# PR #33921 完整报告

sgl-project/sglang

[Kimi K3] Preprocess CPU-transport images on the vision owner

合并时间：2026-08-09 16:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33921

## 执行摘要

- 一句话：K3 图片预处理延迟至 vision owner，高负载 TTFT -79%
- 推荐动作：值得精读。这是多模态特征传输中"载荷最小化 + 延迟物化到消费方"的完整案例：决策门控（transport + dtype + 字节启发式）、数据契约（uint8 feature + deferred config 键）、消费端单点物化与调度白名单四者缺一不可。建议阅读顺序：processor 的 _should_defer_gpu_preprocessing 与 _build_deferred_output → 模型侧 materialize_item_features 的 deferred 分支 → mm_schedule.py 的白名单一行。合并后建议补充：非 EPD 启动路径回归测试、混合 batch 的显式优雅处理（当前直接 ValueError）。

## 功能与动机

Kimi-K3 在 CPU 多模态传输下，原先在 tokenizer worker GPU 上完成全部图像预处理，再把膨胀后的 FP32 patch 张量拷回 CPU、广播 / 序列化到每个 scheduler GPU 并做 H2D，之后才选出 vision owner。PR body 明确指出：DP helper 在调用本地 feature loader 之前就已确定 owner，因此预处理与 H2D 都只需要发生在该 owner 上。这样既把跨机传输载荷从处理后 FP32 张量降为原始 uint8 像素，也消除了非 owner rank 上的冗余预处理与 H2D 拷贝。

## 实现拆解

1. 新增公共图像处理模块 python/sglang/srt/multimodal/kimi_k3_image_processing.py：把原先内联在 processor 中的 _chessboard_background、_fill_transparent_bg 收敛为 fill_transparent_bg，并新增 to_chw_uint8（PIL/ 张量 → uint8 CHW，支持指定 device）、normalization_tensors（mean/std → scale/bias 预计算）与契约常量 DEFERRED_PREPROCESSING_KEY，保证 eager 与 deferred 两端共用同一套像素转换与背景合成实现。
2. Processor 决策层（multimodal/processors/kimi_k3.py）：KimiK3GPUProcessorWrapper.prepare_deferred 只产出 token 展开结果与 NaViT resize_config，不执行 GPU 预处理；KimiK3ImageProcessor._should_defer_gpu_preprocessing 用四个条件收敛延迟范围（传输方式为 cpu、CUDA 可用、全部输入为 PIL 或 uint8 张量、原始载荷字节数 ≤ 处理后 FP32 张量字节数）；_build_deferred_output 把每张图包装为 feature 为 uint8 CPU 张量的 MultimodalDataItem，在 model_specific_data 中写入 DEFERRED_PREPROCESSING_KEY（含 image_mean、image_std、transparent_bg_config、resize_config），并在 process_mm_data_async

中提前返回。

3. 模型消费端（models/kimi_k3.py）：get_image_feature 内嵌的 materialize_item_features 在 DP owner 的 loader 中检测 deferred 标记；命中后调用 K2.5 的 _gpu_preprocess_images，以 to_chw_uint8(device=device) 完成唯一一次 H2D、fill_transparent_bg 完成背景合成、normalization_tensors 生成归一化系数；IPC consumer count 从 server_args.tp_size 改为 get_parallel().tp_size 直接取值。同一 batch 若混有 deferred 与非 deferred item 会抛 ValueError（当前为硬失败策略）。
4. 调度配套（managers/mm_schedule.py）：_can_skip_pre_embed_feature_move 白名单加入 KimiK3ForConditionalGeneration，使 scheduler 在 embedding 之前跳过 _move_items_to_device，保证 uint8 feature 不被每个 rank 提前搬上 GPU，owner-only H2D 语义才真正成立。
5. 测试与验证配套：test/registered/unit/models/test_kimi_k25.py 新增 6 个用例（cpu transport 延迟、字节启发式边界、非 uint8 不延迟、空 batch 不延迟、float 张量 eager 兼容、cuda_ipc/fabric 保持 eager）；test/registered/unit/models/test_kimi_k3_vision.py 新增 owner-only 预处理与调度器跳过移动两个用例，并同步更新 topology mock 与 NaN-safe 断言。无新增配置项或部署配套。

关键文件：

- python/sglang/srt/multimodal/processors/kimi_k3.py（模块 预处理决策；类别 source；类型 core-logic；符号 _should_defer_gpu_preprocessing, _build_deferred_output, prepare_deferred, _k3_to_cuda_chw）：延迟决策与 deferred 输出构造的核心入口：_should_defer_gpu_preprocessing 定义何时延迟，_build_deferred_output 定义 uint8 feature + deferred config 的数据契约，prepare_deferred 提供 token 展开与 resize 配置。
- python/sglang/srt/multimodal/kimi_k3_image_processing.py（模块 图像处理；类别 source；类型 core-logic；符号 to_chw_uint8, fill_transparent_bg, normalization_tensors, DEFERRED_PREPROCESSING_KEY）：新增公共图像处理模块，承载 DEFERRED_PREPROCESSING_KEY 契约，统一 eager/deferred 两端的像素转换、背景合成与归一化系数计算。
- python/sglang/srt/models/kimi_k3.py（模块 模型消费端；类别 source；类型 data-contract；符号 materialize_item_features, get_image_feature）：模型消费端：materialize_item_features 在 DP owner 上执行延迟预处理，复用 K2.5 的 _gpu_preprocess_images，是 deferred 数据契约的最终落地处；同时调整 IPC consumer count 取值来源。
- python/sglang/srt/managers/mm_schedule.py（模块 调度器；类别 source；类型 core-logic；符号 _can_skip_pre_embed_feature_move）：把 KimiK3ForConditionalGeneration 加入 _can_skip_pre_embed_feature_move 白名单，确保 scheduler 不在 embedding 前把每张图 feature 提前搬到 GPU，是 owner-only H2D 语义生效的关键一行。
- test/registered/unit/models/test_kimi_k25.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_kimi_k3_cpu_transport_defers_gpu_preprocessing, test_kimi_k3_defers_only_when_raw_transport_is_smaller,

test_kimi_k3_does_not_defer_non_uint8_tensor_preprocessing,test_kimi_k3_does_not_defer_empty_image_batch）：覆盖延迟判定边界与 eager 兼容性：cpu transport 延迟、字节启发式、非 uint8 不延迟、空 batch 不延迟、float 张量eager 兼容、cuda_ipc/fabric 保持 eager。

- test/registered/unit/models/test_kimi_k3_vision.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_kimi_k3_preprocesses_only_dp_owner_images, test_kimi_k3_scheduler_leaves_feature_placement_to_dp_owner, test_kimi_k3_encoder_dp_defers_feature_materialization）：验证 owner-only 预处理（只对被分配的图片执行 _gpu_preprocess_images）与调度器跳过特征移动，并更新 topology mock 与 NaN-safe 断言。

关键符号：_should_defer_gpu_preprocessing, _build_deferred_output, prepare_deferred, to_chw_uint8, fill_transparent_bg, normalization_tensors, materialize_item_features, get_image_feature, _can_skip_pre_embed_feature_move

## 关键源码片段

### python/sglang/srt/multimodal/processors/kimi_k3.py

延迟决策与 deferred 输出构造的核心入口：_should_defer_gpu_preprocessing 定义何时延迟，_build_deferred_output 定义 uint8 feature + deferred config 的数据契约，prepare_deferred 提供 token 展开与 resize 配置。

```
def _should_defer_gpu_preprocessing(self, images) -> bool:
    # 只有 CPU 特征传输 + CUDA 环境下，且全部输入为 PIL 图像或 uint8 张量时，
    # 才考虑延迟预处理；float 张量、预计算 embedding 等路径保持 eager 行为。
    if (
        not images
        or self.mm_feature_transport != "cpu"
        or not is_cuda()
        or not all(
            isinstance(image, Image.Image)
            or (isinstance(image, torch.Tensor) and image.dtype == torch.uint8)
            for image in images
        )
    ):
        return False

    raw_bytes = 0
    processed_bytes = 0
    patch_size = self._processor._patch_size
    for image in images:
        width, height = _get_image_dimensions(image)
        # 用与 eager 路径相同的 NaViT resize 配置，估算处理后 FP32 patch 张量字节数；
        # 只有延迟真正能减小跨机传输载荷时，才值得走 deferred 路径。
        resize_config = navit_resize_config(
            width,
            height,
```

```
            patch_size,
            self._processor._merge_kernel_size,
            self._processor._in_patch_limit,
            self._processor._patch_limit_on_one_side,
            self._processor._fixed_output_tokens,
        )
        if isinstance(image, torch.Tensor):
            # 2D 或单通道按 3 通道计，否则按张量首维通道数计
            channels = (
                3 if image.dim() == 2 or image.shape[0] == 1 else image.shape[0]
            )
        else:
            # PIL：含 alpha 或 transparency 信息键按 4 通道计，其余按 3 通道计
            channels = (
                4
                if image.mode != "RGB"
                and ("A" in image.getbands() or "transparency" in image.info)
                else 3
            )
        raw_bytes += channels * width * height
        padded_width = resize_config["new_width"] + resize_config["pad_width"]
        padded_height = resize_config["new_height"] + resize_config["pad_height"]
        processed_bytes += 3 * padded_width * padded_height * torch.float32.itemsize

    # 只在原始载荷不比处理后张量更大时延迟，避免超大原图（如 4K）反而传得更慢
    return raw_bytes <= processed_bytes
```

## python/sglang/srt/multimodal/kimi_k3_image_processing.py

新增公共图像处理模块，承载 DEFERRED_PREPROCESSING_KEY 契约，统一 eager/deferred 两端的像素转换、背景合成与归一化系数计算。

```
DEFERRED_PREPROCESSING_KEY = "kimi_k3_deferred_preprocessing"


def to_chw_uint8(
    image: Union[torch.Tensor, Image.Image],
    device: torch.device | str | None = None,
) -> torch.Tensor:
    # 只有 uint8 原始像素适合跨 CPU transport 延迟传输；已是张量则要求 uint8，
    # PIL 图像在这里统一转成 CHW uint8，灰度图扩展为 3 通道。
    if isinstance(image, Image.Image):
        # RGB 图像即使带 stray transparency 信息键也绝不提升为 RGBA，
        # 以对齐 checkpoint 的 fill_transparent_bg_with() 语义
        has_alpha = image.mode != "RGB" and (
            "A" in image.getbands() or "transparency" in image.info
        )
        array = np.array(image.convert("RGBA" if has_alpha else "RGB"), copy=True)
        image = torch.from_numpy(array).permute(2, 0, 1)
```

```
    if image.dtype != torch.uint8:
        raise ValueError(
            f"Kimi-K3 preprocessing expects raw uint8 pixels, got {image.dtype}"
        )
    if image.dim() == 2:
        image = image.unsqueeze(0)
    if image.shape[0] == 1:
        image = image.repeat(3, 1, 1)
    if device is not None:
        image = image.to(device) # 延迟到 owner rank 上的唯一一次 H2D
    return image


def normalization_tensors(
    image_mean: list[float],
    image_std: list[float],
    device: torch.device | str,
) -> tuple[torch.Tensor, torch.Tensor]:
    # 把 mean 与 std 预计算为 scale 与 bias，供 _gpu_preprocess_images 直接做
    # 线性归一化，结果与 eager 路径的逐像素 (x/255 - mean) / std 完全一致
    scale = torch.tensor(
        [1.0 / (255.0 * std) for std in image_std],
        device=device,
        dtype=torch.float32,
    ).view(1, 3, 1, 1)
    bias = torch.tensor(
        [-mean / std for mean, std in zip(image_mean, image_std)],
        device=device,
        dtype=torch.float32,
    ).view(1, 3, 1, 1)
    return scale, bias
```

## python/sglang/srt/models/kimi_k3.py

模型消费端：materialize_item_features 在 DP owner 上执行延迟预处理，复用 K2.5 的 _gpu_preprocess_images，是 deferred 数据契约的最终落地处；同时调整 IPC consumer count 取值来源。

```
def materialize_item_features(image_indices: List[int]) -> torch.Tensor:
    """只物化分配给本 vision-DP rank 的图片特征。"""
    ipc_consumer_count = max(get_parallel().tp_size, 1)
    device_index = device.index
    if device.type == "cuda" and device_index is None:
        device_index = torch.cuda.current_device()

    selected_items = []
    for image_index in image_indices:
        item = items[image_index]
        if device.type == "cuda":
            # CUDA-IPC 代理仍按原逻辑在分配确定后重建，跨边界只传一次
```

```
            item.reconstruct(device_index, ipc_consumer_count=ipc_consumer_count)
        selected_items.append(item)

    # 延迟预处理分支：feature 是跨 CPU transport 传过来的 uint8 原始像素，
    # resize、归一化、patchify 全部推迟到本 owner rank 上执行。
    deferred = [
        item.model_specific_data.get(DEFERRED_PREPROCESSING_KEY)
        for item in selected_items
    ]
    if any(config is not None for config in deferred):
        # 同一 batch 内不允许混用延迟与已预处理 feature，否则契约无法自洽
        if not all(config is not None for config in deferred):
            raise ValueError(
                "Kimi-K3 cannot mix deferred and preprocessed image features"
            )
        # 复用 K2.5 的 GPU 预处理管线，由 deferred 配置驱动归一化与背景合成
        from sglang.srt.multimodal.processors.kimi_k25 import _gpu_preprocess_images

        first_config = deferred[0]
        image_scale, image_bias = normalization_tensors(
            first_config["image_mean"], first_config["image_std"], device
        )
        pixel_values, _ = _gpu_preprocess_images(
            [item.feature for item in selected_items], # uint8 CPU 张量
            [config["resize_config"] for config in deferred], # NaViT resize 配置
            image_scale,
            image_bias,
            self.vision_tower.patch_size,
            to_chw=lambda image: to_chw_uint8(image, device=device), # 唯一一次 H2D
            post_resize=lambda x: fill_transparent_bg(
                x, first_config["transparent_bg_config"]
            ),
        )
        return pixel_values.to(dtype=target_dtype)

    # 非延迟分支：直接对已预处理 feature 做统一物化（与原逻辑一致）
    features = []
    for item in selected_items:
        if not isinstance(item.feature, torch.Tensor):
            raise TypeError(
                "Kimi-K3 image feature must be a torch.Tensor, "
                f"got {type(item.feature)}"
            )
        features.append(item.feature)
    return materialize_multimodal_features(
        features, device=device, dtype=target_dtype
    )
```

## 评论区精华

PR 没有 review 评论，但 issue 中有两条值得注意的讨论：1）yhyang201 的回归报告：非 EPD 单节点服务在 warmup 阶段因 models/kimi_k3.py 消费者读取 first_config['backend'] 抛 KeyError 崩溃，而生产者 prepare_deferred 只写入 image_mean、image_std、transparent_bg_config，属于 producer/consumer 数据契约不一致；最终 head 的消费者已移除 backend 分支选择，统一走 _gpu_preprocess_images，读取键与生产者完全对齐，因此该具体错误在最终形态下应已消除，但提交历史无法确认修复时机，也没有针对非 EPD 启动路径的独立回归用例。2）mickqian 的 draft 说明：在 GPU 资源不可用（GB300/B300 均 0 可用）的情况下，microbenchmark 不足以为 serving 性能背书，因此保留 draft 等待严格真实权重 A/B；PR body 最终补齐了该证据。

- KeyError: 'backend' 导致非 EPD 服务路径崩溃 (correctness): 最终 head 的消费者已移除 backend 分支选择，deferred 分支统一走 _gpu_preprocess_images，读取键与生产者输出对齐；据此推断该回归已在迭代中解决，但提交历史未标明修复时机，也没有针对非 EPD 启动路径的独立回归用例，建议合并后复核该路径。
- draft 状态与严格真实权重 A/B 证据要求 (question): PR body 最终附带了真实权重 A/B 数据（unlimited 吞吐 +248.3%、中位 TTFT -78.7%、BF16 逐位一致），满足作者自己设定的证据门槛。

## 风险与影响

- 风险：
  1. 非 EPD 回归风险：yhyang201 报告的 KeyError: 'backend' 虽在最终 head 上不复现（deferred 分支不再访问 backend 键），但无独立回归测试锁定该路径，后续若有人重新引入 backend 分支选择，风险仍在。
  2. 混合 batch 硬失败：materialize_item_features 对 deferred 与 eager 混批直接抛 ValueError；延迟判定是 per-request 的，同一 forward batch 内不同请求可能一个被延迟、一个不延迟（如 float 张量请求），该路径会中断服务，缺少优雅降级。
  3. 调度行为变更：_can_skip_pre_embed_feature_move 白名单新增 K3 后，所有 K3 请求在 embedding 前不再被 scheduler 搬到 GPU，依赖模型内部 materialize；use_data_parallel=False 等非 DP 拓扑下行为未充分覆盖，一旦回归可能表现为显存或性能异常而非显式报错。
  4. IPC consumer count 取值来源从 server_args.tp_size 改为 get_parallel().tp_size，在 EPD 等异构拓扑下语义可能不同，测试 mock 已同步但线上拓扑未验证。
  5. 字节大小启发式在边界尺寸可能翻转：raw_bytes <= processed_bytes 的估算会随分辨率、通道数与 in_patch_limit 变化，不同请求可能走不同路径，性能表现不稳定（不会 crash）。
  6. 低负载 P90 回归：rate 2 时 P90 TTFT +20%，作者自述为噪声，但说明延迟判定在低并发下收益不稳。 - 影响：用户侧：Kimi-K3 + cpu 传输（结合 #34662 使单节点默认回退 CPU 后成为常用路径）的用户收益显著——unlimited 并发吞吐提升约 2.5 倍、中位 TTFT 降约 4.7 倍，4×1536×1024 多图场景微基准提速 12.76 倍。系统侧：跨 processor、model、scheduler 三层引入新的延迟物化数据契约（

DEFERRED_PREPROCESSING_KEY），CUDA IPC 与 CUDA VMM 传输路径不受影响（保持 *eager*）；影响面限定在 Kimi-K3 与 *cpu transport* 组合，对其他模型无行为变化。团队侧：确立了“载荷最小化 + *owner* 延迟物化”的多模态传输模式，为同族模型提供了可复制的设计范式，测试基线新增 8 个以上用例。 - 风险标记：非 EPD 路径回归报告，混合 *batch* 直接抛 ValueError，调度跳过 *pre-embed* 特征移动，*deferred* 数据契约变更，IPC *consumer count* 取值来源变化，低负载 P90 TTFT 抖动

## 关联脉络

- PR #34662 fix: *restore* VLM nightly *regression coverage*: 该 PR 使单节点多模态传输默认回退 CPU，直接放大了本 PR 的优化价值：Kimi-K3 单节点服务的默认路径即为本 PR 优化的 *cpu transport* 路径。
- PR #34398 [VLM] Add *content-addressed preprocessing cache* infrastructure: 同属多模态预处理管线演进：一条线是缓存已算好的特征（内容寻址预处理缓存），另一条线是本 PR 的延迟物化，两者互补，共同指向减少重复预处理与传输。