

PR #53781 完整报告

vllm-project/vllm

[3/N] HiSparse: host-resident sparse-MLA decode hot-buffering

合并时间: 2026-09-12 12:36

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53781>

执行摘要

- 一句话: HiSparse 新增 host-resident 稀疏 MLA 解码层, GPU 压力下 spill 至主机内存
- 推荐动作: 值得精读, 尤其建议关注四点: 一是 HiSparseCoordinator 的 dirty/clean/unpinned 状态机与 transition_watermark 水位设计, 这是 write-back host 层正确性核心; 二是连接器如何把每步路由状态塞进标准 build_connector_meta 管道并设置 has_sync_kv_loads, 是理解 vLLM 连接器契约的活教材; 三是 HiSparseMLAIndexGroup.convert_decode_logical_to_physical_topk 对多 token decode (投机验证) 的逐步 swap_in 与输出槽位重排; 四是 review 中关于 connector 契约、CUDA graph replay 状态重置的争论, 能帮助理解 vLLM 调度 / 执行两阶段架构的边界。管理层应关注 njhill 提出的接口改动拆分问题, 后续 PR 宜把连接器契约变更独立成小步。

功能与动机

长上下文稀疏 MLA decode 的 KV 占用远超 GPU 显存容量, 原方案 #46326 把 KV 常驻 host, 但牺牲了设备端命中率与显存利用率。PR body 明确说明设计动机: "Unlike the original always-host-resident design in #46326, this implementation keeps as much KV as possible in the normal device cache and spills pages to pinned host memory only when GPU capacity must be reclaimed." 目标是让稀疏注意力的 indexer 缓存保持设备驻留、正常前缀可缓存, 同时把稀疏 MLA 源 KV 的容量边界扩展到 host 内存, 从而支持 GLM 等模型超长上下文 (20k/32k/60k) 的 decode 场景。

实现拆解

实现按以下 5 步拆解:

1. KV cache 布局规划 (vllm/v1/hisparsed/layout.py)。create_hisparsed_layout 把稀疏 MLA 的 cache spec 拆成四类: host-resident source group (HISParsed_SOURCE)、设备驻留 indexer group (HISParsed_INDEXER)、按 indexer 页大小分组的 resident groups (HiSparseResidentSpec) 与每请求固定容量的 hot groups (HiSparseHotSpec)。_partition_hisparsed_specs 校验 source/indexer 都存在且拒绝 DeepSeek V4 压缩几何; get_hisparsed_host_pool_bytes 从 hisparsed_host_pool_gib 导出 host 预算。
2. 调度器端协调 (vllm/v1/hisparsed/coordinator.py)。HiSparseCoordinator 持有 host 块分配、host 前缀身份与驻留状态机: _HiSparseRequestState 里每页经历 dirty (仅 GPU 副本) → clean (host 副本完成) → unpinned (分配引用释放、池可复用) 三态; _ACTIVE_TAIL_PAGES = 2 保证在飞行步写入的页不会被提前交出;

`transition_watermark = max(hot_cost, num_blocks // 10)` 控制请求开始从 host 读取的触发水位；spill 通过 `build_offload_command` 下发，且 host 页只在拷贝完成后才发布前缀哈希（`_PendingPublication`）。

3. 连接器契约与 worker 数据搬运（`vllm/distributed/kv_transfer/kv_connector/v1/hisparse/connector.py` + `worker.py`）。`HiSparseConnectorScheduler.build_connector_meta` 逐步把 offload 命令、host CoW 拷贝、row mirrors 和源块 ID 打包进 `HiSparseConnectorMetadata`，并设置 `scheduler_output.has_sync_kv_loads = True` 强制前向开始前完成加载；`HiSparseConnectorWorker` 通过 `register_kv_caches` 生命周期初始化，使用 `_DMADescriptors` 批量 DMA、`_select_written_row_mirrors` 按 slot 映射 coalesce 已写行、跨 TP 广播 host 写事件做到单写入者（`_create_hisparse_host_events`）。
4. 稀疏 MLA attention 集成（`vllm/v1/attention/backends/mla/index_group.py`、`sparse_mla_attention.py`、`flashinfer/flashmla/sparse` 各后端）。`HiSparseMLAIndexGroup` 为每层创建 host 背书的 `HiSparseCacheHandle`，decode 路径经 `convert_logical_to_physical_topk / convert_decode_logical_to_physical_topk` 把 indexer 的逻辑 top-k 位置逐个 swap_in 到 hot buffer，再交给各后端 MQA kernel；prefill 侧通过 `build_hisparse_prefill_staging_plan` 做固定容量 staged gathering，避免动态输出破坏 CUDA graph。`flashinfer_mla_sparse.py` 新增 `autotune_hisparse_decode` 直接以合法 decode 包络做 autotune，并把 warmup 与模型 dummy run 解耦。
5. P/D 与 NIXL 集成及测试配套。NIXL 仍是 P/D 传输层，decoder 在准入时选择落地策略：前缀能装进 GPU 池则写 resident 页，否则 bulk 稀疏 MLA KV 落到注册的 host 池；`nixl/base_worker.py` 为此引入混合内存类型 region 注册、按 block size 的 descriptor 缓存与失败接收延迟上报。测试配套包括 `tests/v1/e2e/general/test_hisparse.py`（spill 与前缀恢复）、`tests/v1/core/test_prefix_caching.py`（+1096 行 DMA row mirrors、异步投机镜像、indexer offload 补全等）、`tests/v1/kv_connector/unit/` 下多个 NIXL/offloading 组合用例。

关键文件：

- `vllm/v1/hisparse/runtime.py`（模块 核心运行时；类别 source；类型 core-logic；符号 `is_hisparse_decode_batch`, `_get_max_decode_query_len`, `_get_max_swap_rows`, `PagedCacheView`）：`HiSparse` 数据面核心，+1100 行：定义 `PagedCacheView`、`ResolvedHiSparseConfig`、`HiSparseHostPool`、`HiSparseCacheHandle` 与 prefill staging remap，承载 decode 批判定、hot/host 视图与销毁时 pinned 内存释放。
- `vllm/distributed/kv_transfer/kv_connector/v1/hisparse/connector.py`（模块 连接器层；类别 source；类型 core-logic；符号 `HiSparseConnectorMetadata`, `HiSparseConnectorWorkerMetadata`, `HiSparseConnectorScheduler`, `bind_coordinator`）：连接器契约边界核心：把 `HiSparse` 调度器侧 coordinator 状态（offload 命令、host CoW 拷贝、row mirrors、源块 ID）打包成标准 `HiSparseConnectorMetadata`，并通过完成计数回执驱动 spill 与 host 拷贝生命周期。
- `vllm/distributed/kv_transfer/kv_connector/v1/hisparse/worker.py`（模块 数据搬运；类别 source；类型 core-logic；符号 `_DMADescriptors`, `_SlotMappingStaging`,

_allocate_dma_descriptors, _flatten_row_mirrors) : worker 侧数据搬运主体, +832 行 : DMA descriptor 池化、slot-mapping 驱动的已写行选择与 coalesce、跨 TP 单写入者 host 事件、post-forward 批量镜像与 spill 传输生命周期。

- vllm/v1/hisparse/coordinator.py (模块 调度协调; 类别 source; 类型 core-logic; 符号 _PendingPublication, _HiSparseRequestState, _PendingSpill, HiSparseCoordinator) : 调度器端正确性核心, +752 行: 页面三态 (dirty/clean/unpinned) 状态机、spill 命令生成、host 前缀延迟发布、GPU shadow 副本采纳与 _ACTIVE_TAIL_PAGES 飞行窗口保护。
- vllm/v1/attention/backends/mla/index_group.py (模块 索引分组; 类别 source; 类型 core-logic; 符号 SparseMLAIndexGroup, HiSparseMLAIndexGroup, register_layer, prepare_for_batch) : 稀疏 MLA attention 集成的核心逻辑, +503 行: logical top-k 到 physical 位置的单次转换 (side stream + event 编排)、HiSparseMLAIndexGroup 每层 cache handle 创建, 以及多 token decode 的逐步 swap_in 与输出重排。
- vllm/v1/hisparse/binding.py (模块 缓存绑定; 类别 source; 类型 dependency-wiring; 符号 resolve_hisparse_block_size, allocate_hisparse_kv_caches, init_hisparse_kv_cache, release_hisparse_profiling_cache) : worker 侧缓存初始化与绑定的唯一入口: host 池与设备 backing 分离分配、resident/hot view 绑定、布局一致性校验, 以及 profiling 后 pinned 状态释放。
- vllm/v1/hisparse/layout.py (模块 缓存布局; 类别 source; 类型 dependency-wiring; 符号 HiSparseLayout, get_hisparse_kv_cache_groups, get_hisparse_host_pool_bytes, _partition_hisparse_specs) : KV cache 布局规划: 把 MLA spec 拆成 source/indexer/resident/hot 四类 group, host 预算换算 host 块数, 并约束 block size 与 block-outermost 布局。
- vllm/v1/core/single_type_kv_cache_manager.py (模块 缓存管理; 类别 source; 类型 core-logic; 符号 HiSparseSourceManager, take_new_block_ids, bind_host_pool, take_host_cow_copies) : KVCacheManager 基座改动 (+405/-3) : 新增 HiSparseSourceManager (host 私有池、延迟发布哈希、外部导入)、HiSparseHotManager (hot 区准入时机) 与 HiSparseResidentManager (可驱逐页), 并支持 SharedEventQueueBlockPool。
- vllm/distributed/kv_transfer/kv_connector/v1/nixl/base_worker.py (模块 传输层; 类别 source; 类型 core-logic; 符号 get_finished, get_transfer_results, _sync_device_after_direct_recv, _report_failed_recv) : NIXL worker 的混合内存 region 注册、按 block-size 的 descriptor 缓存, 以及失败接收延迟上报 (_failed_recv_pending/_failed_recv_reported) ; 是 P/D 集成与多 group 正确性讨论的集中点。
- tests/v1/e2e/general/test_hisparse.py (模块 端到端测试; 类别 test; 类型 test-coverage; 符号 test_hisparse_spill_and_prefix_restore, _num_hisparse_spills, _offload_load_bytes, record_fullgraph) : HiSparse 端到端主测试: spill 与前缀恢复、offload 字节计数、CUDA graph record_fullgraph 与 finish_forward 生命周期检查。
- tests/v1/core/test_prefix_caching.py (模块 前缀缓存测试; 类别 test; 类型 test-coverage; 符号 make_hisparse_kv_cache_config, make_hisparse_kv_cache_manager, test_hisparse_builds_dma_row_mirrors_across_pages, test_hisparse_async_speculation_mirrors_uncertain_position_range) : 前缀缓存

与 HiSparse 组合的最大测试集 (+1096 行)：DMA row mirrors 跨页构建、异步投机镜像范围、host 前缀由 indexer offload 补全、缺失 host KV 时 connector 补全部分前缀等。

关键符号：HiSparseConnectorScheduler.build_connector_meta,
HiSparseConnectorScheduler.update_connector_output,
HiSparseConnectorWorker.register_kv_caches, HiSparseConnectorWorker.initialize,
HiSparseConnectorWorker._select_written_row_mirrors,
HiSparseCoordinator.commit_computed_blocks,
HiSparseCoordinator.take_host_block_copies,
HiSparseCoordinator.build_offload_command,
HiSparseMLAIndexGroup.convert_decode_logical_to_physical_topk,
HiSparseMLAIndexGroup.register_layer, HiSparseSourceManager.allocate_external_computed_blocks, HiSparseSourceManager.cache_blocks, create_hispase_layout,
bind_hispase_kv_caches, resolve_hispase_block_size, is_hispase_decode_batch,
check_hispase_host_memory, autotune_hispase_decode,
NixlBaseConnectorWorker._report_failed_recv,
NixlBaseConnectorWorker.get_transfer_results

关键源码片段

[vllm/distributed/kv_transfer/kv_connector/v1/hispase/connector.py](https://github.com/vllm-project/vllm/blob/main/vllm/distributed/kv_transfer/kv_connector/v1/hispase/connector.py)

连接器契约边界核心：把 HiSparse 调度器侧 coordinator 状态（offload 命令、host CoW 拷贝、row mirrors、源块 ID）打包成标准 HiSparseConnectorMetadata，并通过完成计数回执驱动 spill 与 host 拷贝生命周期。

```
class HiSparseConnectorScheduler:
    """调度器侧连接器：把 coordinator 的每步状态打包成标准连接器元数据。"""

    def __init__(self, *, async_speculative: bool, draft_kv_lookahead: int = 0) -> None:
        self.coordinator: HiSparseCoordinator | None = None
        self.async_speculative = async_speculative
        self.draft_kv_lookahead = draft_kv_lookahead
        self.requests: dict[str, Request] = {}

    def build_connector_meta(self, scheduler_output: SchedulerOutput) -> HiSparseConnectorMetadata:
        assert self.coordinator is not None
        # HiSparse 每一步都会重建 worker 侧路由状态，因此加载必须在前向开始前
        # 启动，不能像普通 offloading 连接器那样推迟到前向之后。
        scheduler_output.has_sync_kv_loads = True
        scheduler_output.block_table_updates = (
            self.coordinator.take_block_table_updates() or None
        )
        command = self.coordinator.build_offload_command()
        host_block_copies = self.coordinator.take_host_block_copies()
        source_group_id = self.coordinator.host_group_id
        assert source_group_id is not None
```

```

# 收集 host 源组中本次调度新 requests 与缓存命中请求占用的块 ID,
# worker 据此知道哪些 host 行可以作为本次镜像的源。
source_block_ids = [
    block_id
    for request in scheduler_output.scheduled_new_reqs
    for block_id in request.block_ids[source_group_id]
]
for new_block_ids in scheduler_output.scheduled_cached_reqs.new_block_ids:
    if new_block_ids is not None:
        source_block_ids.extend(new_block_ids[source_group_id])
num_computed_tokens = {
    request.req_id: request.num_computed_tokens
    for request in scheduler_output.scheduled_new_reqs
}
num_computed_tokens.update(
    zip(
        scheduler_output.scheduled_cached_reqs.req_ids,
        scheduler_output.scheduled_cached_reqs.num_computed_tokens,
    )
)
scheduled_requests = tuple(
    (
        request_id,
        num_computed_tokens[request_id],
        scheduled_count,
    )
    for request_id, scheduled_count in scheduler_output.num_scheduled_tokens.items()
)
# row_mirrors 描述每请求本步需要镜像到 host 的行区间; 异步投机场景下
# 起点回退到未解析的 placeholder 位置, 确保镜像覆盖 draft 与实际
# target 都可能写入的行。
row_mirrors = {}
for request_id, scheduled_start, scheduled_count in scheduled_requests:
    mirror_start = scheduled_start
    if self.async_speculative:
        mirror_start = max(
            0,
            scheduled_start - self.requests[request_id].num_output_placeholders,
        )
    row_mirrors[request_id] = self.coordinator.build_row_mirrors(
        (
            (
                request_id,
                mirror_start,
                scheduled_count + scheduled_start - mirror_start + self.draft_kv_lookahead,
            ),
        ),
    )
return HiSparseConnectorMetadata(

```

```

        command,
        host_block_copies,
        tuple(source_block_ids),
        row_mirrors,
        self.coordinator.all_context_pages_resident(scheduled_requests),
    )

```

vllm/v1/hisparse/coordinator.py

调度器端正确性核心，+752 行：页面三态（dirty/clean/unpinned）状态机、spill 命令生成、host 前缀延迟发布、GPU shadow 副本采纳与 `_ACTIVE_TAIL_PAGES` 飞行窗口保护。

```

@dataclass
class _HiSparseRequestState:
    """单个请求的驻留状态机。

    resident 页依次经历三态：dirty（只有 GPU 副本）、clean（host 副本完成，
    记录在 ``valid_pages``）、unpinned（clean 且分配引用已释放，池可复用，
    记录在 ``unpinned_pages``）。``pinned_clean`` 只存在于请求可以从 host
    读取之前，此时 clean 页仍需持有引用。
    """

    valid_pages: set[int] = field(default_factory=set)
    ready_prefix_pages: int = 0
    pending_pages: dict[int, int] = field(default_factory=dict)
    publication: _PendingPublication | None = None
    copies_recorded_blocks: int = 0
    pinned_clean: set[int] = field(default_factory=set)
    unpinned_pages: set[int] = field(default_factory=set)


class HiSparseCoordinator:
    """调度器侧 host 分配、发布、spill 与 GPU 驻留管理。

    GPU resident 页是 host 层的 write-back 缓存：host 副本持久化且所有者
    可读（已分配或已请求 hot 区）后，通过 ``BlockPool.unpin_blocks`` 释放
    引用——块仍留在块表里可读，但池将其计为空闲并可能再分发，此时所有者
    页被置空并重新发布块表。
    """

    def __init__(
        self,
        kv_cache_config: KVCacheConfig,
        managers: tuple[SingleTypeKVCacheManager, ...],
        max_model_len: int,
    ) -> None:
        # 收集 resident / hot 管理器并回填 coordinator 引用，供后续页回收回调。
        self.resident_managers = ...
        self.hot_managers = ...
        self.host_manager: HiSparseSourceManager | None = None

```

```

self.host_group_id: int | None = None
self.max_spill_pages = max_model_len // resident_block_size
self.gpu_pool = self.resident_managers[0].block_pool
# 共享池水位低于 transition_watermark 时, 请求开始从 host 读取,
# 这样新准入能找到可驱逐页而不是被钉住的页。
hot_cost = sum(manager.blocks_per_request for manager in hot_managers)
self.transition_watermark = max(hot_cost, kv_cache_config.num_blocks // 10)
# host 块哈希 -> GPU 副本页: 让 host 前缀命中能回到 GPU-resident,
# 形成第一层免拷贝回收。
self.copies: dict[BlockHashWithGroupId, tuple[KVCacheBlock, ...]] = {}
# 未 pin 的 GPU 块 -> 仍在读它的 (request, page) 对。
self._owners: dict[int, set[tuple[str, int]]] = {}

def commit_computed_blocks(self, request_id: str, num_host_pages: int) -> None:
    """记录请求命中的 host 前缀页, 并尝试采纳对应的 GPU shadow 副本。"""
    if not self.resident_managers or self.host_manager is None:
        return
    host_blocks = self.host_manager.req_to_blocks.get(request_id, ())
    num_host_pages = min(num_host_pages, len(host_blocks))
    if num_host_pages == 0:
        return
    state = self._get_request_state(request_id)
    state.valid_pages.update(range(num_host_pages))
    state.ready_prefix_pages = max(num_host_pages, state.ready_prefix_pages)
    # 命中长度完全 reconciled 后才做逐页散列采纳, 避免影响前缀命中本身。
    self._adopt_copies(request_id, state, host_blocks[:num_host_pages])
    state.copies_recorded_blocks = max(state.copies_recorded_blocks, num_host_pages)

```

评论区精华

Review 中最有价值的交锋集中在三处:

1. 连接器契约设计之争。NickLucche 明确反对 HiSparse 绕过标准接口: "the HiSparse connector currently bypasses the connector contract instead of implementing it. It is auto-injected, exposes custom hooks (apply_scheduler_output, finish_forward, set_request_state_indices, etc.) to ActiveKVConnector", 并给出应遵循 `build_connector_meta` 标准管道、按角色实例化 scheduler/worker 的示例。作者随后多轮重构 (commit [d01e49708](#)、[fd61a3531](#)) 把命令改走标准 connector metadata 通道。
2. 合入节奏争议。njhill 在合入前提出尖锐质疑: "I feel that this was rushed in too quickly given how invasive it is. In particular the nontrivial changes to the connector interface and corresponding changes to model runner etc. Could we not have at least separated the kv connector changes into a follow-on PR". LucasWilkinson 最终以赶 v0.30 为由同意合入: "in the interest of getting this in v0.30 i think we can land this and do further cleanup as followups", 该接口层面遗留问题被记 TODO。
3. 实测暴露的正确性问题。samsja 报告高并发 (1024 请求、64/GPU) 下 GPU KV 填满后出现 KV 破坏, 怀疑 "full CUDA graph replay skips the Python setup for eager host mirroring: each step clears the mirror metadata, replay leaves it unset";

MatthewBonanni 确认是 blog 相关 rework 引入并已修复。LCAIZJ 在 H20 上发现三个阻塞问题: `tokens_per_state` 未实现导致启动 `NotImplementedError`、首次 forward 崩溃 "`HiSparse GPU slots have no scheduler mapping envelope`", 以及 GLM 打包缓存池导致 NIXL region 几何失配。

- HiSparse 连接器绕过标准连接器契约 (design): 作者多轮重构 (commit d01e49708、fd61a35318) 把命令与镜像数据改走标准 connector metadata 通道, 移除 ActiveKVConnector 特化绑定。
- PR 体量过大、连接器接口改动应拆分 (design): LucasWilkinson 以赶 v0.30 为由同意合入并承诺 followup: "We can land this and do further cleanup as followups"; 接口层遗留被记 TODO。
- 全量 CUDA graph replay 跳过 eager host mirror 状态导致 KV 破坏 (correctness): MatthewBonanni 确认是 blog 相关 rework 引入并已修复; samsja 另提了 PR#56158 作为替代修复。
- NIXL 失败接收只无效化第 0 组的块 (correctness): 建议改为遍历 `meta.local_block_ids` 全部 group 取并集; PR 合入前未见作者明确修复确认。
- `get_transfer_results` 缺少 PCP producer 过滤 (correctness): 建议在 `kv_role == "kv_producer"` 且 `pcp_rank > 0` 时清空 `results.finished_sending` 后返回; 未确认合入前修复。
- SlotMappingMode 枚举是否为无关重构 (style): 作者后续将其归为 cruft 并移除无关改动, 保留 HiSparseHotSpec/CircularBufferSpec 走 SlotMappingMode.NONE 的必要分支。
- adaptive verification 的 `max_decode_tokens` 是否影响非 HiSparse 成本曲线 (correctness): 随后改为仅在 HiSparse 场景限制, 并恢复完整 adaptive verification profiling (commit 5fcf84d8e、7aa100d0e)。
- `tokens_per_state` 未实现导致引擎启动崩溃 (correctness): 社区实测反馈; PR 正文与 review 均未见针对该问题的明确修复提交, 属于合入前已知边界。
- `build_row_mirrors` 中单个请求无块表会丢弃全批镜像 (correctness): 建议将 `return ()` 改为 `continue`; 未确认合入前修复。
- `num_gpu_blocks_override` 不应作用于 host pool (correctness): 建议 `override` 只作用于 `gpu_num_blocks`, host 数保持由 `host_budget` 推导; 未确认合入前修复。
- warmup 应避免 dummy runs 并覆盖 Triton compile key (testing): 采纳建议, `sparse_mla_triton_warmup` 使用 TritonWarmupTensor 预编译 index 转换内核。

风险与影响

- 风险:
 1. 连接器接口契约被拓宽: `vllm/distributed/kv_transfer/kv_connector/v1/base.py` 新增 `finish_forward/reset_capture_state/get_transfer_results/bind_kv_cache_manager` 等抽象方法, ActiveKVConnector 与 MultiConnector 同步改动。任何第三方连接器实现都会受影响, njhill 与 NickLucche 均认为应与 HiSparse 本体拆分。
 2. NIXL 失败接收路径正确性: coderabbitai 指出 `_report_failed_recv` 只无效化 `meta.local_block_ids[0]`, 多 group 布局下 1..N 组的块仍被标记有效, 可能被调度器当

作已计算前缀复用；同时 `get_transfer_results` 缺少 `get_finished` 已有的 PCPproducer 过滤，非零 `pcp_rank` 的 `finished_sending` 会导致聚合器过早判定发送完成。

3. CUDA graph replay 状态残留：samsja 实测的 host mirror 元数据未重置问题说明，任何在 eager segment 之外维护的每步状态都可能被 breakable/full graph replay 跳过，属于该类系统的高发风险面。
4. HiSparse 与既有连接器组合的回归面：nixl/base_worker.py 的 block-size 解析从单一 `cache_config.block_size` 改为按 transfer group 取 LCM，offloading/offloading scheduler 的 group 索引语义也做了调整 (`group_idx=group_id`)，非 HiSparse 场景存在行为漂移风险；mooncake store 的 region 注册分支亦被改动。
5. 配置边界：coderabbitai 发现 `num_gpu_blocks_override` 同时覆盖 host pool 块数，导致 operator 显式钉住 GPU 容量时 host 池被错误缩放；auto-fit 探测只传 `kv_cache_groups[0]` 会低估非 MLA 组的 GPU stride，可能启动期内存估算失败。- 影响：影响范围是系统级的：调度器 (`scheduler.py` 增加 HiSparse coordinator 绑定)、KV cache 管理器 (`single_type_kv_cache_manager.py` 新增 `HiSparseSourceManager/HiSparseHotManager/HiSparseResidentManager`)、attention 元数据契约 (`sparse_mla_attention.py` 的 `SparseMLAPrefillMetadata.host_staging_plan`、`SparseMLACommonMetadata.decode_max_query_len`)、多个 FlashInfer/FlashMLA 稀疏后端、NIXL 与 mooncake 连接器以及 Model Runner V2 的 `SlotMappingMode` 与 `skip_attn` 均被改动。对用户而言，启用 `hisparsed_config + HiSparseConnector` 后可在不换硬件的情况下把长上下文 decode 的 KV 容量扩展到 host 内存 (B300 上 GLM 20k/32k/60k 均有收益)，但配置组合校验严格 (不支持 DeepSeek V4、要求 HMA 布局 BLHNC)；对团队而言，这是 mrv2 与 kv-connector 两条主线的交叉重构，后续 HiSparse 系列 PR (#53779/#53780/#53263/#53782/#53592) 均依赖本 PR 的接口基座。合入前 138 个提交、多轮主线 rebase 与反复 CI 修复也说明维护成本较高。- 风险标记：连接器接口契约变更，CUDA graph replay 状态残留，高并发 KV 破坏实测，NIXL 失败接收路径缺陷，跨模块核心架构变更，超大 PR review 覆盖不足

关联脉络

- PR #46326 HiSparse original always-host-resident design: 本 PR 明确取代 #46326 的 always-host-resident 设计：改为 GPU 优先、压力下 spill 到 host，是同一功能线的设计演进。
- PR #53779 Connector-external transferable cache groups: PR body 列出的前置依赖之一：让连接器可选择外部可转移 cache group 并保留调度器 group ID，是本 PR P/D 集成的先决条件。
- PR #53780 NIXL region metadata (stride/capacity/group mapping): PR body 列出的前置依赖：为每个 NIXL region 携带 stride、容量、块大小与 group 映射，使 P/D 两端不同布局与内存放置可对齐。

- PR #53263 NIXL host-read staging through decoder GPU memory: PR body 说明的独立可选优化: 同节点 P/D 的 D2H 读经 decoder GPU 内存暂存, 绕开 UCX TCP loopback; 不阻塞本 PR 设计。
- PR #53782 HiSparse metrics split: PR body 声明指标拆到 #53782, 本 PR 不包含 metrics 变更。
- PR #53592 DeepSeek V4 compressed cache geometry split: PR body 声明 DeepSeek V4 压缩缓存几何拆到 #53592; 本 PR 的 layout 显式拒绝 deepseek_v4。
- PR #56109 HiSparse connector binding and eviction design (Lucas Wilkinson): 多个提交 (8d2fca6b、0dfd89dc、2322e69e) 明确适配 Lucas Wilkinson 的 connector 绑定与 resident 页驱逐设计, 是本 PR 的重要设计来源。
- PR #56158 Alternative fix for HiSparse host mirroring under graph replay: samsja 在讨论中提到自己另写的修复 PR, 与本 PR 的 CUDA graph replay host-mirror 状态的修复目标相同。
- PR #50175 Triton compile-key coverage for index conversion kernels: LopezCastroRoberto 在 review 中建议本 PR 的 triton_convert_req_index_to_global_index 新调用与 #50175 的 compile-key 覆盖对齐。
- PR #57027 [Bugfix][HiSparse] Preserve per-layer offsets in KV cache bindings: 合入后的 HiSparse bugfix: 修复跨层 KV 覆盖, 逐层计算绑定偏移, 直接继承本 PR 的 binding.py 设计。
- PR #57041 [Config] Infer HiSparse attention config from HiSparseConnector: 合入后的配置演进: 从 HiSparseConnector 自动推断 attention 配置, 免重复传参, 属于本 PR 引入配置面的后续简化。