

PR #36651 完整报告

sgl-project/sglang

[Qwen3.8-Next] Add PD state transfer for Flash Next

合并时间: 2026-09-12 13:31

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/36651>

执行摘要

- 一句话: 为 Qwen3.8-Next 补齐 PD 非 KV 状态传输
- 推荐动作: 值得精读, 建议把时间放在三处: (a) `should_send_replicated_state` 的写者选举数学, 评论者已与 `CommonKVManager._resolve_rank_mapping` 对齐验证, 是后续任何复制态传输的模板; (b) `_iter_transfer_state_entries` 的 layer-major 重构与 `_slot_siblings` 委托, 它把 PLE 状态无侵入地并入既有 RDMA 注册; (c) GPUDirect flush 从“加”到“删”的完整论证链, 体现了把内存序责任归还传输后端的取舍。工程上建议补一张“后端 × 状态类型 × 是否支持异构 TP / PP>1”的能力矩阵, 并在启用 Qwen3.8 PD 时先固定 Mooncake + PP=1; 若计划用 NIXL/MORI 承载该模型, 需要先补齐 `peer_state_layer_ids` 交换再评估放开守卫。

功能与动机

PR body 明确指出: Qwen 3.8 Flash Next 除 KV cache 外还有“non-KV recurrent and sparse-attention state that must move from prefill to decode together with the normal KV cache. Without an explicit state protocol, PD disaggregation cannot preserve aggregate-mode outputs.”这不是纯理论需求: 评论者 Leslie360 在 8xA800 (SM80) TP8、Mooncake RDMA、1P1D 的真实部署里独立复现了崩溃——“during decode CUDA-graph capture at startup: `ple_state_pool.py` line 183 in `get_context` / `assert self.context is not None` / `AssertionError`”, 根因是 decode 侧 `HybridMambaDecodeReqToTokenPool` 未传 Qwen4 PLE 的 n-gram kwargs。同时 PR body 说明这是 #37500 (Qwen 3.8 Flash Next 基座支持, 取代 #36497) 的 follow-up, 刻意把 diff 收窄在 PD 分离范围内。

实现拆解

1. 状态池侧把三类状态变成可注册、可寻址的传输对象 (`python/sglang/srt/mem_cache/qsas/_kv_pool.py`、`ple_state_pool.py`): 新增 `QSA_ROPE_STATE_LAYER_ID = (1 << 32) - 2` 与 `PLE_NGRAM_STATE_LAYER_ID = (1 << 32) - 1` 两个请求级 sentinel (协议用 uint32 序列化 layer id, 必须互不冲突), 并补齐 `_get_paged_state_buf_infos`、`get_qsa_pending_state_buf_infos`、`get_qsa_pending_state_layer_ids`、`get_qsa_compressed_state_layer_ids`、`get_qsa_compressed_state_buf_infos` 五个注册口; QSA 的 pending ring / RoPE position / compressed-K 现在必须包在父池 `full_kv_pool` 的 memory-saver region 与 `torch.cuda.use_mem_pool(...)` 内, 否则 MNNVL 无法解析这块普通 CUDA 分配。`get_qsa_pending_state_buf_infos` 在

`full_attention_layer_id_mapping` 为空时返回空 triplet，避免不含 QSA 层的 PP stage 与持有 QSA 的 stage 竞争写共享 RoPE ring。

2. 线性状态迭代器重构，让 PLE 状态搭上 Mamba slot 生命周期（`python/sglang/srt/mem_cache/memory_pool.py`）：`_iter_transfer_state_tensors` 改名为语义不同的 `_iter_transfer_state_entries`，由 tensor-major 改为 layer-major，每条 entry 携带全局 `mamba_layer_ids`；`get_contiguous_buf_infos`、`get_state_layer_ids`、`get_state_slice_outer_counts`、`get_state_conv_shard_groups` 全部改为 append 语义（不再 `* num_mamba_layers`），并新增 `for sibling in self._slot_siblings: yield from sibling.iter_transfer_state_entries()` 把 PLE 的 ShortConvPool / NGramPool 状态并入同一份 RDMA 注册元数据。这同时回答了 Leslie360 的疑问：decode 侧 n-gram history 来自 prefill 的真实历史，而非 EOS 初始化。
3. 线协议与索引计算（`python/sglang/srt/disaggregation/utils.py`、`base/conn.py`）：新增 `StateType.QSA_PENDING / QSA_COMPRESSED`，`setup_state_kv_args` 在 `QSA tokenToKVPool` 分支追加这两个 component 并带 `layer_ids`；新增 `get_qsa_pending_state_indices(req)` 从 `req.kv.req_pool_idx` 取请求槽行（原实现误读顶层属性，会在构建 `state_indices` 时崩溃）；新增 `should_send_replicated_state`，scatter（源 TP $\leq$ 目标 TP）返回 True 表示广播，aggregation（源 TP $>$ 目标 TP）只让每 `src // dst` 组的第一个 rank 写；`compute_mamba_state_slice_byte_blocks` 增加 `dim == 0` 分支，把副本态整体拷贝为 `[(0, 0, src_item_len)]`，非选举源返回空块。
4. 三个传输后端的适配与边界（`disaggregation/mooncake/conn.py`、`nixl/conn.py`、`mori/conn.py`）：Mooncake 把 QSA 两种状态加入 `_is_generic_kvcache_state_type` 与 `_requires_exact_state_index_match`（page/ 请求列表位置敏感，禁止静默截断），在 `maybe_send_extra` 中对 QSA 放开非 MLA 异构 attention TP 限制，改为 layer-id 配对 + item_len 布局校验 + 写者选举，并以 `force_flat=True` 与 `src_layer_ids/dst_layer_ids` 走 generic 路径；NIXL 的 `_send_kvcache_generic` 新增 `src_layer_ids/dst_layer_ids/dst_item_lens`，用 layer-id 配对且以源 item_len 作目标 stride，长度不一致直接抛错，空 `layers_params` 返回 None 使空 PP 分量成为 no-op，并对 `0 in src_dims` 的副本态 Mamba 显式报错；MORI 对 QSA 保留 `PP > 1` 运行时守卫（其 wire 协议不交换 peer state_layer_ids），并在 `arg_groups/model_overrides/qwen4_exp.py` 提前 fail-fast。`prefill.py / decode.py` 各自新增 `_qsa_pending_payload()` 复用共享 helper。
5. 测试与配置配套：新增 `test/registered/unit/mem_cache/test_qsa_kv_pool.py`（注册进 base-a-test-cpu，monkeypatch 断言 memory-saver 与 custom-pool 两个 scope 同时生效、共 4 次分配）；`test_disaggregation_wire.py` 新增 `TestQwen4StateWire` 覆盖嵌套 `req.kv.req_pool_idx`、pending/compressed 分离注册（`state_item_lens[1:] == [[32, 48], [32]]`、`state_layer_ids[1:] == [[24, QSA_ROPE_STATE_LAYER_ID], [24]]`）、无 QSA 层 stage 不注册 RoPE ring、compact 列表按全局 layer id 映射、副本态 TP 策略真值表；`test_nixl_backend_basic.py` 新增 `TestNixlEmptyStateTransfer`；`test_mamba_state_transfer_buffers.py` 覆盖 sibling 声明副本态；`test_model_overrides.py` 更新 Qwen4 PD 支持与 MORI PP=1 限制。

关键文件：

- python/sglang/srt/mem_cache/qa_kv_pool.py (模块 稀疏索引池; 类别 source; 类型 core-logic; 符号 QSA_ROPE_STATE_LAYER_ID, _get_paged_state_buf_infos, get_qsa_pending_state_buf_infos, get_qsa_pending_state_layer_ids) : QSA 状态传输的能力入口: 定义请求级 RoPE sentinel、把 pending/compressed 缓冲注册为可传输对象, 并修正分配必须落在父池 memory-saver 与 Mooncake custom pool 内的约束 (否则 MNNVL 首个 QSA state page 发送失败)。
- python/sglang/srt/disaggregation/utlis.py (模块 传输编排; 类别 source; 类型 dependency-wiring; 符号 get_qsa_pending_state_indices, should_send_replicated_state, compute_mamba_state_slice_byte_blocks, setup_state_kv_args) : 线协议与索引计算的中枢: 新增 QSA pending 索引 helper、副本态写者选举, 并在 mamba 切片逻辑中引入 dim == 0 复制标记, 同时把 QSA 组件接入 setup_state_kv_args。
- python/sglang/srt/mem_cache/memory_pool.py (模块 内存池; 类别 source; 类型 core-logic; 符号 _iter_transfer_state_entries, get_contiguous_buf_infos, get_state_dim_per_tensor, get_state_layer_ids) : 混合 Mamba/PLE 状态注册重构: 迭代器由 tensor-major 改为 layer-major 并携带全局 layer id, 且委托 slot sibling 产出 PLE 状态, 是所有线性 / 递归状态 PD 传输的公共基座。
- python/sglang/srt/disaggregation/mooncake/conn.py (模块 传输后端; 类别 source; 类型 core-logic; 符号 _is_generic_kvcache_state_type, _requires_exact_state_index_match, maybe_send_extra) : 异构 attention TP 的唯一支持路径: QSA 状态改走 generic 分页传输, 新增 layer-id 配对、item_len 布局校验与写者选举, 并要求索引长度精确匹配。
- python/sglang/srt/mem_cache/kv_cache_configurator.py (模块 缓存配置; 类别 source; 类型 core-logic; 符号 _get_ple_req_pool_kwargs, _build_hybrid_mamba_decode_req_pool, _build_hybrid_req_pool) : 真实崩溃的修复点: 抽出 _get_ple_req_pool_kwargs 并同时喂给 prefill 与 decode 的请求池构造, 使 decode 侧不再丢失 PLE n-gram kwargs。
- python/sglang/srt/disaggregation/nixl/conn.py (模块 传输后端; 类别 source; 类型 core-logic; 符号 _send_kvcache_generic, maybe_send_extra) : NIXL 侧的契约收紧: _send_kvcache_generic 增加 layer-id 配对与目标 item_len 校验, 空分量成为 no-op, 并对副本态 Mamba 异构 TP 显式 fail-fast。
- python/sglang/srt/mem_cache/ple_state_pool.py (模块 线性状态池; 类别 source; 类型 core-logic; 符号 PLE_NGRAM_STATE_LAYER_ID, SlotIndexedState.iter_transfer_state_entries, ShortConvPool.iter_transfer_state_entries, NGramPool.iter_transfer_state_entries) : PLE 状态的传输来源: ShortConvPool 与 NGramPool 实现 iter_transfer_state_entries, 并定义请求级 n-gram sentinel, 回答了解码侧历史从何而来。
- test/registered/unit/disaggregation/test_disaggregation_wire.py (模块 协议单测; 类别 test; 类型 test-coverage; 符号 TestQwen4StateWire, test_qsa_pending_payload_uses_nested_request_pool_row, test_qsa_registers_request_ring_and_page_state_separately, test_qsa_stage_without_qsa_layers_does_not_register_rope_ring) : 新增

TestQwen4StateWire 精确刻画线协议契约：payload 行号、pending/compressed 分离注册、空 stage 不注册 RoPE ring、layer-id 映射与副本态 TP 真值表。

- python/sglang/srt/disaggregation/mori/conn.py (模块 传输后端；类别 source；类型 core-logic；符号 _send_swa_dsa_state)：边界声明位置：对 qsa_pending/qsa_compressed 保留 PP>1 守卫，明确 MORI 因缺少 peer state_layer_ids 而不支持该组合。
- python/sglang/srt/disaggregation/decode.py (模块 解码队列；类别 source；类型 core-logic；符号 _qsa_pending_payload, DecodeTransferQueue.init)：decode 侧 payload 构造点：新增 _qsa_pending_payload 复用共享 helper，并在评审后清理了 flush 遗留的 token_to_kv_pool 字段。
- python/sglang/srt/arg_groups/model_overrides/qwen4_exp.py (模块 模型开关；类别 source；类型 data-contract)：模型侧开关：开启 Qwen3.8 PD 支持，同时对 MORI QSA 施加 PP=1 的提前 fail-fast，把后端限制前移到启动期。
- test/registered/unit/mem_cache/test_qsa_kv_pool.py (模块 内存单测；类别 test；类型 test-coverage；符号 test_qsa_allocations_follow_parent_mooncake_scope)：新增 CPU 单测锁定 QSA 缓冲必须同时处于 memory-saver 与 custom-pool 两个作用域，防止 MNNVL 解析失败回归。

关键符号：get_qsa_pending_state_buf_infos, get_qsa_pending_state_layer_ids, get_qsa_compressed_state_layer_ids, get_qsa_compressed_state_buf_infos, _get_paged_state_buf_infos, get_qsa_pending_state_indices, should_send_replicated_state, compute_mamba_state_slice_byte_blocks, _iter_transfer_state_entries, iter_transfer_state_entries, _get_ple_req_pool_kwargs, _send_kvcache_generic, maybe_send_extra, _qsa_pending_payload

关键源码片段

python/sglang/srt/mem_cache/qsa_kv_pool.py

QSA 状态传输的能力入口：定义请求级 RoPE sentinel、把 pending/compressed 缓冲注册为可传输对象，并修正分配必须落在父池 memory-saver 与 Mooncake custom pool 内的约束（否则 MNNVL 首个 QSA state page 发送失败）。

```
# disaggregation 协议把 state layer id 序列化为 uint32: PLE 占用最大值作为
# request-wide n-gram sentinel, QSA 的共享 RoPE ring 占用其下一个取值。
QSA_ROPE_STATE_LAYER_ID = (1 << 32) - 2
```

```
class QSATokenToKVPool(HybridLinearKVPool):
    def __init__(self, *, size, dtype, page_size, full_attention_layer_ids, device, ...):
        ...
        # QSA 的 pending ring、RoPE position、compressed-K 与基础 KV、Mamba pool 一样
        # 参与 Mooncake PD 传输，必须落在父池的 memory-saver region 与 custom mem pool 内，
        # 否则 MNNVL 在发送首个 QSA state page 时无法解析这块普通 CUDA 分配。
        allocation_pool = self.full_kv_pool
        with (
```

```

allocation_pool.memory_saver_adapter.region(GPU_MEMORY_TYPE_KV_CACHE),
(
    torch.cuda.use_mem_pool(allocation_pool.custom_mem_pool)
    if allocation_pool.enable_custom_mem_pool
    else nullcontext()
),
):
self.qsa_key_state_buffer_pool = [
    torch.zeros(
        (ring_slots, self.qsa_index_kv_heads, self.qsa_index_head_dim),
        dtype=self.index_state_dtype, device=device,
    )
    for _ in full_attention_layer_ids
]
self.qsa_rope_position_buffer = torch.zeros(
    (ring_slots, 3), dtype=torch.int64, device=device
)
self.qsa_compressed_flat = torch.zeros(
    (len(full_attention_layer_ids),
     self.qsa_compressed_capacity * self.qsa_index_kv_heads * self.qsa_index_head_dim),
    dtype=self.index_state_dtype, device=device,
)
# 每层的压缩页是同一块连续分配上的 view，所有层可由一个基址寻址。
self.qsa_compressed_k_buffer_pool = [
    self.qsa_compressed_flat[i].view(
        self.qsa_compressed_capacity, self.qsa_index_kv_heads, self.qsa_index_head_dim
    )
    for i in range(len(full_attention_layer_ids))
]

```

@staticmethod

```

def _get_paged_state_buf_infos(tensors, page_size: int):
    # 每个 tensor 一条注册项：item_len 是“一行”（一个 request slot 或一个压缩页）的字节数，
    # 再乘 page_size 得到按页传输的 stride，供后端构造 RDMA 描述符。
    return (
        [tensor.data_ptr() for tensor in tensors],
        [tensor.nbytes for tensor in tensors],
        [tensor[0].nbytes * page_size for tensor in tensors],
    )

```

```

def get_qsa_pending_state_buf_infos(self):
    """Per-request pending key-state and RoPE ring transfer buffers."""
    # 不含本地 QSA 层的 PP stage 从不写这块共享 RoPE ring：若仍把它注册为传输源，
    # 就会与持有 QSA 的 stage 竞争，用零初始化或陈旧内容覆盖有效 position。
    if not self.full_attention_layer_id_mapping:
        return [], [], []
    tensors = [*self.qsa_key_state_buffer_pool, self.qsa_rope_position_buffer]
    return self._get_paged_state_buf_infos(tensors, self.qsa_compress_ratio)

```

```

def get_qsa_pending_state_layer_ids(self):
    """Global layer metadata for the compact QSA pending-state list."""
    if not self.full_attention_layer_id_mapping:
        return []
    # 全局 layer id 让发送端在 prefill PP stage 只持有层子集时按 id 而非位置配对;
    # RoPE ring 由所有 full-attention 层共享, 单独占一个请求级 sentinel。
    return [*self.full_attention_layer_id_mapping.keys(), QSA_ROPE_STATE_LAYER_ID]

def get_qsa_compressed_state_buf_infos(self):
    """Per-full-page compressed-K transfer buffers: full page 是压缩比的整数倍,
    所以一个 full KV page 精确对应一个压缩页。"""
    return self._get_paged_state_buf_infos(
        self.qsa_compressed_k_buffer_pool, self.qsa_compressed_page_size
    )

```

python/sglang/srt/disaggregation/utils.py

线协议与索引计算的中枢：新增 QSA pending 索引 helper、副本态写者选举，并在 mamba 切片逻辑中引入 dim == 0 复制标记，同时把 QSA 组件接入 setup_state_kv_args。

```

def get_qsa_pending_state_indices(req: Req) -> np.ndarray:
    """Return the request-pool row that owns a QSA pending-state ring."""
    # Req 把请求槽位嵌在 req.kv.req_pool_idx 下；早期实现误写成顶层 req.req_pool_idx,
    # 构建 QSA_PENDING 的 state_indices 时必然失败（自审发现并修复）。
    req_pool_idx = req.kv.req_pool_idx
    if req_pool_idx is None:
        raise ValueError("QSA pending-state transfer requires an allocated request row")
    return np.array([int(req_pool_idx)], dtype=np.int32)

def should_send_replicated_state(
    *, src_attn_tp_size: int, dst_attn_tp_size: int, local_tp_rank_in_group: int
) -> bool:
    """Elect writers for state replicated within an attention-TP group."""
    if src_attn_tp_size <= 0 or dst_attn_tp_size <= 0:
        raise ValueError("Attention TP sizes must be positive for replicated-state transfer")
    larger_tp_size = max(src_attn_tp_size, dst_attn_tp_size)
    smaller_tp_size = min(src_attn_tp_size, dst_attn_tp_size)
    if larger_tp_size % smaller_tp_size != 0:
        raise ValueError("One attention TP size must divide the other ...")
    if not 0 <= local_tp_rank_in_group < src_attn_tp_size:
        raise ValueError("Source attention TP rank is out of range ...")
    if src_attn_tp_size <= dst_attn_tp_size:
        # Scatter (1 个 prefill rank → 多个 decode rank) 等价于广播：源端发给每个目标注册项。
        return True
    writers_per_decode = src_attn_tp_size // dst_attn_tp_size
    # Aggregation (多个等价源副本 → 1 个 decode rank) 只允许每组第一个源 rank 写入,
    # 否则同一份副本态会被多个 rank 重复写入并互相覆盖。
    return local_tp_rank_in_group % writers_per_decode == 0

```

```

def compute_mamba_state_slice_byte_blocks(
    *, src_item_len, dst_item_len, src_dim, dst_dim, outer_count,
    src_attn_tp_size, dst_attn_tp_size, dst_tp_rank_in_group,
    local_tp_rank_in_group, conv_shard_groups=None,
):
    """ ... """
    # dim == 0 是本 PR 引入的协议标记: 该 item 在 attention TP 组内是副本, 不做 TP 切片,
    # 由选举出的源 rank 整体拷贝。PLE short-conv / n-gram 与 QSA pending / compressed
    # 都走这里。
    if (src_dim == 0) != (dst_dim == 0):
        raise ValueError("Mamba state replication metadata differs between prefill and decode")
    if src_dim == 0:
        if src_item_len != dst_item_len:
            raise ValueError("Replicated Mamba state item lengths differ between prefill and
            decode ...")
        if not should_send_replicated_state(
            src_attn_tp_size=src_attn_tp_size,
            dst_attn_tp_size=dst_attn_tp_size,
            local_tp_rank_in_group=local_tp_rank_in_group,
        ):
            return [] # 非选举源 rank 不产生任何传输块
        return [(0, 0, src_item_len)]
    ...
    # 其余状态维持原有的按 TP 切片逻辑 (GDN conv_state 按 q/k/v 子块分别切片) 。

```

python/sglang/srt/mem_cache/memory_pool.py

混合 Mamba/PLE 状态注册重构: 迭代器由 tensor-major 改为 layer-major 并携带全局 layer id, 且委托 slot sibling 产出 PLE 状态, 是所有线性 / 递归状态 PD 传输的公共基座。

```

def _iter_transfer_state_entries(self):
    """Yield ``[slot, ...]`` state entries and their transfer metadata."""
    for field, value in vars(self.mamba_cache).items():
        if field in self._NON_TRANSFER_STATE_FIELDS or value is None:
            continue
        tensors = value if isinstance(value, list) else [value]
        slice_axis = self.conv_slice_axis if field == "conv" else 0
        for state_tensor in tensors:
            # ShortConv 层没有 temporal state, 空 buffer 会让整批注册失败, 直接跳过。
            if state_tensor.numel() == 0:
                continue
            # 由 tensor-major 改为 layer-major: 每个 (tensor, layer) 组合单独成为一条 entry,
            # 并携带全局 layer id, PP 下 sender 才能按 id 而非位置配对 src/dst。
            for layer_index, layer_id in enumerate(self.mamba_layer_ids):
                yield field, state_tensor[layer_index], slice_axis, layer_id

    # PLE 的 short-conv / n-gram 状态不在 mamba_cache 中, 由 slot sibling 提供,
    # 复用同一套 slot 生命周期与同一份 RDMA 注册元数据。
    for sibling in self._slot_siblings:
        yield from sibling.iter_transfer_state_entries()

```

```

def get_state_dim_per_tensor(self):
    """Get the sliceable dimension size for each state tensor."""
    dim_per_tensor = []
    for _, state_tensor, slice_axis, _ in self._iter_transfer_state_entries():
        # slice_axis 为 None 时输出 dim == 0 协议标记：该条目在 attention TP 内是副本，
        # 异构 PD 由选举出的源 rank 整体拷贝，而不是按 TP 切片。
        if slice_axis is None:
            dim_per_tensor.append(0)
            continue
        # state_tensor 形状为 [size+1, sliceable_dim, ...]; Kimi conv state 把两个
        # per-slot 轴转置成 [K-1, dim]，因此切片轴是 1 + slice_axis。
        axis = 1 + slice_axis
        dim_per_tensor.append(state_tensor.shape[axis])
    return dim_per_tensor

def get_state_layer_ids(self):
    """全局 layer id 与 get_contiguous_buf_infos() 逐元素对齐，
    使 prefill PP stage 只持有 mamba 层子集时仍能按 id 配对条目。"""
    return [layer_id for _, _, _, layer_id in self._iter_transfer_state_entries()]

```

评论区精华

1) 自查抓出真实崩溃点：作者在 prefill.py 的 review 里指出 Req 只有 req.kv.req_pool_idx, “there is no top-level req_pool_idx, so QSA_PENDING crashes while building state_indices”, 要求同步修复 decode 侧并补一个真正执行 payload 构造的回归测试；后续以共享 helper + ReqKvInfo 真值测试在 a03da1ea5 落地。2) GPUDirect flush 的取舍是本 PR 最有价值的交锋：Leslie360 指出 MORI 传输 QSA 状态却被排除在 flush 之外，作者承认 “the flush is a memory-ordering contract of the transport backend, not a property of a state type” 且 PR 从未观测到 stale read，最终在 58581bfc 整体删除该 flush，理由是 main 上 KV/Mamba/DSA/SWA 都不做，并以 no-flush 生产 parity (12/12 用例、最大 logprob 差 0.00047) 作为证据。3) MORI PP>1 被定性为限制：Leslie360 建议 “Removing this guard alone is unsafe: MORI needs peer state_layer_ids and global-layer descriptor pairing, as Mooncake does”，作者保留运行时守卫并新增更早的模型 override fail-fast，明确异构支持是 Mooncake-only。4) ShangmingCai 指出死代码：对 DecodeTransferQueue.__init__ 新增的 self.token_to_kv_pool 提问 “Couldn't find anywhere use it except the test”，作者确认这是已删除 flush 路径的残留并在 d3eba2f695 移除，随后给出 APPROVED: “LGTM, logic is clean and correct”。5) 合并后独立复验：Leslie360 在 16xA800 SM80、TP8-per-side、Mooncake 0.3.9 over mlx5 IB 上确认 decode 节点启动无 assert、6/6 短请求与 3/3 长上下文请求通过，并从源码层回答了自己早先的疑问——_iter_transfer_state_entries (memory_pool.py:1117) 会遍历 sibling, ple_state_pool.py:129/239 分别产出 ple_short_conv 与 ple_ngram，因此 n-gram history 是真实传输而非 EOS 初始化。

- QSA payload 读取了不存在的顶层 req.req_pool_idx (correctness): 在 a03da1ea5 中抽出共享 helper get_qsa_pending_state_indices，未分配请求行时显式抛错；测试用真实

ReqKvInfo 执行 payload 构造，旧实现会失败。

- QSA 专属 GPUDirect flush 是否应存在 (design): 先在 a03da1ea5 收窄到 Mooncake/NIXL + NVIDIA CUDA 并加 fallback, 最终在 58581bfc 整体移除; 以 main 上 KV/Mamba/DSA/SWA 均不做该 flush 及 no-flush 生产 parity (12/12 用例、最大 logprob 差 0.00047) 为依据。
- MORI 的 QSA 是否支持 PP>1 (design): 保留 _send_swa_dsa_state 中的 PP>1 守卫, 并在 arg_groups/model_overrides/qwen4_exp.py 增加更早的 fail-fast; 异构 /PP 元数据支持明确为 Mooncake-only。
- DecodeTransferQueue 新增字段是否被使用 (style): 作者确认这是已删除的 QSA flush 路径残留, 在 d3eba2f695 同时移除赋值与测试中的无用准备代码; 随后 ShangmingCai 给出 APPROVED (“LGTM, logic is clean and correct”)。
- 异构 attention TP 下的副本态写者选举与布局校验 (correctness): 引入 should_send_replicated_state (scatter 全发、aggregation 只让每组首个 rank 写), Mooncake 增加 item_len 布局校验与 _requires_exact_state_index_match, NIXL 传入 dst_item_lens 并在不匹配时抛错。
- decode 侧 PLE n-gram 历史的来源 (testing): 确认为从 prefill 传输的真实历史; 16xA800 SM80、TP8-per-side、Mooncake RDMA 环境下 6/6 短请求与 3/3 长上下文请求零 assert, 确定性差异落在单机噪声范围内。

风险与影响

- 风险: 1) 写可见性完全依赖传输层完成语义: QSA 专属 GPUDirect flush 被移除后, decode 端仅在 poll == KVPoll.Success 后消费状态; 若某个后端的完成回执不隐含写可见性, pop_transferred 可能读到陈旧行。作者以“现有 KV/Mamba/DSA/SWA 均不做该 flush”和 no-flush 生产 parity 为依据, 属于被显式接受的风险。2) 状态注册顺序契约变更: _iter_transfer_state_entries 由 tensor-major 改为 layer-major 并把 sibling (PLE) 状态追加在 mamba state 之后; PP=1 时 build_transfer_entry_pairs 仍允许 allow_positional_fallback, 若 prefill/decode 两侧迭代顺序不一致会静默错配, layer-id 配对只在带 layer_ids 的路径生效。3) PP>1 缺少端到端验证: 作者自述当前模型在 PP>1 下无法进入 serving (真实 checkpoint 先命中 PLE PP 权重布局, dummy 权重命中 support_pp=False 守卫), 因此“零 QSA 层的 PP stage”只由单测覆盖, 注册与 NIXL no-op 路径未被 E2E 证明。4) 后端覆盖不均: 异构 attention TP 仅在 Mooncake 上有 parity; NIXL 对副本态 Mamba 直接抛错、对 QSA 只做长度校验; MORI 的 QSA 被限制在 PP=1。ROCm CI 有既存失败 (test_umbp_store.py 的 MagicMock.numel() TypeError), 与本 PR 无关但会干扰判断。5) 内存分配范围回归风险: QSA buffer 必须落在父池 memory-saver / custom-pool 内, custom pool 关闭时依赖 nullcontext 分支; CPU 单测用 monkeypatch 覆盖了两个 scope, 但未覆盖 MNNVL 真实路径。6) 校验严格化带来的行为变化: Mooncake 对 QSA 走 _requires_exact_state_index_match 抛错、NIXL 对 item_len 不匹配抛错, 会把以往被截断容忍的漂移升级为请求失败, 对上游 index 生成一致性提出更高要求。
- 影响: 用户侧: Qwen3.8-Flash-Next 现在能在 PD 分离下保持 aggregate 语义 (TP4→TP4 逐 token 精确一致; TP1↔TP4、MTP、PLE-offload 也有 parity 数据), 同

族混合 GDN 模型 Qwen3.5-27B 的 PD 亦经复验无回归，Triton/ 传输失败不再表现为启动期 assert。系统侧：PD 状态传输从“KV + Mamba + DSA/SWA”扩展到“索引类稀疏注意力状态 + 请求级递归状态”，协议面新增 2 个 StateType、2 个 layer-id sentinel 与 1 个 dim == 0 复制标记，Mooncake/NIXL/MORI 三个后端必须同步维护，否则会出现只在新状态类型上暴露的错配。团队侧：改动集中在 mem_cache 与 disaggregation 两条线，评审者需同时理解 QSA 池布局、PP layer-id 映射与 RDMA 注册语义；后续任何新状态类型都必须先回答“是 TP 分片还是 attention-TP 内副本”。

- 风险标记：跨后端协议契约变更，写可见性依赖传输完成语义，PP>1 缺少端到端验证，异构 TP 仅 Mooncake 覆盖，状态注册顺序重构

关联脉络

- PR #37500 Qwen 3.8 Flash Next 基座支持 (PR body 中引用的前置 PR)：本 PR body 明确说明是其 follow-up，并在其合入 main 后把 diff 重新定位、收窄到 PD 分离范围。
- PR #36497 Qwen 3.8 Flash Next PD 支持 (被 #37500 取代的旧分支)：本 PR 声明 superseding #36497；Leslie360 的外部验证最初也是基于该分支 head 加上本 PR 的补丁复现崩溃。
- PR #38402 fix(npu): fix hybrid KV transfer with PP prefill in PD disaggregation: 同属 PD 分离下的混合 / 线性状态传输线，共用 memory_pool 的传输注册与 mamba state slice 逻辑，本 PR 的 layer-id 配对与之直接衔接。
- PR #38935 [PD] Do not admit intake-rejected requests to a PD handoff: 改动同一批文件 (disaggregation/utils.py、prefill.py、decode.py)，本 PR 新增的 payload helper 正落在其修改过的 prefill/decode 传输队列中，属同模块近期演进。