PR #52827 完整报告

vllm-project/vllm

[MM] Keep more metadata tensors on CPU

合并时间：2026-08-19 10:56

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52827

执行摘要

- 一句话：多模态元数据 tensor 标记 keep_on_cpu，减少 GPU 拷贝与显存占用
- 推荐动作：值得精读。这个 PR 是“数据契约先行”的典型范例：通过字段级标记把设备放置语义显式化，避免在传输层做启发式判断，为后续 prefix cache 过滤和分布式 KV 复用铺路。建议重点学习 keep_on_cpu 标记与消费端 .to() 恢复的对称设计，以及如何在不破坏 HF 实现的前提下把 tensor 留在 CPU。维护者可结合各模型实际消费路径补充逐字段核对，确保没有漏网的 GPU 依赖。

功能与动机

PR body 明确指出：A bunch of multimodal metadata tensors aren't needed on GPU yet are copied there anyhow. We can easily flag these so that they won't be copied。并说明该改动将服务于 https://github.com/vllm-project/vllm/pull/52041：在 prefix cache 已覆盖时，这些被标记的 tensor 可以被直接跳过，不需要重新传输或计算。

实现拆解

本 PR 的核心是给多模态字段配置的三种构造方式 MultiModalFieldConfig.batched()/flat_from_sizes()/shared() 统一追加 keep_on_cpu=True，并在所有需要这些字段的设备侧消费点显式搬回，具体分四步：

1. 字段标记（数据契约层）：在 vllm/model_executor/models/ 下 20+ 个模型的 _get_mm_fields_config / _field_config 中，把纯元数据字段（如 image_num_crops、image_grid_thw、video_start_times、audio_feature_lengths、num_tokens_per_image、imgs_sizes、video_audio_n_segs 等）标记为 keep_on_cpu=True。这些字段只参与 embedding 切分、token 数计算和 prompt 重建，不参与 GPU 上的张量运算。代表性文件：molmo2.py、minicpmv4_6.py、llava_onevision2.py、keye_vl1_5.py、hyperclovax_vision.py、nano_nemotron_vl.py、mimo_v2_omni.py。
2. 消费端显式搬回（正确性适配）：标记字段留在 CPU 后，凡是被传给 HF 实现消费的地方必须显式 .to(device, non_blocking=True)。vllm/model_executor/models/transformers/multimodal.py 的 _process_image_input 在调用 HF get_image_features 前遍历 kwargs，把所有 torch.Tensor 搬回 pixel_values.device。
3. 音频 / 视频路径的特殊处理：qwen2_5_omni_thinker.py 与 qwen3_asr.py 的 _process_audio_input 对 audio_feature_lengths 单独做 .to(input_features.device,

non_blocking=True)，因为音频塔会根据 feature_lens 推导设备放置；use_audio_in_video 这类共享标量字段也一并标记。

4. 测试与配置配套：本次没有新增或修改测试文件，回归保障完全依赖现有 CI 测试矩阵和 Buildkite 验证（/ci run 已触发 #84489）。所有模型字段的 keep_on_cpu 标记会作为后续 prefix cache 过滤（PR #52041）的输入契约。

关键文件：

- vllm/model_executor/models/transformers/multimodal.py（模块 多模态适配；类别 source；类型 data-contract；符号 _get_mm_fields_config, _process_image_input）：所有 transformers 系多模态模型的通用适配层，本次在 _get_mm_fields_config 中把 num_audio_tokens、image_grid_thw、video_grid_thw、num_image_patches 标记为 keep_on_cpu，并在 _process_image_input 中新增显式搬回 device 的循环，是正确性适配的核心，直接影响所有走 HF get_image_features 的模型。
- vllm/model_executor/models/qwen2_5_omni_thinker.py（模块 音频视觉模型；类别 source；类型 data-contract；符号 _qwen2_5_omni_thinker_field_config, create_qwen2_5_omni_thinker_field_factory, _process_audio_input）：qwen 全模态模型的字段工厂将音频注意力掩码、特征长度、视频时间戳等大量元数据标记为 keep_on_cpu，并在 _process_audio_input 中显式搬回 audio_feature_lengths，是音频路径适配的典型代表。
- vllm/model_executor/models/molmo2.py（模块 视觉模型；类别 source；类型 data-contract；符号 _get_mm_fields_config）：本次改动量最大的模型文件之一（+12/-8），把图像 / 视频的 crops、pooled patches、token 数等 12 个元数据字段全部标记为 keep_on_cpu，覆盖了最完整的字段分类样本，可作为其他模型改动的参照模板。
- vllm/model_executor/models/mimo_v2_omni.py（模块 视觉模型；类别 source；类型 data-contract；符号 _get_mm_fields_config）：视频时间戳、音频 token 长度以及 video_audio 分段信息被标记为 keep_on_cpu，是视频音频联合模态场景下字段分类的代表性改动。
- vllm/model_executor/models/nano_nemotron_vl.py（模块 视觉模型；类别 source；类型 data-contract；符号 _get_image_fields_config, _get_video_fields_config, _get_audio_fields_config）：图像、视频、音频三模态的尺寸与索引字段（image_num_patches、frames_indices、frame_duration_ms 等）全部标记为 keep_on_cpu，展示了一个模型内多模态统一的字段管理方式。
- vllm/model_executor/models/hyperclovax_vision.py（模块 视觉模型；类别 source；类型 data-contract；符号 _get_mm_fields_config）：将图像尺寸与视觉查询长度（image_sizes_images、vision_query_lengths_images/videos）标记为 keep_on_cpu，说明纯视觉模型的字段分类同样适用，体现该模式的通用性。
- vllm/model_executor/models/minicpmv4_6.py（模块 视觉模型；类别 source；类型 data-contract；符号 _minicpmv4_6_field_config）：tgt_sizes、video_image_sizes、video_tgt_sizes、use_vit_merger 等字段被标记，属于小型模型上同模式改动的代表，帮助确认本次改动覆盖到所有已有多模态模型家族。

关键符号：_get_mm_fields_config, _process_image_input, _process_audio_input, _qwen2_5_omni_thinker_field_config, _get_image_fields_config,

_get_video_fields_config, _get_audio_fields_config,
_minicpmv4_6_field_config,_keye_field_config, _field_config

关键源码片段

vllm/model_executor/models/transformers/multimodal.py

所有 transformers 系多模态模型的通用适配层，本次在 _get_mm_fields_config 中把 num_audio_tokens、image_grid_thw、video_grid_thw、num_image_patches 标记为 keep_on_cpu，并在 _process_image_input 中新增显式搬回 device 的循环，是正确性适配的核心，直接影响所有走 HF get_image_features 的模型。

```
def _process_image_input(self, **kwargs) -> list[torch.Tensor] | None:
    pixel_values: torch.Tensor | None = kwargs.pop("pixel_values", None)
    image_embeds: torch.Tensor | None = kwargs.pop("image_embeds", None)
    # 部分模型使用 image_patches 作为输入键名，这里统一归一化
    if pixel_values is None:
        pixel_values = kwargs.pop("image_patches", None)

    if image_embeds is not None:
        return [image_embeds]

    if pixel_values is None:
        return None

    num_image_patches = kwargs.pop("num_image_patches")

    # image_grid_thw、video_grid_thw 等字段已注册为 keep_on_cpu=True，
    # 平时留在 CPU 上；但 HF 的 get_image_features 内部期望这些 tensor
    # 与 pixel_values 处于同一设备，因此这里显式地按需搬回。
    for key, value in kwargs.items():
        if isinstance(value, torch.Tensor):
            kwargs[key] = value.to(pixel_values.device, non_blocking=True)

    # HF 的视觉前向内部可能包含同步点，例如 Idefics3 的
    # 全零 padding 过滤和 LlavaOnevision 的逐样本分叉。
    with gpu_sync_allowed():
        vision_embeddings = self.model.get_image_features(pixel_values, **kwargs)

    # transformers v5 起 get_image_features 可能返回 tuple（含 attention、
    # hidden_states），统一取第一个元素作为 embedding 输出
    if isinstance(vision_embeddings, tuple):
        vision_embeddings = vision_embeddings[0]
    elif isinstance(vision_embeddings, dict):
        vision_embeddings = vision_embeddings.pooler_output

    if isinstance(vision_embeddings, torch.Tensor):
        # 每个图片 item 的 token 数只在切分时使用，保持 CPU 即可
        split_sizes = num_image_patches.flatten().tolist()
```

```
            return self._split_embeddings(vision_embeddings, split_sizes)

        return list(vision_embeddings)
```

vllm/model_executor/models/qwen2_5_omni_thinker.py

qwen 全模态模型的字段工厂将音频注意力掩码、特征长度、视频时间戳等大量元数据标记为 keep_on_cpu，并在 _process_audio_input 中显式搬回 audio_feature_lengths，是音频路径适配的典型代表。

```
def _qwen2_5_omni_thinker_field_config(hf_inputs: Mapping[str, torch.Tensor]):
    audio_feature_lengths = hf_inputs.get("audio_feature_lengths", torch.empty((0,)))
    image_grid_thw = hf_inputs.get("image_grid_thw", torch.empty((0, 3)))
    image_pixel_grid_sizes = image_grid_thw.prod(-1)
    image_embed_grid_sizes = (
        image_pixel_grid_sizes // spatial_merge_size // spatial_merge_size
    )

    return dict(
        input_audio_features=MultiModalFieldConfig.flat_from_sizes(
            "audio", audio_feature_lengths, dim=1
        ),
        # 音频注意力掩码与特征长度只参与长度计算与 embedding 切分，
        # 不需要随 input_audio_features 一起搬到 GPU
        feature_attention_mask=MultiModalFieldConfig.batched(
            "audio", keep_on_cpu=True
        ),
        audio_feature_lengths=MultiModalFieldConfig.batched(
            "audio", keep_on_cpu=True
        ),
        pixel_values=MultiModalFieldConfig.flat_from_sizes(
            "image", image_pixel_grid_sizes
        ),
        image_embeds=MultiModalFieldConfig.flat_from_sizes(
            "image", image_embed_grid_sizes
        ),
        # grid_thw 只用于推导 embedding 的切分尺寸
        image_grid_thw=MultiModalFieldConfig.batched("image", keep_on_cpu=True),
        video_grid_thw=MultiModalFieldConfig.batched("video", keep_on_cpu=True),
        second_per_grid_ts=MultiModalFieldConfig.batched(
            "video", keep_on_cpu=True
        ),
        use_audio_in_video=MultiModalFieldConfig.shared(
            "video", num_videos, keep_on_cpu=True
        ),
    )


def _process_audio_input(
    self, audio_input: Qwen2_5OmniAudioFeatureInputs
```

```
) -> torch.Tensor:
    input_features = audio_input["input_features"]
    # audio_feature_lengths 已标记 keep_on_cpu；音频塔会从 feature_lens
    # 推导设备放置，因此需要显式搬回 input_features 所在设备
    audio_feature_lengths = audio_input["audio_feature_lengths"].to(
        input_features.device, non_blocking=True
    )
    audio_feat_lengths, audio_output_lengths = (
        self.audio_tower._get_feat_extract_output_lengths(audio_feature_lengths)
    )
    with gpu_sync_allowed():
        audio_outputs = self.audio_tower(
            input_features.to(self.audio_tower.dtype),
            feature_lens=audio_feature_lengths,
            aftercnn_lens=audio_feat_lengths,
        )
        split_sizes = audio_output_lengths.tolist()
    return audio_outputs.last_hidden_state.split(split_sizes)
```

评论区精华

本次 review 实质讨论很少，审阅由维护者 WoosukKwon 直接批准：

- claude[bot] 自动提示：由于 PR 来自 fork，自动 review 被禁用，维护者可评论 @claude review 触发一次性 review，但最终未触发，直接由 WoosukKwon 给出 APPROVED。
- 设计层面的关键讨论实际发生在 PR body 而非评论区：作者明确将 keep_on_cpu 标记与 PR #52041 的 prefix cache 过滤绑定，说明这是一次为后续功能铺路的数据契约调整，而非单纯的性能微优化。
- fork 仓库自动 review 被禁用 (other): 未触发一次性 review，维护者 WoosukKwon 直接给出 APPROVED 完成合入。
- keep_on_cpu 标记与 prefix cache 过滤的联动设计 (design): 作为后续 PR 的输入契约落地，本次仅完成标记本身，未在评论中展开进一步讨论。

风险与影响

- 风险：
  1. device 不匹配回归风险：标记 keep_on_cpu 后，任何未适配的消费点若隐式假设 tensor 在 GPU，会触发 device mismatch 崩溃。PR 只修补了 transformers/multimodal.py 通用图像路径和 qwen 系列音频路径，其余模型（如 molmo2.py、minicpmv4_6.py、hyperclovax_vision.py）若存在类似的 HF 消费路径，可能依赖通用机制兜底，存在漏网风险。
  2. 误标风险：若某个字段实际在 GPU 计算中被使用（如 mrope 位置计算、vision encoder 内部），误标会导致隐性性能回退或功能错误。本 PR 未给出逐字段的消费点清单，审阅依赖维护者对模型语义的把握。
  3. 测试覆盖缺失：本次 38 个文件改动没有任何测试文件配套，只能依赖现有 CI 多模型测试矩阵验证，遇到设备断言类错误可能只在特定模型组合下暴露。

4. 与后续功能的耦合：keep_on_cpu 标记将被 PR #52041 用作 prefix cache 命中时跳过 tensor 的判据，标记错误会把缺陷扩散到缓存路径，影响面被放大。 - 影响：用户与系统层面：多模态推理（图文、音视频、OCR 等）时减少 H2D 拷贝与 GPU 显存占用，对长视频、多图混排场景收益明显；该优化对所有后端（NVIDIA、CPU、XPU、ROCm）均生效，因为发生在多模态数据装配层。团队层面：确立了一条字段分类约定——纯元数据字段应标记 keep_on_cpu，后续新增多模态模型时需遵循该模式；同时为 prefix cache 过滤提供了必要的 tensor 维度信息，属于架构演进的前置步骤。影响范围广（38 个文件、覆盖 llama/qwen/deepseek/kimi/cohere 等模型线），但改动模式机械、风险集中在消费端适配。 - 风险标记：跨 38 文件数据契约调整，无新增测试覆盖，device 不匹配回归风险，与 prefix cache 过滤联动

关联脉络

- PR #52041 filtering which tensors can be skipped when already covered by the prefix cache: PR body 直接引用该 PR：keep_on_cpu 标记将被用于判断 prefix cache 命中时可跳过的多模态 tensor，本次改动是它的前置数据契约准备。
- PR #52697 [EPD] Allow KV consumers to omit MM embeddings: 同为多模态数据契约方向的演进，围绕多模态输入在消费端如何按需裁剪与传递，与本 PR 的字段级设备放置管理属于同一技术脉络。