_eagle gate，解决同一 pool memory leak 崩溃。

- PR #34653 Enable unified cache out-of-window slot freeing by default: 默认开启后暴露该 bug；本 PR 使该默认优化在 EAGLE + SWA 路径恢复可用。