PR #35463 完整报告

sgl-project/sglang

[VLM] Split Pixtral multi-image features before the CUDA IPC wrap

合并时间：2026-08-22 15:17

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35463

执行摘要

- 一句话：修复 Pixtral 多图请求在 CUDA IPC 下切片代理导致的 HTTP 500
- 推荐动作：值得精读。核心设计是“把 CUDA IPC 包裹固定为 pipeline 最后一步”的生命周期保证，而不是给代理加切片能力；_finalize_mm_items 的 hook 顺序、copy.copy 后重置派生字段、以及 fail-loud 基数校验都是可迁移到其他多模态处理器的模式。建议重点关注 _postprocess_mm_items_before_transport 的可重写约定，以及测试中用 object.__new__ 构造被测对象的技巧。合并后建议观察一段时间多模态请求与 CI 稳定性，确认无周边 processor 回归。

功能与动机

PR body 明确指出：多图请求到 Pixtral 模型（如 mistralai/Mistral-Small-4-119B-2603）在 CUDA IPC feature transport 开启时返回 HTTP 500，报错为 TypeError: 'CudaIpcTensorTransportProxy' object is not subscriptable。根因是 Pixtral 在 process_and_combine_mm_data 返回后再按图拆分 IMAGE item，而 #26096 让该函数最后一步把 item.feature 换成语义上无 __getitem__ 的 pool 视图代理；问题在 #32541 自动启用 cuda_ipc 后实际暴露，恢复 opt-in（#34662）后仍可经 MMMUServerBase 复现。只有多图请求受影响，单图走不同分支。

实现拆解

1. 重构 base_processor.py 收尾阶段：process_and_combine_mm_data 尾部原先内联完成 set_pad_value、_precompute_hashes_before_cpu_transfer 和 CUDA IPC 包裹，现改为统一调用新方法 _finalize_mm_items。该方法按固定顺序执行：先调用默认透传的 hook _postprocess_mm_items_before_transport（子类在此做模型特定 reshape，此时 feature 仍是真实 tensor），再补齐 padding 与 hash，最后通过 _prepare_mm_items_for_transport 完成 IPC 包裹。这样“传输前 reshape”从约定变成框架保证。
2. Pixtral 拆分逻辑迁入 hook：pixtral.py 的 process_mm_data_async 删除了“多图返回后再 split”的分支，统一直接返回 process_and_combine_mm_data 结果；原拆分实现搬入 _postprocess_mm_items_before_transport，并抽出 _get_image_nrows 复用缓存的 _effective_patch_size 计算每图 patch 行数。
3. 拆分加固：新 hook 先识别“已拆分”状态（IMAGE item 数量等于图像数量时直接透传，避免通用 get_new_expanded_mm_items 已拆好的单行图被二次切片）；再校验特征行数与

offsets 总数是否与图像 patch 行数严格匹配，不匹配立即抛 ValueError；拆分时用 copy.copy 保留 format、model_specific_data 等字段，重置 hash/pad_value 以便后续阶段重新派生，最后确认所有 offsets 被消费完。

4. 测试配套：新增 test/registered/unit/multimodal/test_pixtral_processor.py，注册到 CPU CI 套件 base-a-test-cpu，使用真实 PIL 几何而非 mock，覆盖“多图在传输前拆分成独立 tensor”“已拆分 items 原样透传”“patch 行数错配 fail-loud”三条路径；PR Test (Base) CI 通过。

关键文件：

- python/sglang/srt/multimodal/processors/base_processor.py（模块 处理基类；类别 source；类型 core-logic；符号 _finalize_mm_items, _postprocess_mm_items_before_transport, _prepare_mm_items_for_transport）：多模态预处理核心路径：新增 _finalize_mm_items 生命周期，把原本内联的 padding、hash、CUDA IPC 包裹拆成固定顺序，并新增可重写 hook _postprocess_mm_items_before_transport，从框架层面保证模型特定 reshape 发生在传输包裹之前。
- python/sglang/srt/multimodal/processors/pixtral.py（模块 多模态处理；类别 source；类型 core-logic；符号 _postprocess_mm_items_before_transport, _get_image_nrows）：问题发生地与主要修复点：把原先在 process_and_combine_mm_data 返回后做的多图拆分迁移到传输前 hook，并新增 _get_image_nrows 与基数校验，消除对 CudaIpcTensorTransportProxy 的切片操作。
- test/registered/unit/multimodal/test_pixtral_processor.py（模块 回归测试；类别 test；类型 test-coverage；符号 TestPixtralProcessor, test_multi_image_features_are_split_before_transport, test_already_split_one_row_images_are_preserved, test_mismatched_patch_rows_fail_loudly）：新增回归测试，注册到 CPU CI 套件 base-a-test-cpu，用真实 PIL 几何验证拆分时机、已拆分透传与基数校验，直接防止本问题回归。

关键符号：BaseMultimodalProcessor._finalize_mm_items, BaseMultimodalProcessor._postprocess_mm_items_before_transport, BaseMultimodalProcessor._prepare_mm_items_for_transport, PixtralProcessor._postprocess_mm_items_before_transport, PixtralProcessor._get_image_nrows, PixtralProcessor.process_mm_data_async, BaseMultimodalProcessor.process_and_combine_mm_data

关键源码片段

python/sglang/srt/multimodal/processors/base_processor.py

多模态预处理核心路径：新增 _finalize_mm_items 生命周期，把原本内联的 padding、hash、CUDA IPC 包裹拆成固定顺序，并新增可重写 hook _postprocess_mm_items_before_transport，从框架层面保证模型特定 reshape 发生在传输包裹之前。

```
def _finalize_mm_items(
    self,
    mm_items: List[MultimodalDataItem],
```

```
    *,
    images: Optional[List[Any]],
) -> List[MultimodalDataItem]:
    # 模型特定 reshape（如 Pixtral 按图拆分）必须在特征仍是真实 tensor 时完成，
    # 默认 hook 直接透传，子类只需覆盖这一个入口。
    mm_items = self._postprocess_mm_items_before_transport(
        mm_items,
        images=images,
    )

    # reshape 完成后统一补齐 padding 与 hash，
    # 这样 hook 里经 copy.copy 产生的新 item 也能拿到正确的派生字段。
    for item in mm_items:
        if item.format in (
            MultimodalInputFormat.PROCESSOR_OUTPUT,
            MultimodalInputFormat.PRECOMPUTED_EMBEDDING,
        ):
            item.set_pad_value()
    self._precompute_hashes_before_cpu_transfer(mm_items)

    # CUDA IPC 包裹固定为最后一步，传输层拿到的只能是最终特征，
    # 从根本上杜绝“返回后再切片代理”这类时序问题。
    return self._prepare_mm_items_for_transport(mm_items)


def _postprocess_mm_items_before_transport(
    self,
    mm_items: List[MultimodalDataItem],
    *,
    images: Optional[List[Any]],
) -> List[MultimodalDataItem]:
    """Apply model-specific item reshaping while features are still tensors."""
    return mm_items


def _prepare_mm_items_for_transport(
    self, mm_items: List[MultimodalDataItem]
) -> List[MultimodalDataItem]:
    # pool miss 时回退为普通 CPU tensor，scheduler 会拷贝并释放每个成功切片。
    if not self.use_cuda_ipc:
        return mm_items
    for item in mm_items:
        if isinstance(item.feature, torch.Tensor):
            item.feature = self._wrap_tensor_for_cuda_ipc(item.feature)
        if isinstance(item.precomputed_embeddings, torch.Tensor):
            item.precomputed_embeddings = self._wrap_tensor_for_cuda_ipc(
                item.precomputed_embeddings
            )
    return mm_items
```

python/sglang/srt/multimodal/processors/pixtral.py

问题发生地与主要修复点：把原先在 process_and_combine_mm_data 返回后做的多图拆分迁移到传输前 hook，并新增 _get_image_nrows 与基数校验，消除对 CudaIpcTensorTransportProxy 的切片操作。

```
def _postprocess_mm_items_before_transport(
    self,
    mm_items: List[MultimodalDataItem],
    *,
    images: Optional[List[Any]],
) -> List[MultimodalDataItem]:
    # 单图或空图不需要拆分，直接进入统一收尾阶段。
    if not images or len(images) <= 1:
        return mm_items

    image_items = [item for item in mm_items if item.modality == Modality.IMAGE]
    # 通用展开（get_new_expanded_mm_items）已经按图拆好时直接透传，
    # 避免对已拆分 item 再次切片造成 feature 丢失。
    if len(image_items) == len(images):
        return mm_items
    # 既不是单个 bundled item、也不是等量拆分，说明上游契约变化，fail loud。
    if len(image_items) != 1:
        raise ValueError(
            "Pixtral multi-image processing expected one bundled IMAGE item or "
            f"{len(images)} split items, but found {len(image_items)}"
        )

    old_item = image_items[0]
    all_offsets = old_item.offsets
    old_feature = old_item.feature
    old_image_sizes = old_item.model_specific_data.get("image_sizes")
    image_nrows = self._get_image_nrows(images)

    # 特征数与 offsets 数都必须和图像 patch 行数严格匹配，防止错位。
    if old_feature is None or len(old_feature) != len(image_nrows):
        raise ValueError(
            "Pixtral multi-image feature count does not match the number of "
            f"images: features={0 if old_feature is None else len(old_feature)}, "
            f"images={len(image_nrows)}"
        )
    if all_offsets is None or sum(image_nrows) != len(all_offsets):
        raise ValueError(
            "Pixtral image patch rows do not match the computed offsets: "
            f"rows={sum(image_nrows)}, "
            f"offsets={0 if all_offsets is None else len(all_offsets)}"
        )

    # 依次为每张图像切出 offsets，复制 old_item 并重置派生字段；
```

```
    # hash/pad_value 会在 _finalize_mm_items 后续阶段重新计算。
    split_items = [item for item in mm_items if item.modality != Modality.IMAGE]
    offset_idx = 0
    for image_idx, num_rows in enumerate(image_nrows):
        item_offsets = all_offsets[offset_idx : offset_idx + num_rows]
        offset_idx += num_rows
        new_item = copy.copy(old_item)
        new_item.feature = old_feature[image_idx : image_idx + 1]
        new_item.offsets = item_offsets
        new_item.model_specific_data = copy.copy(old_item.model_specific_data)
        if old_image_sizes is not None:
            new_item.model_specific_data["image_sizes"] = old_image_sizes[
                image_idx : image_idx + 1
            ]
        new_item.hash = None
        new_item.pad_value = None
        split_items.append(new_item)

    # 所有 offsets 必须被消费完，否则说明行数计算与拼接结果不一致。
    if offset_idx != len(all_offsets):
        raise ValueError(
            "Pixtral multi-image split did not consume every offset: "
            f"consumed={offset_idx}, offsets={len(all_offsets)}"
        )
    return split_items


def _get_image_nrows(self, images: List[Any]) -> List[int]:
    # 用缓存的 effective patch 计算每张图在 token 网格中的行数，
    # 供拆分时切分 offsets 并做基数校验。
    image_nrows = []
    for image in images:
        width, height = image.size
        ratio = max(width / self.image_size, height / self.image_size)
        if ratio > 1:
            width = int(math.floor(width / ratio))
            height = int(math.floor(height / ratio))
        num_rows, _ = _get_pixtral_hf_num_image_tokens(
            (height, width),
            (self._effective_patch_size, self._effective_patch_size),
        )
        image_nrows.append(num_rows)
    return image_nrows
```

评论区精华

该 PR 没有实质性的 review 对话，mickqian 直接 APPROVED，仅有一条 /tag-and-rerun-ci 的 CI 重跑评论。技术论证集中在 PR body：作者完整梳理了根因链（#20708 引入拆分 → #26096 把 wrap 放进 process_and_combine_mm_data 尾部 → #32541 自动启用后问题上

线 → #34662 恢复 opt-in），并明确说明 test_ministral4_models.py 的 H100 失败与本 PR 无关——90 个 MMMU 样本中只有 1 个走多图路径，最多影响 1 个样本，而 H100 的 0.4333/0.4111 与 0.45 阈值之间是 2–4 个样本的差距，阈值问题单独跟踪。

- CI 重跑请求 (other): Review 由 mickqian 直接 APPROVED，无 review comments；CI 状态显示 PR Test (Base) 通过，PR Test (Extra) 有一次失败记录。

风险与影响

- 风险：
  1. base_processor 生命周期改动影响所有多模态处理器：_finalize_mm_items 现在统一经过所有 processor，虽然默认 hook 透传、行为等价，但若有其他 processor 或调用方依赖原先“pad/hash/wrap 后返回”的隐式顺序，可能受影响，需要关注周边多模态测试。
  2. 新增 fail-loud 异常路径：pixtral.py 新增三类 ValueError（item 数量异常、特征数不匹配、offsets 未消费完），原本可能静默出错或产生空特征的输入会显式失败，需确认上层能把新异常正确转换为请求错误而非崩溃。
  3. 测试覆盖深度有限：单测用 object.__new__ 绕过 __init__，只覆盖 hook 与 _finalize_mm_items 的组合，未覆盖完整 process_mm_data_async 链路；E2E 仅在 2×H200 单节点验证，其他硬件 / 后端组合仍需观察。
  4. 浅拷贝语义：copy.copy 复制 old_item 后，model_specific_data 换成新 dict，但其中非切片字段仍共享引用，若后续有人原地修改可能串扰；当前流程中 hash/pad_value 已被重置并在后续阶段重算，风险可控。 - 影响：用户侧：修复启用 CUDA IPC 时 Pixtral 多图请求（如 Mistral-Small-4）的 HTTP 500，单图请求与未启用 cuda_ipc 的部署行为保持不变。系统侧：多模态预处理管线从此具备明确的“传输前 reshape”hook 点，process_and_combine_mm_data 的返回内容语义更稳定，任何子类都不应再在返回后修改 item。团队侧：新增 CPU CI 回归测试进入 base-a-test-cpu 套件，可防止同类“拆分层与传输层顺序错位”问题复发；同时为后续 VLM 处理器提供了一个可复用的生命周期扩展点。 - 风险标记：核心路径生命周期重构，多模态处理器全量受影响，新增 fail-loud 异常路径，测试仅覆盖 hook 层

关联脉络

- 暂无明显关联 PR