

PR #56562 完整报告

vllm-project/vllm

[Perf] Fuse DSV4.1 input metadata preparation with Triton

合并时间: 2026-09-12 22:20

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56562>

执行摘要

- 一句话: DSV4.1 输入元数据改用 Triton 内核, decode TPOT 降约 15%
- 推荐动作: 值得精读, 尤其是正在做 vLLM 元数据路径优化或 CUDA-graph 兼容改造的工程师: `_token_request` 的“设备端二分 + lane 级收敛”与 `_indexer_decode_metadata_kernel` 的单内核多产物 (`seq_lens / block table / indices / per-request lens / padding` 清零) 是非常典型的“用内核替代 PyTorch 链”的设计范式, grid 公式 `max(num_decodes, num_decode_tokens + cdiv(capacity - num_decode_tokens, 256))` 同时承担“逐请求写入”和“容量尾部清零”两个职责, 值得重点关注。阅读时建议重点核对三处: (a) `num_mapped_tokens` 依旧来自 CPU 的假设边界; (b) `varlen` 下 `decode_lens_buffer` 由逐请求变逐 token 后所有消费者的适配; (c) `requires_padding = False` 的传播范围。后续建议补两件事: 把作者移出 PR 的元数据 benchmark 以 CI 形态回归 (至少覆盖 batch-16 decode 与 DSpark 自适应路径), 以及请维护者对 `vllm/v1/attention/ops/metadata.py` 的三个内核做一次独立逐行复核, 因为当前 PR 只有机器人在场、无人工评审记录。

功能与动机

PR body 写明动机: DSV4.1 在每个 model step 都用一串 PyTorch 算子构造 token-to-request 映射与扁平化 indexer decode 元数据, 需要替换为“直接写既有 buffer”的 Triton 内核。最关键的约束是 DSpark 自适应验证在设备端决定每个请求的 draft 拆分, 因此 “Device boundaries are necessary for DSpark adaptive verification, where CPU boundaries can be stale”; 同时旧实现依赖 `decode_lens_cpu.sum().item()` 之类 CPU 同步值来驱动 `repeat_interleave`。收益目标也被量化: 元数据 GPU 时间 3-4x、batch-1 DSpark 模式 TPOT 降低 15-18%、CPU `aten::repeat_interleave` 事件从 54 次降到 0。PR 明确做了去重检查 (搜索 `DeepSeekV4metadata`、`indexermetadatatriton`、`token_to_req_indices`、`metadata DSpark`), 并声明与 #56344、#56254、#52628、#55692 的工作不重叠; 未关联任何 issue。

实现拆解

1. 新增内核层: `vllm/v1/attention/ops/metadata.py` (新文件, 87 行)

- `_token_request(query_start_loc, token, num_reqs)`: 设备端向量化二分。以 `while tl.sum((lo < hi).to(tl.int32), 0) > 0` 表达“所有 lane 都收敛”才退出; `tl.load(query_start_loc + mid + 1, mid < num_reqs, other=0x7FFFFFFF)` 防越界读; 返

回 `tl.minimum(lo, num_reqs - 1)`, 使零长请求 /padding token 落回最后一个合法请求而不越界。这是把“CPU 边界可能过期”问题下沉到设备端的关键。

- `_token_request_mapping_kernel`: 每个 program 处理 256 个 token, `token < num_mapped` 写真实 request id, 其余写 0; 用 `do_not_specialize` 标注尺寸参数以避免 batch 变化导致重复编译。
- `_indexer_decode_metadata_kernel`: 一个 program 一个 token 的主分支写 `output_seq_lens` (逐 token 因果长度)、`decode_lens` (恒为 1)、`indices`、`output_block_table` 行; else 分支以 256 为粒度把 `[num_tokens, capacity)` 清零, 用于 CUDA graph 每次 replay 清除残留 padding; 末尾的 `if token < num_reqs` 由低 pid program 写 `per_req_lens`。

2. `CommonAttentionMetadata.token_to_req_indices` 换内核 (

`vllm/v1/attention/backend.py`, +13/-11)

- 旧实现是三步: 构造 `arange + torch.repeat_interleave(query_lens, output_size=num_mapped_tokens)`、`copy_` 进 buffer、必要时 `zero_()`; 新实现一次 `_token_request_mapping_kernel[((num_output_tokens + 255) // 256,))`, 网格按 `max(num_mapped_tokens, num_tokens)` 计算。
- `num_mapped_tokens` 仍取自 `int(self.query_start_loc_cpu[-1])`, 注释改为“Padding requests have a query length of zero and are skipped by the device boundary search”, 并把缓存切片同步为 `buffer[:num_output_tokens]`。影响面超出 DSV4.1: 这是各 attention backend 共用的映射入口, 缓存命中路径 (`_token_to_req_indices_cache`) 保持不变, 保证 CUDA graph replay 不重复映射。
- 采用函数内延迟 import, 避免模块导入期即拉入 Triton 内核与潜在循环依赖。

3. `indexer varlen decode` 分支的元数据融合 (`vllm/v1/attention/backends/mla/indexer.py`, +69/-52)

- 删除 `_build_varlen_decode_indices`: 该实现需要 `int(decode_lens_cpu.sum().item())` 的 CPU 同步与两次 `repeat_interleave`。
- `torch.diff(...)` 写 `decode_lens_buffer`、`per_req_decode_lens_buffer`.`copy_` 一起移入 `if not self.supports_varlen`; `_prepare_global_decode_seq_lens` 同样只在非 varlen 路径调用; `_prepare_decode_tensors` 整体移入 else 分支。
- `varlen` 分支改为一次 `_indexer_decode_metadata_kernel[(grid,)]`, 随后把 `seq_lens / block_table / decode_lens / decode_indices` 重新绑定到持久 buffer 视图, 并直接置 `requires_padding = False` (padding 已由内核清零)。
- 网格公式 `grid = max(num_decodes, num_decode_tokens + triton.cdiv(capacity - num_decode_tokens, 256))` 承担双重职责: 前半保证每个 request 都有 program 写 `per_req_decode_lens_buffer`, 后半覆盖容量尾部的清零。
- DCP 场景保留“先逐 token 展开、再本地化”的顺序: `global_decode_seq_lens_buffer[:num_decode_tokens].copy_(seq_lens)` 之后替换 `global_seq_lens_for_decode`, 避免旧注释里描述的“先按请求本地化再展开导致因果边界过短”的回归。

4. 测试配套 (`tests/v1/attention/test_indexer_deepseek_v4_slot_mapping.py`, +105)

- `test_fused_indexer_decode_metadata`: 参数化 `query_lens=[[1], [6]*16, [0,2,6,0,4], [0,0,0]] × padding=[0,17]`, 构造 65 列 (非 2 次幂, 触发 `next_power_of_2` 掩码) 的 block table 与带上限的 capacity, 断言逐 token 因果长度、尾部清零、padding 位索引等于 `num_reqs + token - actual_tokens`, 全部 `rtol=atol=0`。
- `test_device_token_request_mapping`: 断言返回 buffer 与入参 `data_ptr()` 相同, 随后捕获 CUDA graph、翻转 lengths 并就地改写 `qsl[1:]` 再 `graph.replay()`, 验证“CPU 边界过期时以设备边界为准”。

5. 工程配套与刻意未做的部分

- 第二个 commit 专门移除随 PR 的 metadata benchmark harness, 仓库内没有可复现的性能脚本, 也没有对应的 CI 性能 / 正确性门禁。
- PR body 明确“No new JIT warmup-framework wrappers or registrations are added”, 即内核走 Triton 默认 JIT 路径, 不接入共享预热契约; 非 varlen indexer 实现与 ROCm 路径均未改动, 无新增配置项或 schema 变更。

关键文件:

- `vllm/v1/attention/ops/metadata.py` (模块 元数据内核; 类别 source; 类型 core-logic; 符号 `_token_request`, `_token_request_mapping_kernel`, `_indexer_decode_metadata_kernel`): 本 PR 新增的内核模块, 承载全部新逻辑: 设备端二分的 `_token_request`、`token→request` 映射内核 `_token_request_mapping_kernel`, 以及一次性生成 SM100 varlen indexer decode 元数据并清零 CUDA-graph padding 的 `_indexer_decode_metadata_kernel`。
- `vllm/v1/attention/backends/mla/indexer.py` (模块 索引器元数据; 类别 source; 类型 core-logic; 符号 `_build_varlen_decode_indices`, `build`): DSV4.1 indexer 元数据构建主路径: varlen decode 分支改为一次 `_indexer_decode_metadata_kernel`, 删除 `_build_varlen_decode_indices`, 并把 `torch.diff / _prepare_global_decode_seq_lens / _prepare_decode_tensors` 全部收进非 varlen 分支, 是本次性能收益与语义变更最集中的文件。
- `vllm/v1/attention/backend.py` (模块 注意力元数据; 类别 source; 类型 core-logic; 符号 `token_to_req_indices`): `CommonAttentionMetadata.token_to_req_indices` 是所有 attention backend 共用的映射入口, 本次把 `repeat_interleave + copy_ + zero_` 三步换成一次 `_token_request_mapping_kernel`, 是影响面最大的一处改动, 也把“设备边界权威”写进了公共数据结构。
- `tests/v1/attention/test_indexer_deepseek_v4_slot_mapping.py` (模块 索引器测试; 类别 test; 类型 test-coverage; 符号 `test_fused_indexer_decode_metadata`, `test_device_token_request_mapping`): 新增两个 CUDA 定点测试: `test_fused_indexer_decode_metadata` 覆盖零长请求、padding、非 2 次幂列宽与尾部清零; `test_device_token_request_mapping` 用 CUDA graph replay + 就地改写 `query_start_loc` 验证“CPU 边界过期时以设备边界为准”, 是本 PR 自动化的主要正确性防线。

关键符号: `_token_request`, `_token_request_mapping_kernel`, `_indexer_decode_metadata_kernel`, `CommonAttentionMetadata.token_to_req_indices`,

DeepseekV32IndexerMetadataBuilder.build, _build_varlen_decode_indices, test_fused_indexer_decode_metadata, test_device_token_request_mapping

关键源码片段

vllm/v1/attention/backend.py

CommonAttentionMetadata.token_to_req_indices 是所有 attention backend 共用的映射入口，本次把 repeat_interleave + copy_ + zero_ 三步换成一次 _token_request_mapping_kernel，是影响面最大的一处改动，也把“设备边界权威”写进了公共数据结构。

```
def token_to_req_indices(self, buffer: torch.Tensor) -> torch.Tensor:
    """Build or reuse the per-token request index mapping."""
    num_tokens = self.num_actual_tokens
    if self._token_to_req_indices_cache is not None:
        # CUDA graph replay 命中缓存: buffer 地址与 num_tokens 都不变,
        # 因此不需要重新做设备端映射（重复映射会破坏 graph 内地址稳定性）。
        assert self._token_to_req_indices_cache.device == buffer.device
        assert self._token_to_req_indices_cache.dtype == torch.int32
        assert self._token_to_req_indices_cache.shape[0] >= num_tokens
        return self._token_to_req_indices_cache[:num_tokens]

    # Built from the device query_start_loc: adaptive verification decides the
    # per-request draft split on device, so the CPU copy carries the right total
    # but not the right per-request boundaries. Padding requests have a query
    # length of zero and are skipped by the device boundary search.
    # 注意: num_mapped_tokens 仍来自 CPU 总数，设备侧只负责“怎么分”。
    num_mapped_tokens = int(self.query_start_loc_cpu[-1])
    from vllm.v1.attention.ops.metadata import _token_request_mapping_kernel

    num_output_tokens = max(num_mapped_tokens, num_tokens)
    assert buffer.shape[0] >= num_output_tokens
    # 一次 launch 完成 repeat_interleave + copy_ + zero_ 三件事:
    # [0, num_mapped) 由设备边界二分得到 request id，其余位置写 0。
    _token_request_mapping_kernel(((num_output_tokens + 255) // 256,))(
        self.query_start_loc,
        buffer,
        self.query_start_loc.shape[0] - 1, # num_reqs
        num_mapped_tokens,
        num_output_tokens,
        num_warps=4,
    )
    # 缓存整段输出，replay 时直接切片复用。
    self._token_to_req_indices_cache = buffer[:num_output_tokens]
    return self._token_to_req_indices_cache[:num_tokens]
```

tests/v1/attention/test_indexer_deepseek_v4_slot_mapping.py

新增两个 CUDA 定点测试：test_fused_indexer_decode_metadata 覆盖零长请求、padding、非 2 次幂列宽与尾部清零；test_device_token_request_mapping 用 CUDA graph replay + 就地改写 query_start_loc 验证“CPU 边界过期时以设备边界为准”，是本 PR 自动化的主要正确性防线。

```
@pytest.mark.skipif(not torch.cuda.is_available(), reason="requires CUDA")
@pytest.mark.parametrize("query_lens", [[1], [6] * 16, [0, 2, 6, 0, 4], [0, 0, 0]])
@pytest.mark.parametrize("padding", [0, 17])
def test_fused_indexer_decode_metadata(query_lens, padding):
    """Flatten device query boundaries and clear graph padding on every replay."""
    from vllm.v1.attention.ops.metadata import _indexer_decode_metadata_kernel

    device = "cuda"
    lengths = torch.tensor(query_lens, device=device, dtype=torch.int32)
    qsl = torch.cat(
        [torch.zeros(1, device=device, dtype=torch.int32), lengths.cumsum(0).int()]
    )
    reqs = len(query_lens)
    n = sum(query_lens) # 真实 token 数
    tokens = n + padding # 模拟 graph padding 后的 token 数
    capacity = max(tokens, reqs) + 31
    seq = lengths + 4096
    # 74 列取 65 列：刻意让列宽为非 2 次幂，触发 next_power_of_2 掩码分支；
    # (-99) 初值用于检测未被内核写入的位置。
    bt = torch.arange(reqs * 74, device=device, dtype=torch.int32).view(reqs, 74)[
        :, :65
    ]
    out_bt = torch.full((capacity, 69), -99, device=device, dtype=torch.int32)
    outputs = [
        torch.full((capacity,), -99, device=device, dtype=torch.int32) for _ in range(4)
    ]
    out_seq, out_lens, indices, per_req = outputs
    grid = max(reqs, tokens + (capacity - tokens + 255) // 256)
    _indexer_decode_metadata_kernel[(grid,)](
        qsl, seq, bt, out_seq, out_bt, out_lens, indices, per_req,
        reqs, tokens, capacity, bt.stride(0), out_bt.stride(0),
        BLOCK_COLS=bt.shape[1], num_warps=4,
    )
    # 逐 token 期望：请求 id 展开、因果长度从 4097 起逐 token 递增。
    expected_req = torch.repeat_interleave(torch.arange(reqs, device=device), lengths)
    expected_seq = torch.cat(
        [torch.arange(4097, 4097 + q, device=device) for q in query_lens]
    )
    torch.testing.assert_close(out_seq[:n], expected_seq.int(), rtol=0, atol=0)
    assert torch.count_nonzero(out_seq[n:]) == 0 # padding 位清零
    torch.testing.assert_close(out_bt[:n, :65], bt[expected_req], rtol=0, atol=0)
    assert torch.count_nonzero(out_bt[n:tokens, :65]) == 0
    torch.testing.assert_close(indices[:n], expected_req.int(), rtol=0, atol=0)
    torch.testing.assert_close(
```

```

        indices[n:tokens],
        torch.arange(reqs, reqs + padding, device=device, dtype=torch.int32),
    ) # padding 走「伪请求」索引 num_reqs + token - actual_tokens
    assert torch.all(out_lens[:tokens] == 1)
    torch.testing.assert_close(per_req[:reqs], lengths)

@pytest.mark.skipif(not torch.cuda.is_available(), reason="requires CUDA")
@pytest.mark.parametrize("query_lens", [[1], [0, 257, 1, 0, 3], [6] * 64, [2, 6, 4, 0]])
def test_device_token_request_mapping(query_lens):
    """Graph replay follows device boundaries even when CPU lengths are stale."""
    # ... (构造 CommonAttentionMetadata 略) ...
    result = common.token_to_req_indices(output)
    assert result.data_ptr() == output.data_ptr() # 必须原地写回入参 buffer
    # 捕获 graph 后翻转 query 长度并就地改写 qsl: CPU 副本不再被重新读取,
    # 只有设备边界参与映射, 正是自适应验证场景的近似复现。
    graph = torch.cuda.CUDAGraph()
    common._token_to_req_indices_cache = None
    with torch.cuda.graph(graph):
        common.token_to_req_indices(output)
    reversed_lens = lengths.flip(0)
    qsl[1:].copy_(reversed_lens.cumsum(0))
    graph.replay()
    expected = torch.repeat_interleave(
        torch.arange(len(query_lens), device="cuda", dtype=torch.int32), reversed_lens
    )
    torch.testing.assert_close(output[:n], expected)
    assert torch.count_nonzero(output[n:]) == 0

```

评论区精华

本 PR 的讨论极其稀薄，Review 线程只有一条机器人消息，没有任何人工 review 评论，讨论价值主要来自作者在 PR body 中的自我披露：

claude[bot] (state=COMMENTED) : "This pull request is from a fork — automated review is disabled. A repository maintainer can comment @claude review to run a one-time review."

即自动化 review 因 fork 被跳过，维护者也没有补跑；Issue 侧仅有一条作者自己的 /ci run 与 github-actions[bot] 的 Buildkite CI #88471 触发回执。

作者在 body 中主动标注了验证边界，值得作为“已知缺口”阅读：

"AI assistance: OpenAI Codex implemented the change and ran the reported checks and evaluations under the submitter's direction. This PR is a draft; human review of every changed line and human-run validation remain required before marking it ready."

"These are single-run speculative-mode evaluations of the optimized implementation, not before/after accuracy-equivalence measurements."

"Latency means pool all requests across both repetitions; throughput is total output tokens / total measured duration." (并注明 concurrency-128 差异在波动范围内、adaptive 运行含一个 2.142 ms 离群点、长 prompt 输出在 baseline 重复间也会波动, "universal generated-token parity is not claimed")

结论: 变更的“共识”并非来自评审博弈, 而是作者提供的自测矩阵 (30 个 attention 测试、24 个 metadata-builder 用例 $rtol=atol=0$ 、四卡实时元数据对比、GSM8K 单次评测), 最终由作者本人 self-merge, 缺少独立复核这一环。

- 自动化 review 因 fork 被跳过, 人工 review 缺位 (other): 该 PR 在缺少独立人工复核的情况下合入, 正确性依据完全来自作者自测矩阵 (30 个 attention 测试、24 个 metadata-builder 用例 $rtol=atol=0$ 、四卡实时对比)。这是流程性缺口而非技术结论。
- 性能数据与仓库内可复现性 (performance): 30 个 attention 测试、24 个 metadata-builder 用例以 $rtol=atol=0$ 对齐基线, 且四卡实时元数据比对通过, 功能正确性有较强证据; 但 3-4x 元数据加速与 15-18% TPOT 收益没有 CI 级回归护栏, benchmark 工具未入库, 后续重构无法被自动发现退化。
- CI 触发与验证状态 (testing): CI 侧未见异常, 但没有针对新内核本身的 CI 门禁, 新增测试依赖 CUDA 环境按 skipif 跳过时可能静默不生效。

风险与影响

- 风险: 1) 设备 /CPU 边界语义分叉 (正确性): token_to_req_indices 仍用 `int(self.query_start_loc_cpu[-1])` 作为 num_mapped_tokens, 新内核把 `[num_mapped_tokens, num_output_tokens)` 统一写 0; 而旧实现的 `torch.repeat_interleave(..., output_size=num_mapped_tokens)` 在总量不足时会循环补齐、过量时截断。PR 注释只保证“CPU 总量正确、逐请求边界可能过期”, `test_device_token_request_mapping` 覆盖的也正是“分布变化而总量一致”的场景; 若 CPU 总量本身过期, 新旧行为会产生分叉且现有测试不覆盖。2) indexer.py 中的 CPU 依据仍在: `max_decode_len / min_decode_len / write_is_uniform / use_native` 仍来自 `torch.diff(query_start_loc_cpu)`, 只有映射改由设备边界决定。DSpark 自适应若真的改变设备侧 token 分布, 设备元数据与 PyTorch 侧形状 /legality 判定之间的一致性需要复核。3) 持久 buffer 语义变更 (回归面): `decode_lens_buffer` 在 `varlen` 下由“逐请求”变为“逐 token”, `decode_indices_buffer`、`decode_seq_lens_buffer`、`expanded_block_table_buffer` 被内核原地改写, 且 `requires_padding` 在 `varlen` 下硬编码为 `False`。任何下游消费者若仍按旧的逐请求 `decode_lens` 语义或 `requires_padding=True` 分支行事, 会出现静默错误; 片段中未出现这些 buffer 的全部消费者, 无法完全确认覆盖。4) 内核内数据依赖循环 (性能 / 时延抖动): `_token_request` 的 while 循环次数取决于 `query_start_loc` 分布 (最坏约 $\log_2(\text{num_reqs})$ 轮), SIMT 下以“全部 lane 收敛”为退出条件, 极端偏斜的批次会拉长最慢 program 的耗时; 同时每次 launch 都要按 `cdiv(capacity - num_decode_tokens, 256)` 清零容量尾部, 工作量与容量 (`max_num_batched_tokens`) 而非实际 batch 成比例, 小 batch 大容量时这部分开销固定存在。5) 验证强度不足 (流程风险): 无人工 review 评论、自动化 review 因 fork 被禁用、作者 self-merge; GSM8K 与 SPEED-Bench 均为单次 / 单侧测量, 作者自述 “not directly comparable”“parity is not claimed”; benchmark harness 在第二个 commit 被

移除，仓库内无性能回归护栏，后续对 `_indexer_decode_metadata_kernel` 或 `buffer` 布局的任何改动都没有自动化保护。

- 影响：用户侧：DeepSeek-V4.1 在 DSpark 投机解码场景的 TPOT 明显受益（batch-1: 1.657 → 1.407 ms, -15.07%；加自适应 1.781 → 1.457 ms, -18.20%；非 DSpark 也降 5.82%），SPEED-Bench 8K/1K 下 concurrency=1 的 TPOT 降 19.31%、吞吐 1.223x，高并发（128/1024）收益收敛到 0.2%-1.4%，TTFT 基本持平甚至出现 -0.48% 的噪声级波动。系统侧：CommonAttentionMetadata.token_to_req_indices 是所有 attention backend 共用的映射入口，语义变化（尾部写 0、设备端二分）影响面大于 DSV4.1 本身；元数据路径不再有 `aten::repeat_interleave`，解码步的 GPU kernel launch 数从 77 降到 17，对 CPU 侧 launch-bound 场景尤其友好。团队侧：这是“把 CPU 可见的控制流下沉到设备端”这一范式在 vLLM 元数据路径上的又一次落地，会给后续所有依赖 `query_start_loc_cpu` 的元数据构造带来范式压力；同时本 PR 几乎零人工 review 即合入，短期内其他贡献者若照此模式改元数据，缺少可参考的评审标准。
- 风险标记：核心路径变更，跨 backend 共用的元数据语义变更，设备 /CPU 边界假设，CUDA graph 持久 buffer 原地改写，`requires_padding` 语义硬编码变更，缺少独立人工 review，性能收益无 CI 回归护栏，AI 生成实现 + 自测验证

关联脉络

- PR #56554 [DSV4.1] Remove compressor-aware image sentinel token padding: 同属 DeepSeek-V4.1 输入准备路径的改动（`mm_preprocess / vl_model / config`），与本 PR 一样在修正 DSV4.1 输入侧的 token 流与元数据语义，是同一模型线的相邻工作。
- PR #53566 [5/N][warmup][DSV4] Migrate NVIDIA CuTeDSL attention kernels: 该系列建立了共享 JIT 预热契约，而本 PR 明确声明 "No new JIT warmup-framework wrappers or registrations are added"，即刻意绕开预热框架直接在调用点启动 Triton，属于对同一契约的边界性选择。
- PR #56503 [ROCm][DSV4.1][Perf] Use AITER mHC for the delayed pre block: 同样是 DSV4.1 的元数据 / 内核级性能优化，只是走 ROCm + AITER 路线；两者共同体现 DSV4.1 输入与 mHC 路径正在被逐段下沉到内核层。
- PR #53781 [3/N] HiSparse: host-resident sparse-MLA decode hot-buffering: 涉及 MLA/indexer 侧 KV 与元数据路径（`single_type_kv_cache_manager`、`sparse_mla_attention`），与本 PR 改写的 `indexer decode` 元数据同属 SM100 稀疏 MLA 栈，消费同一批 buffer 语义。
- PR #56398 （PR body 提及的 #56344，标题未在上下文中提供：融合 attention 数学计算）：作者在 PR body 中列为相关工作并声明职责不同：该 PR 融合 attention 计算，本 PR 只替换输入元数据构造。
- PR #55692 （PR body 提及的 #55692，标题未在上下文中提供：加固下游 top-k block table gather）：作者在去重检查中明确区分：该 PR 加固下游 top-k block-table gather，本 PR 只负责生成上游元数据，二者的 block table 语义需要保持对齐。