PR #25910 完整报告

sgl-project/sglang

vit optimization

合并时间：2026-05-22 14:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25910

执行摘要

- 一句话：跨请求 ViT 批处理优化，减少图像编码调用次数
- 推荐动作：该 PR 引入了重要的跨请求 ViT 批处理优化，设计上拆分了原有大函数为可组合工具，值得精读以理解 multimodal 批处理策略。但 AMD CI 失败表明需要在 AMD 平台验证并修复，建议尽快确认问题并跟进。

功能与动机

PR 标题为 'vit optimization'，目的一是减少每个请求独立的 ViT 编码调用，通过跨请求批处理提高 GPU 利用率和吞吐量；二是减少特征在 GPU 与 CPU 之间不必要的拷贝。

实现拆解

1. 新增 _cpu_feature 字段（schedule_batch.py）：在 MultimodalDataItem 中添加 _cpu_feature: Optional[torch.Tensor]，用于在 GPU 编码时保存 CPU 端的原始特征引用。
2. 修改 _move_items_to_device（mm_utils.py）：将特征移至 GPU 时，先将原特征赋值给 item._cpu_feature，使得后续卸载（offload）可以直接使用该 CPU 引用，避免重新从 GPU 复制。
3. 重命名旧函数为 Legacy：将 _get_chunked_embedding_full 更名为 get_chunked_embedding_legacy，清晰标识为旧的全量编码路径。
4. 拆分 _get_chunked_embedding_by_item：将原按物品编码的逻辑拆分为两个可组合函数：find_chunk_items_and_check_cache（查找 chunk 重叠物品并检查缓存）和 assemble_chunk_embedding（从每个物品的嵌入中切片组装 chunk）。拆解后逻辑更清晰，便于复用。
5. 新增 get_chunked_prefill_embedding_legacy：实现跨请求收集缓存未命中物品，统一调用 data_embedding_func 进行 ViT 批处理，然后组装每个请求所需的 embedding chunk。该函数替代了原来的逐请求编码，是本次优化的核心。

关键文件：

- python/sglang/srt/managers/mm_utils.py（模块 Multimodal 工具；类别 source；类型 core-logic；符号 _get_chunked_embedding_full, get_chunked_embedding_legacy, _get_chunked_embedding_by_item, find_chunk_items_and_check_cache）：核心变更文件：重构 multimodal 编码路径，新增跨请求批处理函数，拆分原函数为可组合工具，并修改设备移动逻辑以保留 CPU 引用。

- python/sglang/srt/managers/schedule_batch.py（模块 调度批次；类别 source；类型 core-logic）：新增 _cpu_feature 字段到 MultimodalDataItem，为特征卸载优化提供 CPU 引用。

关键符号：get_chunked_embedding_legacy, find_chunk_items_and_check_cache, assemble_chunk_embedding, get_chunked_prefill_embedding_legacy, _move_items_to_device

关键源码片段

python/sglang/srt/managers/mm_utils.py

核心变更文件：重构 multimodal 编码路径，新增跨请求批处理函数，拆分原函数为可组合工具，并修改设备移动逻辑以保留 CPU 引用。

```
def find_chunk_items_and_check_cache(
    embedding_items_per_req: List[MultimodalDataItem],
    items_offset: List[Tuple[int, int]],
    chunk_start: int,
    chunk_end: int,
) -> List[Tuple[MultimodalDataItem, Optional[torch.Tensor], int, int]]:
    """Return (item, cached_embedding_or_None, start, end) for items in [chunk_start, chunk_
    end)."""
    chunk_entries = []
    for item, (start, end) in zip(embedding_items_per_req, items_offset):
        if end >= chunk_start and start < chunk_end:
            cached = embedding_cache.get_single(item.hash)
            emb = cached.embedding if cached is not None else None
            chunk_entries.append((item, emb, start, end))
    return chunk_entries


def assemble_chunk_embedding(
    chunk_entries: List[Tuple[Any, torch.Tensor, int, int]],
    chunk_start: int,
    chunk_end: int,
) -> Optional[torch.Tensor]:
    """
    Assemble a chunk of embeddings by slicing each item's embedding
    to the portion that falls within [chunk_start, chunk_end).
    """
    chunk_slices = []
    for _, emb, start, end in chunk_entries:
        overlap_start = max(start, chunk_start)
        overlap_end = min(end, chunk_end - 1) # inclusive
        local_start = overlap_start - start
        local_end = overlap_end - start + 1 # exclusive for slicing
        chunk_slices.append(emb[local_start:local_end])

    if not chunk_slices:
```

```
            return None
        return torch.cat(chunk_slices, dim=0)
```

python/sglang/srt/managers/schedule_batch.py

新增 _cpu_feature 字段到 MultimodalDataItem，为特征卸载优化提供 CPU 引用。

```
@dataclasses.dataclass
class MultimodalDataItem:
    modality: Modality
    hash: int = None
    pad_value: int = None
    offsets: Optional[list] = None
    format: MultimodalInputFormat = MultimodalInputFormat.NORMAL
    # the raw features returned by processor, e.g. pixel_values or audio_features
    feature: Union[torch.Tensor, np.ndarray] = None
    # CPU reference kept during GPU encoding, used to skip GPU->CPU copy on offload
    _cpu_feature: Optional[torch.Tensor] = None
    # the precomputed embeddings, passed as final encoder embeddings
    precomputed_embeddings: Optional[Union[torch.Tensor, np.ndarray]] = None
    model_specific_data: dict[str, Any] = dataclasses.field(default_factory=dict)
```

评论区精华

Review 中仅有一个讨论线程：mickqian 询问 mm_utils.py 第 674 行一段代码是否应移除（'remove this?'），作者 yhyang201 回应 'this is intentional; i still need to use it, but it will be removed in a follow-up pr.' 表明该代码在后续 PR 中会清理，当前故意保留。

- 第 674 行代码保留意图 (question): 作者确认当前保留是为了后续使用，将在后续 PR 中清理。

风险与影响

- 风险：
  - 跨请求批处理改变了现有调度逻辑：新的 get_chunked_prefill_embedding_legacy 将多个请求的物品收集到一次调用中，打破了原有的逐请求处理假设，可能与上层的调度器或其他并发处理逻辑不兼容。
  - AMD CI 失败：合并后 PR 在 AMD CI 上引发 3 个 VLM 测试失败（来自 issue 评论），虽然不一定是直接原因，但表明变更在 AMD 平台上可能有兼容性或正确性问题。
  - 缺少测试覆盖：本次提交未包含对应的单元测试或集成测试，跨请求批处理路径的正确性依赖已有测试，可能漏掉边界情况（如无缓存命中、部分命中、不同 modality 混用等）。
- 影响：
  - 用户 / 模型：所有使用 multimodal 输入（图像、视频、音频）的模型都会受益于更高效的 ViT 编码，吞吐量有望提升，尤其是多图像场景。但若存在兼容性问题，可能导致推理结果错误或性能退化。
  - 系统：减少了 GPU kernel 调用次数和 CPU-GPU 数据拷贝，对整体推理延迟和吞吐有正面影响。

- 团队：需要关注 AMD CI 失败并修复；后续应补充测试用例确保跨请求批处理的正确性。
- 风险标记：跨请求批处理改变了现有调度逻辑，AMD CI 失败，缺少测试覆盖

关联脉络

- 暂无明显关联 PR