PR #46315 完整报告

vllm-project/vllm

[Bugfix][Spec Decode] Fix EAGLE drafter multimodal encoder cache misses

合并时间：2026-06-23 02:14

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46315

执行摘要

- 一句话：修复 EAGLE 超前查看导致的编码器缓存缺失
- 推荐动作：该 PR 虽然以 bugfix 为目标，但涉及调度器与模型运行器之间关于编码器缓存生命周期的协作设计，值得阅读。特别是 _free_encoder_inputs 与 gather_mm_embeddings 的对称调整（一个延迟释放，一个容忍边界缺失），展示了如何通过两个切入点共同解决一致性问题。

功能与动机

EAGLE/MTP drafter gathers multimodal encoder embeddings one position ahead of the target model's processed range, but the encoder cache's schedule / free / evict accounting only tracks the target. The drafter's +1 look-ahead can therefore reference a not-yet-encoded next-chunk feature or a prematurely reclaimed entry, raising "Encoder cache miss" and killing EngineCore.

实现拆解

- 在条件中增加 spec_lookahead = 1 if self.use_eagle else 0，使得 start_pos + num_tokens 必须多跨越一个 token 才能释放，确保 drafter 的 +1 超前仍有缓存。
- 当 draft_lookahead > 0 时，偏移 computed_prefill_lens，将特征选择窗口锁定在未偏移的已处理边界内。
- 当 encoder_output 为 None 时，若特征起始位置恰好等于或超过当前查询结束（边界位置），则跳过该特征（回退到 token embedding）；否则仍抛出 "Encoder cache miss"。
- 当 encoder_output 为 None 时，判断 start_pos 是否 >= num_computed_tokens + num_scheduled_tokens（即边界位置），若是则跳过，否则抛异常。
- tests/v1/worker/test_encoder_runner.py：测试 V2 EncoderRunner 的 draft 场景（缓存命中、边界缺失容忍、内部缺失告警、多请求批量）。
- tests/v1/worker/test_gpu_model_runner_mm_gather.py：测试 V1 的 _gather_mm_embeddings 同样场景。
- tests/v1/core/test_scheduler.py：新增 test_free_encoder_inputs_defers_for_eagle_look ahead 验证调度器延迟释放行为。

关键文件：

- vllm/v1/worker/gpu/mm/encoder_runner.py（模块 编码器；类别 source；类型 core-logic；符号 gather_mm_embeddings）：核心修复文件之一：修改 gather_mm_embeddings 方法，增加 draft_lookahead 参数，调整查询范围并容忍边界缺失。
- vllm/v1/core/sched/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _free_encoder_inputs）：核心修复文件之一：修改 _free_encoder_inputs 方法，在 use_eagle 时增加 1 个 token 的额外裕度，延迟释放编码器缓存。
- tests/v1/worker/test_encoder_runner.py（模块 编码器测试；类别 test；类型 test-coverage；符号 _feature, _make_runner, _gather, test_draft_lookahead_uses_boundary_feature_when_cached）：新增的 V2 EncoderRunner gather 测试，覆盖 draft lookahead 下的缓存命中、边界缺失容忍及内部缺失抛出。
- tests/v1/worker/test_gpu_model_runner_mm_gather.py（模块 Gather 测试；类别 test；类型 test-coverage；符号 _feature, _gather, test_draft_shift_uses_boundary_feature_when_cached, test_draft_shift_tolerates_missing_boundary_feature）：新增的 V1 MM Gather 测试，覆盖 draft shift 下的边界特征查找和缺失回退逻辑。
- vllm/v1/worker/gpu_model_runner.py（模块 模型运行器；类别 source；类型 data-contract；符号 _gather_mm_embeddings）：修改 _gather_mm_embeddings，增加对边界缺失的容忍逻辑。
- tests/v1/core/test_scheduler.py（模块 调度器测试；类别 test；类型 test-coverage；符号 test_free_encoder_inputs_defers_for_eagle_lookahead）：新增测试调度器 _free_encoder_inputs 在 use_eagle 时的延迟释放行为。

关键符号：_free_encoder_inputs, gather_mm_embeddings, _gather_mm_embeddings, test_free_encoder_inputs_defers_for_eagle_lookahead, test_draft_shift_uses_boundary_feature_when_cached, test_draft_lookahead_uses_boundary_feature_when_cached, test_draft_shift_tolerates_missing_boundary_feature, test_draft_lookahead_tolerates_missing_boundary_feature, test_draft_shift_raises_on_interior_miss, test_draft_lookahead_raises_on_interior_miss, test_target_path_raises_on_encoder_cache_miss

关键源码片段

vllm/v1/worker/gpu/mm/encoder_runner.py

核心修复文件之一：修改 gather_mm_embeddings 方法，增加 draft_lookahead 参数，调整查询范围并容忍边界缺失。

```
def gather_mm_embeddings(
    self,
    req_ids: list[str],
    total_num_scheduled_tokens: int,
    num_scheduled_tokens: np.ndarray,
    query_start_loc: np.ndarray,
```

```
    prefill_lens: np.ndarray,
    computed_prefill_lens: np.ndarray,
    draft_lookahead: int = 0,
) -> tuple[list[torch.Tensor], torch.Tensor]:
    # 当 `draft_lookahead > 0` 时，将所有偏移量加上 draft_lookahead，
    # 模拟 drafter 的 +1 位置，从而将特征选择窗口锁定在未偏移的已处理边界内。
    if draft_lookahead:
        computed_prefill_lens = computed_prefill_lens + draft_lookahead

    is_prefilling_np = computed_prefill_lens < prefill_lens
    if not is_prefilling_np.any():
        # 所有请求都处于 decode 阶段，无需收集 embeddings。
        return [], torch.zeros(
            total_num_scheduled_tokens, dtype=torch.bool, device=self.device
        )

    is_prefilling = is_prefilling_np.tolist()
    query_start = computed_prefill_lens.tolist()
    query_end = (computed_prefill_lens + num_scheduled_tokens).tolist()

    mm_embeds: list[torch.Tensor] = []
    is_mm_embed = torch.zeros(
        total_num_scheduled_tokens, dtype=torch.bool, device="cpu"
    )
    for i, req_id in enumerate(req_ids):
        if not is_prefilling[i]:
            # 跳过 decode 请求（优化）。
            continue

        cur_query_start = query_start[i]
        cur_query_end = query_end[i]

        mm_features = self.encoder_cache.mm_features[req_id]
        lo, hi = get_mm_features_in_window(
            mm_features, start=cur_query_start, end=cur_query_end
        )
        for idx in range(lo, hi):
            mm_feature = mm_features[idx]
            pos_info = mm_feature.mm_position
            start_pos = pos_info.offset
            num_encoder_tokens = pos_info.length

            start_idx = max(cur_query_start - start_pos, 0)
            end_idx = min(cur_query_end - start_pos, num_encoder_tokens)
            assert start_idx < end_idx
            curr_embeds_start, curr_embeds_end = (
                pos_info.get_embeds_indices_in_range(start_idx, end_idx)
            )
            if curr_embeds_start == curr_embeds_end:
```

```
                # 未覆盖任何 embedding 切片，跳过。
                continue

            mm_hash = mm_feature.identifier
            encoder_output = self.encoder_cache.encoder_outputs.get(mm_hash, None)
            if encoder_output is None:
                # 特征起始位置等于或超过当前查询结束位置时，说明它位于
                # 已处理边界之外，只可能由 drafter 的 +1 超前访问。
                # 此时尚不确定是否已编码，为了健壮性回退到 token embedding，
                # 避免抛出"Encoder cache miss"异常。
                if start_pos + draft_lookahead >= cur_query_end:
                    continue
                # 如果特征在已处理范围内仍然缺失，说明真•缓存缺失，必须告警。
                raise RuntimeError(f"Encoder cache miss for {mm_hash}.")

            if (is_embed := pos_info.is_embed) is not None:
                is_embed = is_embed[start_idx:end_idx]
                mm_embeds_item = encoder_output[curr_embeds_start:curr_embeds_end]
            else:
                mm_embeds_item = encoder_output[start_idx:end_idx]

            req_start_pos = query_start_loc[i] + start_pos - cur_query_start
            is_mm_embed[req_start_pos + start_idx : req_start_pos + end_idx] |= (
                True if is_embed is None else is_embed
            )
            mm_embeds.append(mm_embeds_item)

    return mm_embeds, is_mm_embed
```

vllm/v1/core/sched/scheduler.py

核心修复文件之一：修改 _free_encoder_inputs 方法，在 use_eagle 时增加 1 个 token 的额外裕度，延迟释放编码器缓存。

```
def _free_encoder_inputs(self, request: Request) -> None:
    cached_encoder_input_ids = self.encoder_cache_manager.get_cached_input_ids(
        request
    )
    # 优化：为空则提前返回。
    if not cached_encoder_input_ids:
        return

    # 延迟释放裕度：EAGLE 模式下 drafter 需要 +1 步的 look-ahead，
    # 因此编码器缓存必须多保持一个 token 位置，避免过早回收。
    spec_lookahead = 1 if self.use_eagle else 0

    # 使用 list(set) 避免在迭代中修改集合。
    for input_id in list(cached_encoder_input_ids):
        mm_feature = request.mm_features[input_id]
        start_pos = mm_feature.mm_position.offset
```

```
num_tokens = mm_feature.mm_position.length

if self.is_encoder_decoder and request.num_computed_tokens > 0:
    # Whisper 模型：一旦生成一个 token，编码器输入即可释放。
    self.encoder_cache_manager.free_encoder_input(request, input_id)
elif (
    start_pos + num_tokens + spec_lookahead
    <= request.num_computed_tokens - request.num_output_placeholders
):
    # 已处理完毕并存入 decoder KV 缓存，且进度已安全地超出
    # placeholder 范围（加上 drafter look-ahead），后续不再需要。
    self.encoder_cache_manager.free_encoder_input(request, input_id)
```

评论区精华

- 提问者：ywang96
- 内容：询问 _gather_mm_embeddings 中 feature_window_end 相关改动的必要性。
- 回答：njhill 解释 num_computed_tokens 在 eagle 场景下已被偏移，直接使用会漏掉边界特征；但后续决定不强制 clamp，而是用容忍缺失的 fallback 替代。
- 结论：修改方案被接受，reviewer 表示赞同。
- 提问者：TheEpicDolphin
- 内容：质疑 scheduler 中增加 spec_lookahead 的必要性，认为 draft 模型超前于 base 模型，可能不需要多保留一个 token。
- 回答：njhill 解释因为 base 模型会早一个 token 完成，所以 draft 模型仍可能需要该特征，确认延迟释放是必要的。
- 结论：理解并认同该设计。
  - 为什么需要修改 _gather_mm_embeddings 中的 feature_window_end？ (question): 解释后决定调整方案，使用容忍缺失的方式处理边界特征。
  - 调度器 spec_lookahead 是否必要？ (design): 理解并认同该设计。

风险与影响

- 风险：#### 边界条件依赖
- 调度器延迟释放仅当 use_eagle == True 时生效；若未来有其他 spec decode 方案有类似偏移但未正确设置标志，可能再次出现缓存缺失。
- encoder_runner.py 中 if start_pos + draft_lookahead >= cur_query_end 的判断依赖准确计算，若 draft_lookahead 或 cur_query_end 计算有误，可能导致错误跳过有效特征或仍抛出异常。
- 多保留一个 token 的特征可能会略微增加编码器缓存压力，尤其在长序列或大量多模态输入时，但影响微小（仅 1 token）。
- 新增的三个测试文件覆盖了主要边界场景，但未覆盖多请求共享同一特征、编码器预算等复杂交互。
- 影响：#### 影响范围

- 用户：修复了启用了 EAGLE（或 MTP）spec decode 且使用多模态模型时的 EngineCore 崩溃问题，这些用户将不再遇到 "Encoder cache miss" 错误。
- 系统：调度器增加了条件分支，gather 函数增加了参数和判断，性能影响可忽略（仅 CPU 逻辑）。
- 团队：为后续维护 spec decode 与多模态的交互提供了明确的代码路径和测试基准。
- 风险标记：边界条件依赖，延迟释放增加内存，仅覆盖 use_eagle 标志

关联脉络

- 暂无明显关联 PR