PR #50294 完整报告

vllm-project/vllm

[Kernel][Model] Optimize FA4 mm_prefix range lookup

合并时间：2026-08-05 22:33

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50294

执行摘要

- 一句话：FA4 mm_prefix 范围扫描改 O(1) 查表，QPS 提升约 93%
- 推荐动作：值得精读。三个设计决策有很强的迁移价值：

1. 把内核内扫描转化为元数据预计算，并利用『ranges 不重叠』不变式把双向判断简化为只查 query 侧边界，彻底消除 key 侧查找——这是让查找变为 O(1) 的关键，而非简单缓存。
2. functools.cache + 固定函数对象规避 FA4 hash_callable 编译 key 抖动，对 vllm 自研 FA4/CuTe-DSL 内核生态（如 PR #49792 方向）具有直接参考意义。
3. 热路径零分配约束下的持久缓冲设计（pinned CPU + GPU 双缓冲、seq_lens_cpu_upper_bound 的乐观上界技巧），是 CUDA graph 捕获期内存管理的优秀范例。

建议阅读顺序：PR body 的性能数据与问题分析 → fill_mm_prefix_query_ranges → _make_mm_prefix_mask_mod → 测试文件中的 dense float32 参考实现。

功能与动机

PR body 明确指出两个动因：一是 max_ranges 通过 cutlass.range_constexpr(max_ranges) 被固化进 CuTe JIT 编译 key，视频帧数变化（1/8/16/32/46/64）即触发重复冷编译，『The GPU can sit idle while CuTe compiles a new specialization』；二是即使已编译，掩码工作量仍为 O(max_ranges) per score，并随 batch 最大帧数放大。Gemma4 没有原生视频塔，每个视频帧被拆成一个双向 mm_prefix 范围（每帧对应一个 <|video|> 软 token 区间），因此视频负载天然产生 1~64 个范围。此外 FA4 是 Gemma4 统一 sliding（head_dim=256）与 global（512）层的唯一后端，mm_prefix 路径必须走 FA4，无法绕开该热点。

实现拆解

整个优化把『内核内扫描』转化为『步进前元数据预计算 + O(1) 查表』，分五步落地：

1. 元数据预计算（vllm/v1/attention/backends/utils.py）：新增 fill_mm_prefix_query_ranges，把 CommonAttentionMetadata.mm_req_doc_ranges 的请求级范围列表转换为按已调度 query token 行布局的 (num_actual_tokens, 2) int32 表。范围外填 (-1, -1)，退化范围（start >= end）跳过以对齐 Triton 路径的 start < end 语义，chunked prefill 下超出当前 chunk 的部分直接裁剪。函数返回写入行数，0 表示无范围覆盖，调用方据此跳过整个 mask_mod（纯文本与 decode batch 零开销）。内存上界因此从 num_seqs * max_seq_len 降至 max_num_batched_tokens，这是 MatthewBonanni

在 review 中否定了 '32 bits * num_seqs * max_seq_len 太大 ' 的 per-token range-id 张量方案后确定的形状。

2. 元数据契约变更（vllm/v1/attention/backend.py、flash_attn.py）：FlashAttentionMetadata.mm_prefix_range_tensor（(num_seqs, max_ranges, 2)）改名为 mm_prefix_query_range_tensor（(num_actual_tokens, 2)）；CommonAttentionMetadata.mm_req_doc_ranges 文档补充『一个请求内的 ranges 不得重叠』不变式。该不变式是 O(1) 等价性（query 与 key 同范围 ⟺ key 落在 query 自己范围内）的前提，也是内核可以完全省略 key 侧查找的依据。
3. 持久缓冲与热路径零分配（flash_attn.py builder）：FlashAttentionMetadataBuilder.__init__ 在 is_mm_prefix_lm 时按 max_num_batched_tokens 预分配 pinned CPU 暂存 + GPU 持久缓冲；build() 用 query_start_loc_cpu 与 seq_lens_cpu_upper_bound 填充，再 non_blocking 拷贝到 GPU 前段切片并挂到 metadata 上。decode 行的乐观上界会把 query 位置推离所有范围，保证『decode query 必在范围外』的不变式仍然成立；要求 seq_lens_cpu_upper_bound 而非已废弃的 seq_lens_cpu，也避免了旧字段隐式 D2H 拷贝在每次 build 上的同步开销。
4. mask_mod 重写与 JIT 缓存（flash_attn.py）：_make_mm_prefix_mask_mod 删除 max_ranges 参数并加 @functools.cache——FA4 的 hash_callable 会把闭包 cell 的 repr() 混入编译 key，不缓存则每次 forward 都生成新函数地址并触发完整 CuTe 重编译。新增 @cute.jit 的 _load_q_range：从 aux tensor（mm_prefix_query_range_tensor）加载当前 query 行的 [start, end]，行号由 cu_seqlens_q[b] + q_local 还原（FA4 传本地 q_idx，kv_idx 是绝对索引）；固定 __vec_size__ = 1 防止向量化错位，并对 seqlen_q == 0 的 padding 行 clamp 索引。配套处理上游 flash-attn #155 的行为变化：附加 mask_mod 时显式 causal = False、sliding_window_size = None，否则 SM90 走内置 causal 路径短路 mask_mod、SM100 裁剪双向块。
5. Gemma4 配套与测试（gemma4_mm.py、tests/v1/attention/test_mm_prefix.py）：gemma4_mm.py 的全注意力层清理逻辑同步置空新字段 mm_prefix_query_range_tensor，保持『双向注意力只作用于 sliding 层』的既有语义。新增 636 行测试：元数据层覆盖带 context offset 的旧 / 新语义等价性、chunked prefill 越界裁剪、持久缓冲复用防泄漏、无范围返回 None；FA4 层测试在 SM100 上把真实内核与 dense float32 参考逐元素对比（含 hd128/512、paged FA4 等形状矩阵）。作者另跑 MMMU 120 样本精度评估并通过合并门槛。

关键文件：

- vllm/v1/attention/backends/flash_attn.py（模块 注意力后端；类别 source；类型 core-logic；符号 _load_q_range, _make_mm_prefix_mask_mod, FlashAttentionMetadata, FlashAttentionMetadataBuilder）：FA4 mm_prefix 热路径的核心改造：FlashAttentionMetadata 字段改名、builder 持久缓冲与 build() 填充、mask_mod 重写为 O(1) 查表并加 functools.cache、显式清除 FA 内置 causal 以适配上游 flash-attn #155。
- vllm/v1/attention/backends/utils.py（模块 注意力工具；类别 source；类型 core-logic；符号 fill_mm_prefix_query_ranges）：新增 fill_mm_prefix_query_ranges：把 mm_req_doc_ranges 的请求级范围列表转换为按 query token 行布局的边界表，是 O(1)

查表方案的元数据基础；同时保留 compute_mm_prefix_range_tensor 供 Triton 路径继续使用。

- tests/v1/attention/test_mm_prefix.py（模块 前缀掩码；类别 test；类型 test-coverage；符号 _fa4_available, _query_ranges, test_matches_range_scan_semantics_with_context_offset, test_ranges_beyond_scheduled_chunk_are_clipped）：新增 636 行测试，是本次变更正确性的核心保障：元数据层锚定 O(1) 查表与旧扫描的语义等价性、chunked prefill 裁剪、缓冲复用防泄漏；FA4 层以 dense float32 参考在 SM100 上验证真实内核。
- vllm/model_executor/models/gemma4_mm.py（模块 模型实现；类别 source；类型 data-contract）：Gemma4 全注意力层清理逻辑同步置空新字段 mm_prefix_query_range_tensor，保证『双向注意力只作用于 sliding 层』的既有语义不被新字段破坏。
- vllm/v1/attention/backend.py（模块 注意力元数据；类别 source；类型 data-contract）：CommonAttentionMetadata.mm_req_doc_ranges 文档补充『一个请求内的 ranges 不得重叠』不变式，这是 O(1) 查表语义正确性的前提契约。

关键符号：fill_mm_prefix_query_ranges, _make_mm_prefix_mask_mod, _load_q_range, FlashAttentionMetadataBuilder.build

关键源码片段

vllm/v1/attention/backends/flash_attn.py

FA4 mm_prefix 热路径的核心改造：FlashAttentionMetadata 字段改名、builder 持久缓冲与 build() 填充、mask_mod 重写为 O(1) 查表并加 functools.cache、显式清除 FA 内置 causal 以适配上游 flash-attn #155。

```
# vllm/v1/attention/backends/flash_attn.py —— FA4 mask_mod 的 O(1) 范围查找
@functools.cache
def _make_mm_prefix_mask_mod(
    sliding_window: int = 0,
    sliding_window_left: int | None = None,
):
    """构造 CuTe-DSL mask_mod，实现 ``(causal AND sliding_window) OR mm_prefix``。

    ``functools.cache`` 缓存函数对象本身：FA4 的 ``hash_callable`` 会把闭包
    cell 的 ``repr()`` 混入编译 key，不缓存则每次 forward 都生成新的嵌套
    函数地址，触发一次完整 CuTe JIT 重编译（旧实现还因 ``max_ranges`` 参与
    编译 key 而随视频帧数波动反复冷编译）。
    """
    from vllm.vllm_flash_attn.cute.utils import scalar_to_ssa, ssa_to_scalar

    @cute.jit
    def _load_q_range(q_idx, seqlen_info, aux_tensors, batch_idx):
        # aux_tensors[0] = (num_actual_tokens, 2) 的绝对 [start, end] 表；
        # 行号 = cu_seqlens_q[batch] + 本地 q_idx，与 FA4 的 varlen 打包一致。
        # 两个加载只依赖 query 索引，会被编译器提升出逐元素掩码循环。
        # 固定向量宽度 1：本函数只读 q_idx 的 lane 0，向量化调用会把同一行
        # 边界错误地套用到整个向量上。
```

```
        token_idx = aux_tensors[1][batch_idx] + q_idx
        # seqlen_q == 0 的 padded 批成员，其 cu_seqlens_q 已越过表尾，
        # clamp 保证不越界（这类行不产生输出，行为不变）。
        token_idx = min(token_idx, aux_tensors[0].shape[0] - 1)
        r_start = aux_tensors[0][token_idx, 0]
        r_end = aux_tensors[0][token_idx, 1]
        return r_start, r_end

    def mask_mod(q_idx, kv_idx, seqlen_info, aux_tensors, batch_idx):
        # FA4 传入的 q_idx 是当前 chunk 内本地索引，kv_idx 是相对完整 KV
        # cache 的绝对索引；用 cu_seqlens_q 还原绝对 query 位置，与 Triton
        # 参考路径（compute_kv_seq_mask）保持一致。
        q_abs = aux_tensors[1][batch_idx] + q_idx
        delta = q_abs - kv_idx

        keep = delta >= 0 # causal 侧
        if sliding_window_left is not None:
            keep &= delta < sliding_window_left

        # mm_prefix 侧：key 落在 query 自己范围的 [start, end] 内即可双向
        # 可见。ranges 不重叠，故无需 key 侧查表；(-1, -1) 哨兵自动失效，
        # 因为 kv_idx <= -1 对所有合法 key 恒为假。
        r_start, r_end = _load_q_range(q_idx, seqlen_info, aux_tensors, batch_idx)
        mm = (r_start <= kv_idx) & (kv_idx <= r_end)
        if sliding_window > 0:
            # Gemma4 局部层把双向块钳制进窗口：只约束过去侧，
            # 范围内未来的 key 仍然可见（mm_prefix_clamp_sliding_window）。
            mm &= delta < sliding_window
        return keep | mm

    return mask_mod

# forward() 中附加 mask_mod 的接线：mm_prefix 掩码不是 causal 的子集，
# FA #155 之后上游不再在设置 mask_mod 时自动清除 causal/local —— 必须显式
# 关闭，否则 SM90 走内置 causal 路径短路 mask_mod，SM100 则裁剪双向块。
if mm_prefix_query_ranges is not None and not is_dynamic_causal and causal is True \
        and self.vllm_flash_attn_version == 4:
    layer_window = self.sliding_window # 用 impl 的逐层窗口，而非 model-wide 值
    sw_val = 1 + layer_window[0] if layer_window is not None and layer_window[0] >= 0 \
        else None
    mm_mask_mod = _make_mm_prefix_mask_mod(
        sliding_window=sw_val, sliding_window_left=sw_val,
    )
    mm_aux = [mm_prefix_query_ranges, attn_metadata.query_start_loc]
    causal = False
    sliding_window_size = None
```

vllm/v1/attention/backends/utils.py

新增 fill_mm_prefix_query_ranges：把 mm_req_doc_ranges 的请求级范围列表转换为按 query token 行布局的边界表，是 O(1) 查表方案的元数据基础；同时保留 compute_mm_prefix_range_tensor 供 Triton 路径继续使用。

```
# vllm/v1/attention/backends/utils.py —— mm_prefix 范围元数据预计算
def fill_mm_prefix_query_ranges(
    out: np.ndarray,
    mm_prefix_range: dict[int, list[tuple[int, int]]] | None,
    query_start_loc_cpu: torch.Tensor,
    seq_lens_cpu: torch.Tensor,
) -> int:
    """把每个已调度 query token 映射到包含它的 mm_prefix 范围，写入调用方持有的
    ``(max_num_batched_tokens, 2)`` int32 暂存缓冲，返回实际写入行数。

    第 ``i`` 行保存 query token ``i`` 所属双向范围的绝对 ``[start, end]``，
    范围外为 ``(-1, -1)``。返回 0 表示没有任何范围覆盖已调度 query token，
    调用方应直接跳过 mask_mod（纯文本 / decode batch 零开销）。
    """
    if mm_prefix_range is None:
        return 0

    query_start_loc = query_start_loc_cpu.numpy()
    num_actual_tokens = int(query_start_loc[-1])
    if num_actual_tokens <= 0:
        return 0
    assert num_actual_tokens <= out.shape[0], (
        f"mm_prefix staging buffer holds {out.shape[0]} tokens, got {num_actual_tokens}"
    )

    # 先解析全部 span 再写 out：chunked prefill 下范围可能整体落在已调度
    # token 之外，此时跳过填充即可，不必碰缓冲。
    spans: list[tuple[int, int, int, int]] = []
    for req_idx, req_ranges in mm_prefix_range.items():
        if not req_ranges:
            continue
        token_start = int(query_start_loc[req_idx])
        query_len = int(query_start_loc[req_idx + 1]) - token_start
        if query_len <= 0:
            continue
        # 该请求第一个已调度 query token 的绝对位置
        context_len = int(seq_lens_cpu[req_idx]) - query_len
        for start, end in req_ranges:
            if start >= end:
                continue # 与 Triton 路径 start < end 的有效性检查对齐
            first = max(start - context_len, 0)
            last = min(end - context_len, query_len - 1)
            if first > last:
                continue
            spans.append((token_start + first, token_start + last + 1, start, end))
```

```
    if not spans:
        return 0

    # 整段先填 -1 再覆盖范围行：持久缓冲跨 step 复用，必须保证上一 step 的
    # 旧边界不会残留在本次范围之外的行上，否则双向掩码会被静默放大。
    out[:num_actual_tokens] = -1
    for row_start, row_end, start, end in spans:
        out[row_start:row_end] = (start, end)
    return num_actual_tokens
```

tests/v1/attention/test_mm_prefix.py

新增 636 行测试，是本次变更正确性的核心保障：元数据层锚定 O(1) 查表与旧扫描的语义等价性、chunked prefill 裁剪、缓冲复用防泄漏；FA4 层以 dense float32 参考在 SM100 上验证真实内核。

```
# tests/v1/attention/test_mm_prefix.py —— 元数据语义锚定
def _query_ranges(mm_ranges, query_lens, seq_lens):
    """填充暂存缓冲并返回已写行；无范围覆盖时返回 None。"""
    query_start_loc = torch.tensor(
        [0, *torch.tensor(query_lens).cumsum(0).tolist()], dtype=torch.int32
    )
    # 用毒值初始化缓冲，让任何残留旧行在断言中显形
    out = np.full((STAGING_CAPACITY, 2), 12345, dtype=np.int32)
    num_tokens = fill_mm_prefix_query_ranges(
        out, mm_ranges, query_start_loc, torch.tensor(seq_lens, dtype=torch.int32)
    )
    if num_tokens == 0:
        return None
    return torch.from_numpy(out[:num_tokens])


def test_matches_range_scan_semantics_with_context_offset():
    """钉死 O(1) 查表所依赖的等价性：r_start <= kv_idx <= r_end 等价于旧扫描
    的 any(q in r and kv in r)——因为 ranges 永不重叠。请求 1 带 context
    offset，顺带覆盖本地到绝对 query 位置的换算。
    """
    mm_ranges = {0: [(1, 3), (5, 7)], 1: [(2, 4), (9, 12)]}
    query_lens = [8, 6]
    seq_lens = [8, 13]

    query_ranges = _query_ranges(mm_ranges, query_lens, seq_lens)
    assert query_ranges is not None

    token_start = 0
    for req_idx, query_len in enumerate(query_lens):
        context_len = seq_lens[req_idx] - query_len
        for q_local in range(query_len):
            q_abs = context_len + q_local
            r_start, r_end = query_ranges[token_start + q_local].tolist()
```

```
            for kv_idx in range(seq_lens[req_idx]):
                old_scan_keep = any(
                    start < end and start <= q_abs <= end and start <= kv_idx <= end
                    for start, end in mm_ranges[req_idx]
                )
                new_keep = r_start <= kv_idx <= r_end
                assert new_keep == old_scan_keep, (req_idx, q_abs, kv_idx)
        token_start += query_len
```

评论区精华

核心讨论围绕『收益来源』与『元数据形状』展开：

- MatthewBonanni 最初评论：『Since the majority of the speedup is coming from eliminating the JIT on the hot path, I'd prefer to just stick to that unless we can come up with a better way of fixing the range scan』——倾向只保留 JIT 缓存收益。随后他亲自在 fork PR #2 中提出更优方案（query-token 行布局）并合入本 PR，两个优化点最终都保留。
- 对最初 (num_seqs, max_seq_len) range-id 张量方案的否决：『We can't allocate on the hot path. Preallocating this isn't an option because 32 bits * max_num_seqs * max_seq_len is too large』——长上下文 Gemma4 下可达数 GiB，最终改为受 max_num_batched_tokens 约束的 (num_actual_tokens, 2) 持久缓冲。
- 上游 FA pin 升级（flash-attn #155）引发的 CI 失败是本 PR 最终形态的重要转折：mask_mod 存在时不再自动清除内置 causal/local，SM90 短路 mask_mod、SM100 裁剪双向块，CI 出现『Tensor-likes are not close』。修复方式是显式 causal=False，该经验被固化到测试文件头部注释。
- 合入门槛：Approval 附条件『Please add an lm_eval result though, I'll wait to merge until that's verified』，作者补充 MMMU 120 样本评估（FA4 before 0.6000 / after 0.6083 / Triton 0.6167，新旧 FA4 bit-exact）后合入。
- 热路径分配约束与 per-token 元数据形状设计 (design)：采纳 query-token 行布局方案，由 MatthewBonanni 提交核心 commit 并合入本 PR；原 (num_seqs, max_seq_len) range-id 设计被丢弃。
- 主要收益来自消除 JIT，是否值得保留范围扫描优化 (performance)：两个优化点都保留：functools.cache 固定 mask_mod 函数对象，per-token 查表去掉 max_ranges 循环与编译 key 抖动。
- FA #155 后 mask_mod 与内置 causal 的交互 (correctness)：附加 mm_prefix mask_mod 时显式 causal=False、sliding_window_size=None；测试直接调用内核处同样处理。对应 commit『Disable FA causal when attaching mm_prefix mask_mod』。
- 合并门槛：lm_eval 精度验证 (testing)：精度达标后合入。
- _load_q_range 向量化与 padding 行索引安全 (correctness)：固定向量宽度 1 并 clamp 索引，行为对每个产生输出的行不变。

风险与影响

- 风险：风险集中在以下四点：

1. 上游 FA pin 耦合（flash_attn.py）：正确性依赖 flash-attn #155 之后『不再自动清除 causal/local』的行为。若 FA pin 回退或行为再次变化，SM90 会静默退化回纯 causal 掩码、SM100 裁剪双向块，且不会显式报错——CI 曾真实触发过该问题。
2. 持久缓冲容量假设（flash_attn.py / utils.py）：fill_mm_prefix_query_ranges 的缓冲按 max_num_batched_tokens 预分配，若调度配置变更导致单步 num_actual_tokens 超限，会以 assert 直接崩溃（fail-fast，而非静默越界）；num_mm_tokens == 0 时 metadata 字段保持 None，逻辑依赖该分支的正确实现。
3. decode 行不变式（flash_attn.py）：填充使用 seq_lens_cpu_upper_bound，decode 行是乐观上界，支撑『decode query 必在范围外』的假设。若未来 mm_prefix 范围扩展到生成 token，该假设将失效并产生错误掩码。
4. 平台覆盖盲区（测试）：真实内核测试需要 FA4 + SM100 才能执行，其余 CI 平台只能跑元数据层测试，GPU 侧的索引计算（q_ranges[token_idx, 0] 运行时 Int32 索引、cu_seqlens_q[b] + q_local 打包）缺少跨平台回归保障。

影响面有限：仅 FA4 + mm_prefix + prefill 路径受影响；纯文本、decode、非 mm_prefix 模型以及 Triton 注意力路径均不受影响（提前返回 None）。

- 影响：对用户：Gemma4 视频 / 多模态负载在 FA4 后端下的 prefill 吞吐显著提升（pooling 场景 QPS +92.8%），冷启动反复 JIT 编译被消除，FA4 路径反超 Triton 参考路径；正确性经 MMMU 与 bit-exact 对比验证。对系统：v1 attention 元数据契约发生变化（mm_prefix_range_tensor 改名为 mm_prefix_query_range_tensor 且语义从『请求级范围表』变为『query token 级边界表』），任何引用旧字段的 backend 或模型代码都需要同步迁移；Triton 路径与 compute_mm_prefix_range_tensor（(batch, max_ranges, 2)）保持不变，形成两套并行元数据。对团队：该 PR 是 Gemma4 在 v1 下正确性与性能收尾的一部分，也为后续 FA4/CuTe 内核的 JIT 稳定性提供了可复用的工程模式（functools.cache 固定函数对象、持久 pinned 缓冲、显式清除内置掩码）。
- 风险标记：核心路径变更（FA4 mask_mod），依赖上游 FA pin 行为（flash-attn #155），注意力元数据字段契约变更，测试依赖 FA4/SM100 硬件，预分配缓冲绑定 max_num_batched_tokens

关联脉络

- PR #50940 [R3] Unify routed expert shape configuration: 同属 Gemma4 模型族在 v1 下的修复 / 重构批次，且同为模型配置与执行路径的契约调整，与本 PR 的注意力元数据契约变更相互印证。
- PR #50958 [Bugfix][Model] Gemma3n/Gemma4: pad variable-length audio batches: 同为 Gemma4 多模态正确性修复（变长批处理边界），与本 PR 的 mm_prefix 元数据边界处理（chunked prefill 裁剪、decode 行上界）属于同一类边界问题。
- PR #49792 [Kernel][SM100] Add a CuTeDSL fused query kernel: 同在 vllm 自研 FA4/CuTe-DSL 内核方向（flash-attn fork），本 PR 对 hash_callable 编译 key 的分析与 functools.cache 方案对该内核生态的 JIT 稳定性有直接参考价值。