PR #29173 完整报告

sgl-project/sglang

feat: Session-reference-aware Unified Radix Cache for agentic multi-turn workloads

合并时间：2026-08-02 13:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29173

执行摘要

- 一句话：会话感知驱逐的 Unified Radix Cache，提升 agentic 多轮命中率
- 推荐动作：值得精读。重点学习：UnifiedLRUList 的 mid/cursor 分区哨兵设计（O(1) 未引用优先驱逐）、软保护（soft-protected）与硬 pin 的语义区别、SWA 按窗口长度精确记账避免重复引用、跨组件原子驱逐保护（_can_evict_leaf_atomically）、以及用 session generation + closed tombstone 解决 in-flight 竞态的手法。同时留意两个已承认的 follow-up：会话引用 TTL、close 后分区切换导致的 recency 失真。

功能与动机

PR body 明确说明：在多轮推理（agentic workflow、RL rollouts）场景中，活跃会话跨轮次持续积累可复用 KV 前缀；内存压力下标准 radix cache 只按配置的缓存策略驱逐，不考虑前缀是否仍被活跃会话引用，可能逐出活跃会话前缀却保留无关未引用 KV，导致后续轮次重算、命中率和吞吐下降。因此需要让驱逐排序感知会话引用（is_referenced 优先于 base_eviction_priority），并让会话关闭只解除引用、保留 KV 供常规回收复用。

实现拆解

1. 配置与入口改造：server_args.py 保留 --enable-session-radix-cache 开关并移除“必须配合 --radix-eviction-policy priority”的旧约束；mem_cache/registry.py 新增校验，仅 UnifiedRadixCache 允许开启（否则启动告警）；scheduler.py 中 open_session / close_session 转发到缓存的 open_radix_session / release_radix_session，并增加 init_session_radix_cache_guard 兜底提示。
2. 会话引用追踪核心：新增 unified_cache/session_ref_tracker.py，以 UnifiedSessionRefTracker dataclass 组合进 UnifiedRadixCache（遵守仓库 general-code-style 的“不用 mixin”约定），统一管理 closed-session tombstone（上限 8192，防 close 后迟到请求重新挂引用）、会话代际 generation（open 时递增，旧在途请求注册被拒）与 register_session_ref / release_radix_session 入口。
3. 组件级引用记账：TreeComponent 抽象类增加 per-session 叶子索引 _session_leaves、ComponentData.session_ref / session_ids，以及 register_session_leaf、release_session、discard_deleted_session_leaf（节点被驱逐时引用回退到最近可复用祖先）与 validate_session_state 一致性校验。FULL 组件把整条路径计入 session_ref（advance/recede 时仅更新叶子到旧祖先之间的增量区间）；SWA 组件用 _session_leaf_covered_len 记录每个叶子实际覆盖的滑窗长度，避免多窗口重复计数；

Mamba 组件只对叶子状态计数。

4. 驱逐策略改造：unified_tree_core.py 的 UnifiedLRUList 引入“引用分区”构造（is_referenced 谓词、mid 分区哨兵、cursor 游走哨兵），device 与 host 两层 LRU 都按“未引用在前、引用在后”布局；FULL 组件的驱逐堆键改为 (is_referenced, ref_count, base_priority)，关闭功能时保持原 get_priority 直接路径；_cascade_evict 增加跨组件保护，防止无引用组件连带驱逐被引用的高优先级组件（SWA 叶子原子驱逐会连带 FULL/Mamba）。
5. 生命周期集成与清理：UnifiedRadixCache.cache_finished_req 在成功插入后调用 register_session_ref(req)（排除 FINISH_ABORT 与流式会话），并相应调整 disaggregation/prefill.py 中 finished_reason 的设置顺序；删除旧 session_radix_cache.py（121 行）与 test/manual/core/test_session_radix_cache.py，新增 registered 单测 test_session_unified_radix_cache.py（所有权、注册释放、stale generation、unreferenced-first 驱逐）与文档 docs_new/docs/advanced_features/session_radix_cache.mdx。

关键文件：

- python/sglang/srt/mem_cache/unified_cache/session_ref_tracker.py（模块 会话追踪；类别 source；类型 core-logic；符号 UnifiedSessionRefTracker, post_init, reset, session_id_for_req）：本 PR 的核心新增模块：以组合方式承载会话生命周期（open/close、generation、tombstone），是注册与释放引用的唯一入口，也是与 SessionController 对接的纽带。
- python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py（模块 树内核；类别 source；类型 core-logic；符号 UnifiedLRUList, cursor_begin, cursor_next, cursor_end）：LRU 列表引入 referenced/unreferenced 分区与 cursor 游走哨兵，device/host 驱逐两条路径都依赖它实现未引用优先；同时新增 _session_lru_predicate 与 _cascade_evict 跨组件保护。
- python/sglang/srt/mem_cache/unified_cache/components/tree_component.py（模块 组件抽象；类别 source；类型 core-logic；符号 reset_session_state, session_ref, _can_evict_leaf_atomically, _refresh_session_partition）：会话引用语义的抽象地基：per-session 叶子索引、覆盖记账抽象方法、注册 / 释放 / 回退逻辑与跨组件原子驱逐保护，FULL/SWA/Mamba 都基于它实现。
- python/sglang/srt/mem_cache/unified_cache/components/swa_component.py（模块 滑窗组件；类别 source；类型 core-logic；符号 reset_session_state, _walk_session_coverage, _inc_session_coverage, _dec_session_coverage）：SWA 的会话引用必须按滑动窗口长度精确记账（_session_leaf_covered_len），避免整条前缀路径重复计数；同时 device/host 驱逐走 cursor 双 pass。
- python/sglang/srt/mem_cache/unified_cache/components/mamba_component.py（模块 曼巴组件；类别 source；类型 core-logic；符号 _inc_session_coverage, _dec_session_coverage, _advance_session_coverage, _recede_session_coverage）：Mamba 只对叶子状态计会话引用（路径祖先不计数），并保持叶子拆分时 session_ref/session_ids 复位与驱逐 cursor 切换。

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 统一缓存；类别 source；类型 core-logic；符号 cache_finished_req, open_radix_session, ensure_session_generation, release_radix_session）：统一缓存入口：组装 UnifiedSessionRefTracker、在 cache_finished_req 成功插入后注册引用，并暴露 open/ensure/release 三个公开方法给调度器。
- python/sglang/srt/mem_cache/session_radix_cache.py（模块 旧版缓存；类别 source；类型 deletion；符号 SessionRadixCacheMixin, _reset_session_radix_state, _ensure_session_radix_state, _remember_closed_session）：旧 SessionRadixCacheMixin 整体删除（121 行），会话功能从普通 RadixCache/HiRadixCache 移出，收敛到 UnifiedRadixCache 组合实现，避免双实现漂移。
- test/registered/unit/mem_cache/test_session_unified_radix_cache.py（模块 会话缓存；类别 test；类型 test-coverage；符号 TestSessionCacheOwnership, TestRadixCacheSessionRemoval, TestSessionUnifiedRadixCache, test_register_and_release_update_full_component_reference）：会话引用的核心单测：验证只有 UnifiedRadixCache 拥有会话追踪、注册 / 释放更新 session_ref、stale generation 拒绝、以及 unreferenced-first 驱逐顺序。

关键符号：UnifiedSessionRefTracker.register_session_ref, UnifiedSessionRefTracker.release_radix_session, UnifiedSessionRefTracker.open_radix_session, UnifiedSessionRefTracker.ensure_session_generation, TreeComponent.register_session_leaf, TreeComponent.release_session, TreeComponent.discard_deleted_session_leaf, TreeComponent._can_evict_leaf_atomically, TreeComponent._refresh_session_partition, SWAComponent._walk_session_coverage, SWAComponent._inc_session_coverage, SWAComponent._dec_session_coverage, SWAComponent._advance_session_coverage, SWAComponent._recede_session_coverage, MambaComponent._inc_session_coverage, MambaComponent._dec_session_coverage, FullComponent._session_ref_eviction_strategy, UnifiedLRUList.cursor_begin, UnifiedLRUList.cursor_next, UnifiedLRUList.cursor_end, UnifiedTreeCore._session_lru_predicate, UnifiedRadixCache.cache_finished_req

关键源码片段

python/sglang/srt/mem_cache/unified_cache/session_ref_tracker.py

本 PR 的核心新增模块：以组合方式承载会话生命周期（open/close、generation、tombstone），是注册与释放引用的唯一入口，也是与 SessionController 对接的纽带。

```
"""会话引用追踪器：按 session_id 给 KV 打标签，close 时只解除引用、不立即释放 KV。

被 UnifiedRadixCache 组合（composition）而不是 mixin 继承，符合仓库 general-code-style 规则。
"""

@dataclass(kw_only=True)
class UnifiedSessionRefTracker:
```

```
"""每个组件各自维护 session_ids 与 session_ref，tracker 只负责跨组件的生命周期协调。"""

components: tuple[TreeComponent, ...]
tree_core: UnifiedTreeCore
enable_session_radix_cache: bool

def __post_init__(self) -> None:
    self.reset()

def reset(self) -> None:
    # 有界 tombstone：防止"请求在 close 之后才完成"把引用重新挂回去；
    # 8192 次 close 后最早条目被淘汰，属可接受的极端迟到竞态。
    self._closed_session_ids: OrderedDict[str, None] = OrderedDict()
    # 会话代际：每次 open 递增，旧的在途请求因 generation 不匹配被拒绝注册。
    self._session_incarnation_counter: int = 0
    self._session_generations: dict[str, int] = {}
    for component in self.components:
        component.reset_session_state()

def session_id_for_req(self, req: Req) -> Optional[str]:
    # 顶层 session_id 与流式 session 对象两种来源，统一摘出稳定 id。
    session_id = req.session_id
    if session_id is None and req.session is not None:
        session_id = req.session.session_id
    return session_id

def register_session_ref(self, req: Req) -> None:
    """请求成功落缓存后调用：把可复用叶子登记到各组件。"""
    if not self.enable_session_radix_cache:
        return
    session = req.session
    if session is not None and session.streaming:
        return # 流式会话的 KV 由 StreamingSession 自己管理

    session_id = self.session_id_for_req(req)
    if session_id is None or session_id in self._closed_session_ids:
        return # close 之后的迟到请求直接丢弃

    current_generation = self._session_generations.get(session_id)
    if current_generation is None or req.session_generation != current_generation:
        logger.warning("register_session_ref called for stale request; Skip it.")
        return # 旧代际请求，禁止污染新会话

    assert req.last_node is not None
    last_node = self.tree_core.node_by_id(req.last_node)
    if last_node is self.tree_core.root_node:
        return

    for component in self.components:
```

```
            # 各组件解析自己语义下的可复用叶子：
            # FULL 是整条前缀路径，SWA 是窗口尾部，Mamba 是叶子状态。
            leaf = component.resolve_session_leaf(req, last_node)
            component.register_session_leaf(session_id, leaf)

    def release_radix_session(self, session_id: str) -> int:
        """关闭会话：解除所有引用，KV 本身留给常规驱逐回收（软保护而非 pin）。"""
        if not self.enable_session_radix_cache or session_id is None:
            return 0
        if session_id in self._closed_session_ids:
            return 0 # 幂等：重复 close 是 no-op

        self._remember_closed_session(session_id)
        self._session_generations.pop(session_id, None) # 代际作废

        indexed = 0
        for component in self.components:
            indexed += component.release_session(session_id)
        logger.info(
            "release_session %s: indexed %d component leaves", session_id, indexed
        )
        return 0

    def open_radix_session(self, session_id: str) -> Optional[int]:
        """打开会话：清掉 tombstone 并分配新代际，旧在途请求随即失效。"""
        self._closed_session_ids.pop(session_id, None)
        self._session_incarnation_counter += 1
        self._session_generations[session_id] = self._session_incarnation_counter
        return self._session_incarnation_counter

    def ensure_session_generation(self, session_id: str) -> int:
        """未见过的 session 先隐式 open 一次，保证每个请求都有可校验代际。"""
        generation = self._session_generations.get(session_id)
        if generation is None:
            generation = self.open_radix_session(session_id)
        return generation
```

python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py

LRU 列表引入 referenced/unreferenced 分区与 cursor 游走哨兵，device/host 驱逐两条路径都依赖它实现未引用优先；同时新增 _session_lru_predicate 与 _cascade_evict 跨组件保护。

```
class UnifiedLRUList:
    """带会话引用分区的 LRU 列表。

    分区布局：[head .. mid) 是 session 引用中的节点，(mid .. tail] 是未引用节点，
    驱逐从 tail 向 head 走，先消耗未引用 KV；mid 与 cursor 是带 lock_ref 的哨兵，
    不会被当作真实候选节点。
    """
```

```
def __init__(
    self,
    component_type: ComponentType,
    tree_components: tuple[ComponentType, ...],
    use_host_ptr: bool = False,
    is_referenced: Optional[Callable[[UnifiedTreeNode], bool]] = None,
):
    self.component_type = component_type
    # 指针槽：host LRU 用偏移量避免与 device 指针在同一节点上冲突。
    self._pt: int = component_type + (_NUM_COMPONENT_TYPES if use_host_ptr else 0)
    self.head = UnifiedTreeNode(tree_components)
    self.tail = UnifiedTreeNode(tree_components)
    self.head.lru_next[self._pt] = self.tail
    self.tail.lru_prev[self._pt] = self.head
    self.cache: dict[int, UnifiedTreeNode] = {}
    self._is_referenced = is_referenced
    self.mid: Optional[UnifiedTreeNode] = None
    self.cursor: Optional[UnifiedTreeNode] = None
    if is_referenced is not None:
        # 分区与游走哨兵都加锁，驱逐遍历时不会把它们当候选节点。
        self.mid = UnifiedTreeNode(tree_components)
        self.cursor = UnifiedTreeNode(tree_components)
        for ct in tree_components:
            for node in (self.mid, self.cursor):
                node.component_data[ct].lock_ref = 1
                node.component_data[ct].host_lock_ref = 1
        self._add_node_after(self.head, self.mid)

def _add_node(self, node: UnifiedTreeNode):
    # 按 is_referenced 决定插入哪个分区：引用节点进 [head .. mid)，其余进 (mid .. tail]。
    if self._is_referenced is None or self._is_referenced(node):
        self._add_node_after(self.head, node)
    else:
        self._add_node_after(self.mid, node)

def cursor_begin(self):
    # 把游走哨兵挂到 LRU 最尾，驱逐按 tail -> head 方向推进。
    if self.cursor.lru_prev[self._pt] is not None:
        self._remove_node(self.cursor)
    self._add_node_after(self.tail.lru_prev[self._pt], self.cursor)

def cursor_next(self, *, host_lock: bool = False):
    """返回下一个可驱逐节点（跳过带锁节点）；越过 head 表示本轮走完。"""
    pt = self._pt
    ct = self.component_type
    x = self.cursor.lru_prev[pt]
    while x is not self.head:
        cd = x.component_data[ct]
        if (cd.host_lock_ref if host_lock else cd.lock_ref) == 0:
```

```
                break
            x = x.lru_prev[pt]
        if x is self.head:
            return None
        self._remove_node(self.cursor)
        self._add_node_after(x.lru_prev[pt], self.cursor)
        return x

    def cursor_end(self):
        """遍历结束摘掉哨兵，避免污染真实 LRU。"""
        self._remove_node(self.cursor)
```

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

会话引用语义的抽象地基：per-session 叶子索引、覆盖记账抽象方法、注册 / 释放 / 回退逻辑与跨组件原子驱逐保护，FULL/SWA/Mamba 都基于它实现。

```
class TreeComponent(ABC):
    # ... 既有抽象方法省略，以下为本次新增的会话引用核心逻辑 ...

    def _can_evict_leaf_atomically(self, node: UnifiedTreeNode) -> bool:
        """跨组件联动的原子驱逐保护。

        SWA 叶子被整体驱逐时会连带回收 FULL 与 Mamba；
        若本组件无引用却要连带驱逐一个引用中的高优先级组件，就拒绝。
        """
        if self.session_ref(node) > 0:
            return True # 自己被引用，驱逐后引用信息随节点删除，安全
        priority = self.eviction_priority(is_leaf=False)
        return not any(
            comp.eviction_priority(is_leaf=False) >= priority
            and comp.session_ref(node) > 0
            for comp in self.tree_core.components
        )

    def register_session_leaf(
        self, session_id: str, leaf: Optional[UnifiedTreeNode]
    ) -> None:
        """把新叶子挂到会话下；若已有祖先叶子，只增补增量区间（advance）。"""
        if (
            not self.tree_core.enable_session_radix_cache
            or leaf is None
            or leaf is self.tree_core.root_node
        ):
            return
        current_leaves = self._session_leaves[session_id]
        if leaf in current_leaves:
            return
        old_ancestor = self._nearest_session_ancestor(leaf, current_leaves)
```

```
        self._advance_session_coverage(session_id, leaf, old_ancestor)
        self._mark_session_leaf(session_id, leaf)
        if old_ancestor is not None:
            self._unmark_session_leaf(session_id, old_ancestor)

    def release_session(self, session_id: str) -> int:
        """会话关闭：对每个叶子 dec 一次，再摘除叶子标记；KV 保持可复用。"""
        leaves = tuple(self._session_leaves.get(session_id, ()))
        for leaf in leaves:
            self._dec_session_coverage(session_id, leaf)
            self._unmark_session_leaf(session_id, leaf)
        return len(leaves)

    def discard_deleted_session_leaf(self, node: UnifiedTreeNode) -> None:
        """节点被驱逐时回调：引用回退到最近的可复用祖先，保证计数不泄漏。"""
        cd = node.component_data[self.component_type]
        for session_id in tuple(cd.session_ids or ()):
            leaves = self._session_leaves.get(session_id)
            assert leaves is not None and node in leaves
            fallback = self._find_reusable_session_leaf(node.parent)
            if fallback is not None and fallback in self._session_leaves.get(
                session_id, ()
            ):
                fallback = None # 祖先已是会话叶子，无需再挂
            self._recede_session_coverage(session_id, node, fallback)
            self._unmark_session_leaf(session_id, node)
            if fallback is not None:
                self._mark_session_leaf(session_id, fallback)
```

评论区精华

> ishandhanani（Issue 评论）："IIUC the distinction seems to be:
>
> 1. RefAwareCache keeps active session KV resident under pressure. 2. SessionRadixCache tracks ownership and reclaims KV on session close. I wonder if we can share the existing session_id lifecycle instead of introducing a parallel rid lifecycle?" —— 最终方案采纳此建议，复用 req.session_id 与 SessionController 生命周期，废弃独立 rid 双优先级模型。

> krishs0404（Issue 评论）："mixed-priority refs can leak within the same session_id... the suffix can still have high_ref == 1. release_ref decrements all nodes using the session's original RefInfo.is_high." —— 作者回应低优先级会话不应发高优生成请求，应记录并管理优先级不匹配；该双优先级模型在后续重写中被整体替换为按 session_id 单一引用 + generation 方案。
>
> ishandhanani（重写后 full rereview）：指出无显式 /open_session 工作流时旧实现会跳过 generation 校验（session_generation=None），close+reopen 后旧请求仍能注册；并指出 session 注册应放在成功 cache 插入生命周期而非仅 result processor（否则 disagg prefill / dLLM 完成路径丢失引用）。head 版本已把 register_session_ref 内置进

cache_finished_req，并让 ensure_session_generation 为顶层请求预分配代际。

ispobock："Do not use mixin. Ref: .claude/rules/general-code-style.md"；同时指出未 close 的会话引用会永久保留、discard_deleted_session_leaf 只向上回退导致 refs 堆积在浅层热节点（"unreferenced first degrades back toward plain LRU"），建议 TTL 留作 follow-up；close 时 reset_node_mru 会把刚关闭的冷会话 KV 移到 MRU 端，改变 recency。作者认同后者但认为保持 O(1) 较难，同意留作后续 PR。

hzh0425：要求补充 SWE Bench 对比测试（DSV4 / Qwen3.5 的 Session+HiCache vs Plain HiCache），作者在 8xH200 上给出数据；并要求 FULL 驱逐堆用 flag 分支、不要改动原逻辑（head 已实现），同时提醒 SWA 叶子原子驱逐会连带 FULL/Mamba，需考虑跨组件 session_ref（由 _can_evict_leaf_atomically 实现）。

- 与仓库并行 session-radix-cache 工作统一 session_id 生命周期 (design): 最终方案采纳：会话参考完全绑定 req.session_id / SessionController 生命周期，废弃独立 rid 与双优先级模型，引入 session generation 提供确定性 close 语义。
- 混合优先级引用在同一 session 内泄漏 (correctness): 该双优先级模型在后续重写中被整体替换为按 session_id 单一引用 + generation 方案，泄漏问题随之消解。
- 无显式 open_session 时 incarnation 校验被绕过（P1） (correctness): head 版本通过 ensure_session_generation 为未见过的 session 预分配代际，register_session_ref 统一比较 req.session_generation，测试 test_reopen_rejects_stale_generation 覆盖。
- 会话注册应放入成功 cache 插入生命周期（P1） (correctness): head 将 register_session_ref 内置到 UnifiedRadixCache.cache_finished_req，按 finished_reason 排除 FINISH_ABORT 后注册；disaggregation/prefill.py 相应调整 finished_reason 设置顺序。
- 拒绝 mixin，改为组合（general-code-style） (design): 重写为 UnifiedSessionRefTracker dataclass 组合进 UnifiedRadixCache，删除 SessionUnifiedRadixCacheMixin；测试静态断言 mixin 已移除。
- 从未 close 的会话引用永久滞留，退化为普通 LRU（建议 TTL） (performance): 本 PR 接受现状，TTL 留作 follow-up PR。
- close 时分区切换改变 recency（reset_node_mru 移到 MRU 端） (performance): 本 PR 接受宏观 unreferenced-first 顺序，recency 保真作为已知权衡，作者建议后续讨论实现。
- SWA/Mamba 全 referenced 时驱逐 O(n) 全表扫描 (performance): head 采用 mid/cursor 分区哨兵维护 referenced/unreferenced 两个分区，未引用队列头部 O(1) 可达，缓解该问题；全部被引用时仍存在一次引用区遍历。
- SWA 关联驱逐需考虑 full/mamba 的 session_ref (design): 实现为 TreeComponent._can_evict_leaf_atomically：本组件被引用可驱逐；否则若存在更高或同级优先级且被引用的组件则拒绝原子驱逐。
- SWE Bench 额外验证（DSV4 / Qwen3.5） (testing): DeepSeek-V4-Pro batch 128 device 命中率 0.418→0.510，Qwen3.5-397B batch 32 0.050→0.336；数据发布于 PR 评论并附复现脚本。

风险与影响

- 风险：
  1. 行为兼容性：--enable-session-radix-cache 现在只对 UnifiedRadixCache 生效；旧启动命令若未启用 SGLANG_ENABLE_UNIFIED_RADIX_TREE=1，启动仅告警（scheduler.py 的 init_session_radix_cache_guard），会话标签功能会静默失效，升级用户需留意。
  2. 跨组件原子驱逐：SWA 叶子被整体驱逐时连带回收 FULL 与 Mamba，_can_evict_leaf_atomically 依赖各组件 eviction_priority 大小比较；未来调整组件优先级定义时可能误放行"引用中"的组件 KV。
  3. 引用永久滞留：session_ref_tracker.py 中从未显式 close 的会话引用不会过期（ispobock 指出），配合 discard_deleted_session_leaf 只向祖先回退，长期运行后引用会集中在浅层热节点，驱逐排序退化为普通 LRU。
  4. 性能开销：开启后 FULL 堆键为三元组，早期实测 100k 叶子堆建约 33 ms、峰值内存约 12 MiB（关闭状态下 head 已恢复直接 get_priority 路径）；SWA/Mamba 双 pass 驱逐在全部被引用场景下仍需遍历引用分区（cursor 遍历 O(n)，但未引用分区头部可 O(1) 到达）。
  5. 竞态边界：generation/tombstone 已覆盖 close 后迟到请求与 close/reopen 场景，但顶层请求依赖 ensure_session_generation 预分配代际，若未来调整 Scheduler 调用顺序（如 open 校验与 tombstone 清理顺序），仍可能引入新竞态。
  6. 内存占用：每个树节点 ComponentData 增加 session_ref 与可选 session_ids set；共享前缀大的会话会把 session_id 集合复制到多个节点，虽比旧方案轻，但在多会话共享长前缀时仍有累积开销。 - 影响：用户侧：agentic 多轮、RL rollout、SWE-agent、tool-calling 等负载显著受益——DeepSeek-V4-Pro 在 batch 128 下 device 命中率从 0.418 提升到 0.510，Qwen3.5-397B-A17B 在 batch 32 下从 0.050 提升到 0.336；普通单轮负载默认关闭无影响。系统侧：每节点增加少量引用计数内存，调度器在会话 open/close 时多一次缓存回调，FULL 驱逐堆键在开启时变大。团队侧：确立了 UnifiedRadixCache 上"组合而非 mixin"的扩展范式，并验证了会话生命周期与缓存引用一致性的通用模式，为后续 TTL、准入控制（PR body 提到的 ThunderAgent / Concur 方向）铺路。 - 风险标记：核心缓存路径变更，跨组件驱逐联动，会话引用永久滞留，close 后 recency 失真，非 Unified 后端静默降级，默认关闭的增量特性

关联脉络

- PR #33171 test: recover the config-namespace-migration deferrals: 与本 PR 同改 test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py，共同服务于 UnifiedRadixCache 的稳定性建设。
- PR #33170 config: route parallel config-leaf reads through get_parallel(): 同一批 mem_cache/UnifiedRadixCache 路径的配置读取重构；本 PR 的 enable_session_radix_cache 接入与 tree_core 初始化顺序依赖这些配置读取语义。
- PR #33168 Fix the chunked-prefix-cache gate writing config the backends never read: 同为 mem_cache 配置门控语义修正：后端实际读取的配置与写入位置不一致会导致功能静默失效，与本 PR 中 enable_session_radix_cache 在非 Unified 后端静默降级的风险同源。

- PR #33126 perf(startup): skip unused PyTorch headers for KV VMM allocator stub: 同属 mem_cache/KV 内存子系统优化脉络，反映 UnifiedRadixCache 内核持续演进的上下文。