

PR #39426 完整报告

sgl-project/sglang

bugfix:fix unifiedcache c128 radix cache management

合并时间: 2026-09-16 16:47

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39426>

执行摘要

- 一句话: 修复 unified cache C128 对齐与释放区间重叠
- 推荐动作: 值得精读, 但重点不在代码量而在两处契约: 一是 base.py 中 floor_cache_len 的 "min 之后再做对齐" 语义, 它决定了组件体系的长度协商分两个阶段, 读者应理解为什么不能放在 prepare_for_caching_req 里; 二是 cache_finished_req 中把两个相接区间合并为 (free_from, len(kv_indices_full)) 的判据, 这是典型的 "边界共享一页" 陷阱, 建议对照 free_kv_row 的升序 / 不重叠约束一起读。

建议后续动作: 为该修复补注册单测 (覆盖 EAGLE 与非 EAGLE 下 floor_cache_len 的边界值、以及 tail_free_start == len(kv_indices) 时命中合并分支 / 不命中分支), 并为 floor_cache_len 增加 "结果不得低于 cache_protected_len" 的显式约束或断言, 避免多组件粒度不一致时出现隐性回退。

功能与动机

PR body 明确指出两个动机: 1) "Current runs have revealed the need to check unaligned truncation boundaries, so as to prevent the end of the previous interval from overlapping the start of the following interval", 即实际跑任务时发现未对齐截断边界可能让前一个释放区间与后一个区间重叠; 2) "For the DeepSeek V4 NPU platform, independent C128 cache management is used. When performing alignment truncation, the C128 alignment length (12814=2048) must be taken into account. The floor_cache_len in the component is invoked in the unified cache.", 即 NPU 上 DeepSeek V4 使用独立的 C128 缓存管理, 截断时必须按 C128 对齐粒度 (12814=2048) 向下取整。该改动从更大的 PR#38251 中拆分而来, 作者标注 Speed Tests "has no impact"。

实现拆解

- 1) 在组件基类引入对齐契约: python/sglang/srt/mem_cache/unified_cache/components/base.py 新增 floor_cache_len(cache_len) 钩子, docstring 说明其语义是 "在所有组件的 prepare_for_caching_req 截断结果被取 min 之后, 再约束合并后的有效缓存长度", 默认实现直接返回入参。这样 Full / SWA / Mamba 等现有组件行为完全不变, 只有声明更强对齐要求的组件才覆写, 属于向后兼容的接口扩展。
- 2) 在 C128 组件实现对齐: python/sglang/srt/hardware_backend/npu/dsv4/c128_sidecar_component.py 覆写 floor_cache_len, 复用 prepare_for_caching_req 的同一套规则

——EAGLE (bigram) 场景先减 1、按 `group_tokens = 128 *`

`self.allocator.c128_attn_allocator.page_size` 向下取整、再补回 1，保证 sidcar 页数是整数且与插入时使用的 key 长度口径一致。

3) 在统一 radix cache 集成两个阶段的长度协商：`python/sglang/srt/mem_cache/unified_radix_cache.py` 的 `cache_finished_req` 与 `cache_unfinished_req` 中，在原有 `prepare_for_caching_req` 取 min 的循环之后，新增 `effective_cache_len = comp.floor_cache_len(effective_cache_len)` 循环。放在 min 之后而非之前是关键：先 floor 再 min 会被 min 重新打散对齐。在 `cache_unfinished_req` 中该调用位于构造 `RadixKey` 之前、`effective_cache_len <= 0` 早退判断之前，因此被取整到 0 时会直接回退为完整 `prefix_indices`，不产生插入。

4) 修复释放区间重叠：`cache_finished_req` 中 `ranges = [(free_from, len(kv_indices))]` 与延后的截断尾巴 (`tail_free_start, len(kv_indices_full)`) 原本无条件追加。由于 `tail_free_start = max(effective_cache_len, req.kv.cache_protected_len)` 且 `len(kv_indices) == effective_cache_len`，只要 `tail_free_start <= len(kv_indices)` 成立就意味着两段在截断边界处首尾相接、共用同一页；新逻辑改为 `ranges[0] = (free_from, len(kv_indices_full))` 合并成一个区间，否则才追加，避免把相邻（潜在重叠）范围交给 `free_kv_row`。

5) 测试与配置配套：本次没有任何测试文件变更，也没有新增配置项或环境变量；`floor_cache_len` 的 EAGLE / 非 EAGLE 两条分支以及区间合并分支都缺少单测覆盖。CI 方面 PR body 显示 PR Test (Base) 通过，Extra 与 AMD ROCm 10 为失败，作者在评论中反复触发 `/rerun-failed-ci`，提交历史中只有一次 `Merge branch 'main'` 的同步合并。

关键文件：

- `python/sglang/srt/mem_cache/unified_radix_cache.py`（模块 统一缓存树；类别 source；类型 core-logic；符号 `cache_finished_req`, `cache_unfinished_req`）：统一 radix cache 的核心集成点：在 `cache_finished_req` / `cache_unfinished_req` 中加入 `floor_cache_len` 第二阶段协商，并修复截断尾巴与未对齐尾巴释放区间接续时的合并逻辑，是本次唯一影响所有 unified cache 用户的共享代码。
- `python/sglang/srt/hardware_backend/npu/dsv4/c128_sidcar_component.py`（模块 旁路缓存组件；类别 source；类型 core-logic；符号 `floor_cache_len`, `prepare_for_caching_req`）：C128 sidcar 组件是本次对齐逻辑的唯一实现方，`floor_cache_len` 复用了 `prepare_for_caching_req` 的 EAGLE 减一 / 补一与 `128*page_size` 取整规则，决定了 NPU DeepSeek V4 上实际写入 radix 树的长度。
- `python/sglang/srt/mem_cache/unified_cache/components/base.py`（模块 组件基类；类别 source；类型 core-logic；符号 `floor_cache_len`）：组件基类新增 `floor_cache_len` 钩子，定义了两阶段长度协商契约，默认恒等实现保证 Full / SWA / Mamba 等既有组件行为零变化，是本次接口设计的落点。

关键符号：`floor_cache_len`, `cache_finished_req`, `cache_unfinished_req`, `prepare_for_caching_req`, `free_kv_row`

关键源码片段

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

统一 radix cache 的核心集成点：在 cache_finished_req / cache_unfinished_req 中加入 floor_cache_len 第二阶段协商，并修复截断尾巴与未对齐尾巴释放区间接续时的合并逻辑，是本次唯一影响所有 unified cache 用户的共享代码。

```
# python/sglang/srt/mem_cache/unified_radix_cache.py —— cache_finished_req()
中的长度协商与区间释放
```

```
# 第一阶段：各组件 prepare 并返回自己可接受的 cache 长度，调用方取 min
```

```
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=True,
    )
```

```
    if cl is not None:
```

```
        effective_cache_len = min(effective_cache_len, cl)
```

```
# 第二阶段（本次新增）：min 之后再让每个组件做一次 " 对齐向下取整 "。
```

```
# 顺序不能颠倒：若先 floor 再 min，各组件声明的对齐义务会被 min 重新打散。
```

```
# C128 sidecar 会在这里把长度压到 128 * page_size 的整数倍。
```

```
for comp in self._components_tuple:
```

```
    effective_cache_len = comp.floor_cache_len(effective_cache_len)
```

```
# 按协商结果截断；尾部释放先记账（tail_free_start），稍后与未对齐尾巴一起释放
```

```
kv_indices_full = kv_indices
```

```
tail_free_start = None
```

```
if effective_cache_len < len(token_ids):
```

```
    tail_free_start = max(effective_cache_len, req.kv.cache_protected_len)
```

```
    token_ids = token_ids[:effective_cache_len]
```

```
    kv_indices = kv_indices[:effective_cache_len]
```

```
# ... 此处省略构造 RadixKey、insert、prompt 拓扑再插入等逻辑 ...
```

```
# 释放未对齐尾巴（+ 延后的截断尾巴）。
```

```
# 因 tail_free_start >= effective_cache_len == len(kv_indices),
```

```
# tail_free_start <= len(kv_indices) 成立即意味着两段在截断边界处首尾相接、
```

```
# 共用同一页；此时必须合并成一个区间，否则 free_kv_row 会收到相邻 / 重叠范围。
```

```
free_from = (
```

```
    # min(): 受保护前缀可能已经越过被截断的 cache_len，且 free_kv_row 只接受升序范围
```

```
    min(req.kv.cache_protected_len, len(kv_indices))
```

```
    if result.rotation_tail_declined
```

```
    else page_aligned_len
```

```
)
```

```
ranges = [(free_from, len(kv_indices))]
```

```
if tail_free_start is not None:
```

```
    if free_from < len(kv_indices) and tail_free_start <= len(kv_indices):
```

```
        # 两段首尾相接且共享边界页：释放到 kv_indices_full 末尾的单个区间
```

```

        ranges[0] = (free_from, len(kv_indices_full))
    else:
        ranges.append((tail_free_start, len(kv_indices_full)))
self.free_kv_row(req.kv, ranges)

# python/sglang/srt/mem_cache/unified_radix_cache.py —— cache_unfinished_req()
中的同款调用
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)

# 新增的对齐取整发生在构造 RadixKey 与 effective_cache_len <= 0 早退之前,
# 因此被取整到 0 时不会插入任何节点, 而是回退为完整的 prefix_indices。
for comp in self._components_tuple:
    effective_cache_len = comp.floor_cache_len(effective_cache_len)

radix_key = RadixKey(
    token_ids[:effective_cache_len],
    req.extra_key,
    is_bigram=self.tree_core.is_eagle,
    cache_salt=req.cache_salt,
)

```

python/sglang/srt/hardware_backend/npu/dsv4/c128_sidecar_component.py

C128 sidecar 组件是本次对齐逻辑的唯一实现方, floor_cache_len 复用了 prepare_for_caching_req 的 EAGLE 减一 / 补一与 128*page_size 取整规则, 决定了 NPU DeepSeek V4 上实际写入 radix 树的长度。

```

# python/sglang/srt/hardware_backend/npu/dsv4/c128_sidecar_component.py

def prepare_for_caching_req(
    self,
    req: Req,
    insert_params: InsertParams,
    token_ids_len: int,
    is_finished: bool,
) -> int:
    # 把请求侧 token 对齐到 C128 组粒度, 并把对应的 sidecar 页写入 insert_params
    logical_len = token_ids_len
    if self.tree_core.is_eagle and logical_len > 0:
        # bigram: EAGLE 下 key 长度比 token 少 1, 先减一以对齐口径

```

```

    logical_len -= 1
    group_tokens = 128 * self.allocator.c128_attn_allocator.page_size
    cache_len = logical_len // group_tokens * group_tokens
    num_pages = cache_len // group_tokens
    insert_params.c128_value = self.cache.req_to_token_pool.req_to_c128_sidecar[
        int(req.kv.req_pool_idx), :num_pages
    ].clone()
    # 再补回 1，保证返回的长度与 radix key 语义一致
    return cache_len + 1 if self.tree_core.is_eagle and cache_len > 0 else cache_len

```

```

def floor_cache_len(self, cache_len: int) -> int:
    # 本次新增：min() 之后的组合长度仍可能不满足 C128 粒度，
    # 这里做统一向下取整，保证 sidecar 页数是整数、不出现半页缓存。
    # 取整规则必须与 prepare_for_caching_req 完全一致（含 EAGLE 的减一 / 补一），
    # 否则插入时记录的长度与实际页数会错位。
    logical_len = cache_len
    if self.tree_core.is_eagle and logical_len > 0:
        logical_len -= 1
    group_tokens = 128 * self.allocator.c128_attn_allocator.page_size
    floored = logical_len // group_tokens * group_tokens
    return floored + 1 if self.tree_core.is_eagle and floored > 0 else floored

```

python/sglang/srt/mem_cache/unified_cache/components/base.py

组件基类新增 floor_cache_len 钩子，定义了两阶段长度协商契约，默认恒等实现保证 Full / SWA / Mamba 等既有组件行为零变化，是本次接口设计的落点。

```
# python/sglang/srt/mem_cache/unified_cache/components/base.py
```

```

class UnifiedCacheComponent:
    # ... 其余 match / insert / evict 钩子略 ...

    def finalize_match_result_in_cache(self, params, result):
        """Cache-level finalize after the match walk."""
        return result

    def floor_cache_len(self, cache_len: int) -> int:
        """Constrain the combined effective cache length after every
        component's `prepare_for_caching_req` truncation has been min'd."""
        # 基类默认恒等：未覆写该钩子的组件（Full / SWA / Mamba 等）行为完全不变，
        # 只有声明了更强对齐要求的组件（例如 NPU DeepSeek V4 的 C128 sidecar）才覆写。
        # 调用方保证在 min 之后按组件顺序串行调用本方法，因此后调用的组件会看到
        # 前一个组件取整后的结果，多组件组合粒度需要由实现方自行保证兼容。
        return cache_len

```

评论区精华

本 PR 没有任何 inline review 评论（review_comments_count = 0），两位 reviewer（sglang-npu-bot、iforgetmyname）均为无正文的 APPROVED，设计权衡只能从 PR body 与

代码注释反推：

```
sglang-npu-bot / iforgetmyname: APPROVED (空正文)
```

```
vstone-w: /rerun-failed-ci try again again again; sglang-npu-bot: /tag-run-ci-label
```

可提炼的 " 隐性讨论 " 有三点：其一，为什么要把对齐单独拆成 `floor_cache_len` 而不是塞进 `prepare_for_caching_req`——因为前者是 " 组合后约束 "、后者是 " 单组件返回 "，两者语义不同，基类注释明确写了要在 `min` 之后调用；其二，C128 的取整必须沿用 `prepare_for_caching_req` 同一套 EAGLE 减一 / 补一逻辑，否则插入 `key` 与实际页数会错位；其三，CI 在 Extra 与 AMD 流水线上为红，作者仅以重跑回应，未见失败原因分析。

- 为什么把对齐取整拆成独立钩子 `floor_cache_len`，而不是并入 `prepare_for_caching_req` (design): 以新增基类钩子（默认恒等）的方式落地，向后兼容既有组件；两位 reviewer (sglang-npu-bot、iforgetmyname) 无正文 APPROVED，未提出异议。
- 释放区间在截断边界相接、共用同一页时必须合并 (correctness): 改为在 `free_from < len(kv_indices)` 且 `tail_free_start <= len(kv_indices)` 时合并为 `(free_from, len(kv_indices_full))`，否则才 `append`；该分支目前只在两段恰好相接时命中。
- CI 在 Extra 与 AMD ROCm 流水线上为红，仅以重跑回应 (testing): 未见失败原因分析或后续结论，最终由 sglang-npu-bot 合入；无法确认是环境抖动还是真实回归。
- 缺少单元测试覆盖 (testing): 以拆分自 PR#38251 的整体验证 (accuracy 截图) 替代单测，仓库内没有可回归的针对性测试。

风险与影响

- 风险：1) 共享路径回归风险：`cache_finished_req` 中释放区间合并的改动位于所有 unified radix cache 用户都会走到的代码里（不只是 NPU/C128）。合并分支的判定 `free_from < len(kv_indices)` and `tail_free_start <= len(kv_indices)` 目前等价于 `tail_free_start == len(kv_indices)`（因为 `tail_free_start >= effective_cache_len == len(kv_indices)`），一旦未来 `tail_free_start` 语义变为可以更小，这个条件会静默切换分支，语义脆弱。
- 2) 对齐顺序耦合：`for comp in self._components_tuple: effective_cache_len = comp.floor_cache_len(effective_cache_len)` 是按组件顺序串行取整。当多个组件各自声明不同粒度时，后一个组件的取整可能破坏前一个组件的对齐保证（除非粒度互为倍数）。当前只有 C128 组件覆写、其余为恒等实现，所以暂时安全，但组件体系缺少 " 如何合并多种粒度 " 的明确约定。
- 3) 与页对齐的嵌套：`floor_cache_len` 之后仍会执行 `RadixKey(...).page_aligned(self.page_size)`，两层级取整叠加。C128 粒度 `128*page_size` 恰好是 `page_size` 的整数倍，因此现在不会出问题，但这是隐含耦合，若 C128 粒度或 `page_size` 配置变化需要复核。
- 4) 缺少下界守卫与测试：`floor_cache_len` 未保证结果不低于 `req.kv.cache_protected_len`。在 `cache_finished_req` 中由 `tail_free_start = max(...)` 兜底，但 `cache_unfinished_req` 会执行 `kv_indices = kv_indices_orig[:effective_cache_len]`，若取整后短于受保护前缀，`prefix_indices` 与受保护 KV 之间的一致性没有测试保护。本次未新增任何单测，`floor_cache_len` 的 EAGLE/ 非 EAGLE 分支和区间合并分支均无覆盖。

5) CI 证据不足: PR body 中 Extra 与 AMD ROCm 10 流水线为红, 作者只做了重跑, 无法从材料判断是环境抖动还是真实回归。

- 影响: 对用户: NPU 上运行 DeepSeek V4 (尤其启用 C128 sidecar 与 unified cache) 的部署是直接受益方, 缓存长度截断更精确, 可避免半页 C128 数据被误当成完整页缓存、以及释放区间重叠导致的 KV 池重复回收; 普通 GPU 部署因基类默认恒等实现、C128 组件不参与, 长度协商路径行为不变。

对系统: `unified_radix_cache.py` 是统一缓存的核心文件, 本次给 "组件体系" 补上了第二阶段长度协商契约 (min $\rightarrow$ floor), 后续任何带对齐要求的组件 (对比特对齐、块量化缓存、分页 sidecar) 都可以沿用这个钩子, 属于可复用的架构接口。释放区间合并则修正了

`free_kv_row` 的输入约定 (升序、不重叠), 对内存池正确性是正收益。

对团队: 改动仅 24 行、3 个文件, review 成本低, 但两位 reviewer 都是 NPU/ 仓库机器人侧的快速批准, 缺少对共享释放路径的独立审视; 同时没有单测沉淀, 后续重构 (如 unified radix cache 继续演进) 时这个修复容易被无声破坏。

- 风险标记: 核心路径变更, 缺少测试覆盖, CI 未全绿, 对齐粒度耦合, 释放区间语义脆弱

关联脉络

- PR #38251 unified cache 与 DeepSeek V4 的 C128 兼容性改动 (父 PR): PR body 明确写明 "splited from: <https://github.com/sgl-project/sglang/pull/38251>", 本次修复是该改动的拆分落地, 遵循同一功能线。
- PR #39567 [HiCache] Forward prefix metadata to v2 storage calls: 同为 KV cache 元数据 / 前缀信息在调用链上传递不完整导致的正确性修复 (`cache_controller.py`、`hybrid_cache_controller.py`), 与本次 "组件协商长度未传达到统一缓存" 属同类问题模式。
- PR #39295 [SRT] Clean up no-op compiler pass, dead helpers, and migration tests: 同样触及 `python/sglang/srt/mem_cache` 目录 (`buffer_mode/buffer_page_cache.py`) 并伴随大批量测试清理, 说明该目录近期处于持续演进 / 收敛状态, 本 PR 的改动需要关注后续重构是否会破坏这套对齐契约。