PR #34729 完整报告

sgl-project/sglang

Retain SWA down to the last state checkpoint

合并时间：2026-08-15 01:12

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34729

执行摘要

- 一句话：保留 SWA 至最后 checkpoint，混合模型 decode 前缀复用翻倍
- 推荐动作：值得精读。核心设计决策有三个：一是“驱逐基准（tail）与匹配落点（state checkpoint）不一致”的根因分析，二是让 helper 保持对 mamba 无知、由知道两方组件的缓存层统一计算 floor 的模块化边界，三是用 KL bitexact 测试作为正确性仪表来验证“多保留的状态是否真的正确”。注意作者标注的两个 TODO（prefill 覆盖、SWA 池 sizing），合并该 PR 后应持续跟踪。

功能与动机

PR body 开篇即指出问题：'A hybrid SWA + mamba model throws away most of its decode-region prefix reuse at the default --mamba-track-interval'。作者用 32 个 prompt 的实测数据定位：page_size=128、track_interval=256 时仅 16/32 请求在第二轮命中 decode 区域，track_interval=512 时仅 5/32。根因是 SWA 驱逐相对 tail 计算（evict_threshold = pre_len - max(window, page)），而前缀匹配落在位于 tail 之前的 state checkpoint 上，匹配需要 checkpoint 之下有一整窗口存活 SWA，但部分窗口已被释放，且 swa_evicted_seqlen 只前进不回退。此前唯一的绕法是让 track_interval 等于 page_size，代价是 checkpoint 密度与 mamba 池 footprint 翻倍；本 PR 将两者解耦。

实现拆解

变更入口是驱逐工具函数与缓存层接口的成对改造，共 4 个步骤：

1. 驱逐工具增加可选下限：python/sglang/srt/mem_cache/common.py 的 free_swa_out_of_window_slots 新增 retain_floor: int | None = None 关键字参数；在 radix cache 分支算出 evict_threshold = pre_len - max(window, page) 后追加 min(evict_threshold, retain_floor)。None 时行为与旧版完全一致；is_chunk_cache=True 时显式跳过 floor（chunk cache 不建树，checkpoint 不可能被匹配，保留纯属成本）。frontier 单调性由外层 max(req.kv.swa_evicted_seqlen, ...) 保证，floor 永远不会导致“反释放”。
2. floor 计算集中到缓存层：BasePrefixCache 新增默认方法 swa_retain_floor(req) -> int | None，直接返回 None——没有第二状态流的缓存不受影响；UnifiedRadixCache 覆写该方法，仅当 is_mamba_enabled 且 req.mamba_last_track_seqlen 存在时返回 checkpoint - sliding_window_size，保证 checkpoint 之下保留完整窗口供前缀匹配读取。

3. 两个调用点统一走缓存接口：ScheduleBatch._evict_swa（decode 侧驱逐）和 SWAComponent.free_out_of_window_slots（chunked-prefill 插入路径）分别传入 retain_floor=self.tree_cache.swa_retain_floor(req) 与 retain_floor=self.cache.swa_retain_floor(req)，floor 推导只在一处完成，helper 对 mamba 保持无知。
4. 测试与精度验证：test/registered/unit/mem_cache/test_swa_eviction_boundary.py 新增 5 个用例（钳制、chunk cache 忽略、None 兼容旧行为、floor 高于阈值时惰性、frontier 只前进不反释放）；复用 test_unified_radix_cache_kl_hybrid_bitexact（KL 下限 1e-9）在 H200/B200 验证：floor 开启后新增命中的请求 KL 全部为 0，证明保留的窗口内容正确而非仅存在。

关键文件：

- python/sglang/srt/mem_cache/common.py（模块 驱逐逻辑；类别 source；类型 core-logic；符号 free_swa_out_of_window_slots）：驱逐逻辑行为落点。新增 retain_floor 可选参数并用 min() 钳制 evict_threshold，None 时与旧行为逐字节一致，chunk cache 分支显式忽略 floor，是整个变更的核心机制。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 统一缓存；类别 source；类型 core-logic；符号 swa_retain_floor）：新增 UnifiedRadixCache.swa_retain_floor 覆写，把 floor 计算集中到同时了解 SWA 与 mamba 两种组件的缓存层：仅 mamba 启用且 checkpoint 存在时返回 checkpoint - sliding_window_size。
- python/sglang/srt/mem_cache/base_prefix_cache.py（模块 缓存基类；类别 source；类型 core-logic；符号 swa_retain_floor）：定义默认 swa_retain_floor 返回 None 的接口缝，保证没有第二状态流的缓存不受影响；新增方法带注释说明覆写契约。
- python/sglang/srt/managers/schedule_batch.py（模块 调度器；类别 source；类型 core-logic；符号 _evict_swa）：decode 侧驱逐调用点，一行改动接入 floor：ScheduleBatch._evict_swa 将 retain_floor 传给 free_swa_out_of_window_slots。
- python/sglang/srt/mem_cache/unified_cache/components/swa_component.py（模块 SWA 组件；类别 source；类型 core-logic；符号 free_out_of_window_slots）：chunked-prefill 插入路径的第二个调用点，同样通过 self.cache.swa_retain_floor 获取 floor，保证预填充插入时也保留 checkpoint 窗口。
- test/registered/unit/mem_cache/test_swa_eviction_boundary.py（模块 缓存驱逐；类别 test；类型 test-coverage；符号 test_retain_floor_clamps_eviction, test_retain_floor_ignored_for_chunk_cache, test_retain_floor_none_matches_old_behaviour, test_retain_floor_above_threshold_is_inert）：5 个新用例完整刻画 retain_floor 行为契约：钳制、chunk cache 忽略、None 兼容旧行为、惰性、不反释放；配合 KL bitexact 精度测试作为正确性仪表。

关键符号：free_swa_out_of_window_slots, BasePrefixCache.swa_retain_floor, UnifiedRadixCache.swa_retain_floor, ScheduleBatch._evict_swa, SWAComponent.free_out_of_window_slots

关键源码片段

python/sglang/srt/mem_cache/common.py

驱逐逻辑行为落点。新增 retain_floor 可选参数并用 min() 钳制 evict_threshold，None 时与旧行为逐字节一致，chunk cache 分支显式忽略 floor，是整个变更的核心机制。

```
def free_swa_out_of_window_slots(
    req: Req,
    pre_len: int,
    *,
    sliding_window_size: int,
    page_size: int,
    req_to_token_pool: ReqToTokenPool,
    token_to_kv_pool_allocator: BaseTokenToKVPoolAllocator,
    is_chunk_cache: bool = False,
    retain_floor: int | None = None,
) -> None:
    if req.kv is None:
        return

    # SWA radix cache 需要驱逐不在树缓存、也不在滑动窗口内的 token。
    # cache_protected_len 必须按 page 对齐，避免驱逐侵入受保护区域。
    assert (
        req.cache_protected_len % page_size == 0
    ), "cache_protected_len must be page aligned"
    evict_floor = max(req.cache_protected_len, getattr(req, "swa_evict_floor", 0))
    if page_size > 1 and evict_floor > req.cache_protected_len:
        evict_floor = -(-evict_floor // page_size) * page_size
    req.kv.swa_evicted_seqlen = max(req.kv.swa_evicted_seqlen, evict_floor)

    if is_chunk_cache:
        # Chunk cache 不构建 radix tree，无 tombstone-leaf 顾虑；
        # 直接驱逐到窗口边界，尾部 queue 保证按 page 对齐。
        evict_threshold = pre_len - sliding_window_size
    else:
        # Radix cache：至少保留 max(window, page)。尾部按 page 对齐，
        # 且多减一页保证 frontier 低于插入边界 page_floor(seq_len)，
        # 避免最后一个叶子变成全 tombstone。
        evict_threshold = pre_len - max(sliding_window_size, page_size)

    if retain_floor is not None and not is_chunk_cache:
        # 调用方决定 floor 的位置（见 BasePrefixCache.swa_retain_floor）；
        # 这里只承诺不释放越过 floor 的 slot。
        # Chunk cache 没有树，保留的 checkpoint 永远不会被匹配，纯属成本。
        evict_threshold = min(evict_threshold, retain_floor)

    new_swa_evicted_seqlen = max(
        req.kv.swa_evicted_seqlen,
        evict_threshold,
    )
```

```
    if page_size > 1:
        new_swa_evicted_seqlen = (new_swa_evicted_seqlen // page_size) * page_size

    if new_swa_evicted_seqlen > req.kv.swa_evicted_seqlen:
        free_slots = req_to_token_pool.req_to_token[
            req.req_pool_idx, req.kv.swa_evicted_seqlen : new_swa_evicted_seqlen
        ]
        token_to_kv_pool_allocator.free_swa(free_slots)
        maybe_evict_dsv4_state_on_swa(
            token_to_kv_pool_allocator, req_to_token_pool, req, new_swa_evicted_seqlen
        )
        req.kv.swa_evicted_seqlen = new_swa_evicted_seqlen
```

python/sglang/srt/mem_cache/unified_radix_cache.py

新增 UnifiedRadixCache.swa_retain_floor 覆写，把 floor 计算集中到同时了解 SWA 与 mamba 两种组件的缓存层：仅 mamba 启用且 checkpoint 存在时返回 checkpoint - sliding_window_size。

```
def swa_retain_floor(self, req) -> int | None:
    # 只有 SWA 与 mamba 混合缓存需要保留到最后一个 state checkpoint：
    # 前缀匹配落在 checkpoint 上，而 checkpoint 位于 tail 之后，
    # 若按 tail - window 驱逐会毁掉本可复用的 decode 区域。
    if not self.is_mamba_enabled or self._sliding_window_size is None:
        return None
    checkpoint = req.mamba_last_track_seqlen
    if checkpoint is None:
        # 首个 decode 步之前 checkpoint 尚未写入树，暂无 floor。
        return None
    # 保留 checkpoint 之前一整个窗口的 SWA slot，供前缀匹配读取。
    return checkpoint - self._sliding_window_size
```

test/registered/unit/mem_cache/test_swa_eviction_boundary.py

5 个新用例完整刻画 retain_floor 行为契约：钳制、chunk cache 忽略、None 兼容旧行为、惰性、不反释放；配合 KL bitexact 精度测试作为正确性仪表。

```
def test_retain_floor_clamps_eviction(self):
    """混合缓存把 SWA 保留到最后一个 state checkpoint，而不是 tail 之后的
    窗口，因为前缀匹配落在 checkpoint 上。即使 tail 已远远前移，
    floor 也必须钳制驱逐前沿。"""
    page_size, window = 8, 16
    tree, allocator, pool = _build_swa_tree(
        page_size=page_size, sliding_window_size=window
    )
    seq_len = 200
    checkpoint = 96
    kv = _swa_alloc(allocator, seq_len)
    pool.write((0, slice(0, seq_len)), kv)
    req = _make_req(0, list(range(seq_len)), 0, tree)
    batch = _make_batch(tree, allocator, pool)
```

```
    free_swa_out_of_window_slots(
        req,
        seq_len - 1,
        sliding_window_size=window,
        page_size=page_size,
        req_to_token_pool=batch.req_to_token_pool,
        token_to_kv_pool_allocator=batch.token_to_kv_pool_allocator,
        retain_floor=checkpoint - window,
    )

    # 无 floor 时会推进到 page_floor(199 - 16) = 176,
    # 有 floor 时驱逐前沿必须被钳制在 checkpoint - window 之内。
    self.assertLessEqual(req.kv.swa_evicted_seqlen, checkpoint - window)
    self.assertEqual(req.kv.swa_evicted_seqlen % page_size, 0)


def test_retain_floor_does_not_unfree(self):
    """frontier 只前进。floor 在 slot 已释放后到达时，不能把 slot 认领回来，
    否则下一轮会双重释放。"""
    page_size, window = 8, 16
    tree, allocator, pool = _build_swa_tree(
        page_size=page_size, sliding_window_size=window
    )
    seq_len = 200
    kv = _swa_alloc(allocator, seq_len)
    pool.write((0, slice(0, seq_len)), kv)
    req = _make_req(0, list(range(seq_len)), 0, tree)
    batch = _make_batch(tree, allocator, pool)
    common_kwargs = dict(
        sliding_window_size=window,
        page_size=page_size,
        req_to_token_pool=batch.req_to_token_pool,
        token_to_kv_pool_allocator=batch.token_to_kv_pool_allocator,
    )

    # 第一次无 floor 驱逐，frontier 前移。
    free_swa_out_of_window_slots(req, seq_len - 1, **common_kwargs)
    advanced = req.kv.swa_evicted_seqlen
    self.assertGreater(advanced, 0)

    # 第二次带 floor=0 调用，必须保持原 frontier 不变。
    free_swa_out_of_window_slots(req, seq_len - 1, retain_floor=0, **common_kwargs)
    self.assertEqual(req.kv.swa_evicted_seqlen, advanced)
```

评论区精华

本 PR 没有 review 评论（review_comments_count = 0），核心讨论来自 PR body 的作者自查：

1) prefill 区域尚未覆盖——req.mamba_last_track_seqlen 在 cache_unfinished_req 末尾被清空，首个 decode 步 floor 读到 None，需要把 floor 计算迁移到树侧； 2) SWA 池占用成本未直接测量，作者给出解析上界（checkpoint 间距）与探针数据（每请求额外保留 127-255 token，窗口 511，占比 <1%），并明确表态 'Worth a reviewer's judgement rather than my assertion'； 3) 后续计划把 SWA 池 sizing floor 从 window 提升为 window + interval，防止池压力下额外保留反噬。Issue 评论中主要是 CI 重跑：第一次在 2-gpu-h100 上 test_unified_radix_cache_kl_swa.py 与 test_unified_radix_cache_kl_full.py 失败，重跑后全部通过。

- prefill 区域未被保留覆盖（作者 TODO） (design): 本 PR 明确不覆盖 prefill 区域，留待后续将 floor 计算迁移到树侧后处理。
- SWA 池占用成本未测量 (performance): 作者给出解析界限（checkpoint 间距）与实测探针数据，并计划后续把池 sizing floor 从 window 改为 window + interval，防止池压力下反噬。

风险与影响

- 风险： 1) SWA 池占用上升（未量化）：floor 让每个请求多保留至多一个 checkpoint 间距的 slot，作者估算占比低于 1% 但未直接测量；池按 mem-fraction-static 预分配，常规配置不改变设备峰值，但池压力场景可能影响批处理并发，作者已计划把池 sizing floor 改为 window + interval。 2) prefill 区域未覆盖：req.mamba_last_track_seqlen 清空时机导致首个 decode 步无 floor，prefill 阶段仍可能丢失复用，属作者明示 TODO。 3) 正确性防护：frontier 只前进不后退，floor 不会导致已释放 slot 被重新认领（双重释放风险已被 test_retain_floor_does_not_unfree 覆盖）；chunk cache 显式忽略；非 mamba 缓存走 None 默认路径行为不变。 4) CI 偶发：KL 相关测试在 2-gpu-h100 首轮失败、重跑通过，未完全排除环境因素。
  - 影响：对用户：混合 SWA + mamba 模型（如带 mamba 状态的 Qwen3.x 系列）第二轮请求 decode 区域缓存命中率从约 50%（16/32）甚至 16%（5/32）提升到 100%（32/32），复用前缀 token +12.9%、重算 token -77.1%，在多轮对话与多请求共享前缀场景下收益显著。对系统：改动集中在 mem_cache 驱逐路径，非 mamba 缓存与 chunk cache 显式不受影响；SWA 池占用小幅上升需在池压力配置下观察。对团队：确立了“匹配落在 checkpoint、驱逐相对 tail”这一缓存语义认知，为后续把 floor 迁移到树侧（覆盖 prefill）以及 SWA 池 sizing 调整铺路。
  - 风险标记：SWA 池占用上升未量化，prefill 区域复用未覆盖，核心缓存驱逐路径变更

关联脉络

- PR #34808 Fix mamba checkpoint depth under dcp: 同属 mamba checkpoint 状态语义修复，同样触及 schedule_batch.py 与 unified_cache 组件；本 PR 读取的 req.mamba_last_track_seqlen 与 checkpoint 深度管理直接相关，二者构成 mamba 状态缓存的连续演进。
- PR #34823 Skip oow slot freeing under eagle: 同属 unified_radix_cache.py 中 SWA 槽位释放 / 驱逐行为修正：一个避免 OOW 释放导致缓存池崩溃，一个调整释放边界提升前缀复用，方向互补。