

PR #42785 完整报告

vllm-project/vllm

[MM][CG] Enable encoder CUDA Graph for MiniCPM-V

合并时间: 2026-09-10 14:28

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/42785>

执行摘要

- 一句话: 为 MiniCPM-V 2.5/2.6/4.0 启用编码器 CUDA Graph, 并引入通用捕获轴机制。
- 推荐动作: 本 PR 是一个有价值的性能与架构改进, 值得精读。核心亮点在于引入的“捕获轴”机制, 这是一个解决多模态编码器 CUDA Graph 中输入形状多样性问题的优雅方案, 设计模式具有借鉴意义。实现者对 Idefics2 位置编码的重构也展示了如何使遗留代码适应 CUDA Graph 约束。建议关注以下几点: 1) 捕获轴机制的通用性如何在其他模型上应用; 2) Idefics2 重构对其他模型的影响; 3) MiniCPM-V 4.5 未来支持的可能性。

功能与动机

MiniCPM-V 模型依赖的 ViT 编码器涉及大量小 CUDA 内核的启动, 在生产服务中存在不可忽视的开销。为延续 #38061 中为 Qwen3-VL 引入的编码器 CUDA Graph 框架, 本 PR 旨在将此性能优化扩展到 MiniCPM-V 系列。关键动机在于, MiniCPM-V 的编码器捕获形状不仅取决于 token budget, 还依赖于离散的 patch-grid/ 像素布局层级, 因此需要一个新的“二次捕获轴”机制来区分这些共享同一 token budget 但形状不同的输入, 从而保持调度和内存分配逻辑不变。

实现拆解

1. 引入捕获轴机制: 在 EncoderCudaGraphConfig 数据类中新增 capture_axes 字段, 类型为 tuple[tuple[Hashable, ...], ...]。这允许模型声明其编码器 CUDA Graph 捕获需要额外的维度。EncoderCudaGraphManager 相应地将其 budget_graphs 字典的键类型从 int 扩展为 int | tuple[int, CaptureAxisKeys], 以支持按 (token_budget, axis_keys) 的组合来存储和查找捕获的图。
2. MiniCPM-V 模型集成: 在 vllm/model_executor/models/minicpmv.py 中, 创建了 _MiniCPMVEncoderCudaGraphMixin 类, 它实现了 SupportsEncoderCudaGraph 接口。该 mixin 为 MiniCPM-V 2.5/2.6/4.0 实现了所有必需的方法, 如 get_encoder_cudagraph_config (配置了 capture_axes)、prepare_encoder_cudagraph_capture_inputs (创建包含展平像素值、patch 掩码、tgt_sizes 等的缓冲区) 和 encoder_cudagraph_forward。关键辅助函数 _mcpmv_pack_flat_pixels 负责将图像切片打包成固定大小的缓冲区。
3. 重构 Idefics2 位置编码: 为使 Idefics2VisionEmbeddings.get_position_ids 适应 CUDA Graph 捕获, 将其从基于循环的 CPU 计算重构为完全向量化的 GPU 计算。关键变更是使用 async_tensor_h2d 异步传输 tgt_sizes, 并使用 torch.where 处理填充掩码, 避免了

GPU 同步。

4. 更新接口与契约：在 SupportsEncoderCudaGraph 协议的 `select_encoder_cudagraph_items` 方法文档中，明确要求支持捕获轴的模型必须在返回的字典中通过 `ENCODER_CUDAGRAPH_AXIS_KEYS_KWARG` 键传递解析出的轴键。所有已实现此协议的模型文件（如 `cohere_compass.py`, `qwen2_vl.py` 等）的 `prepare_encoder_cudagraph_capture_inputs` 方法签名均添加了 `axis_keys` 参数以保持兼容。
5. 测试覆盖：在 `tests/models/multimodal/generation/test_vit_cudagraph.py` 中，为 MiniCPM-V 2.5、2.6、4.0 添加了端到端的 CUDA Graph 测试配置，覆盖图像和视频模态，并设置了适合 CI 环境（35GB MIG）的小 token budget 集合以避免 OOM。

关键文件：

- `vllm/model_executor/models/minicpmv.py`（模块 MiniCPM-V 模型；类别 source；类型 core-logic；符号 `get_mm_max_tokens_per_item`, `_mcpmv_tgt_sizes_tensor`, `_mcpmv_normalize_tgt_sizes`, `_mcpmv_pack_flat_pixels`）：核心模型实现文件，包含 MiniCPM-V 编码器 CUDA Graph 的全部逻辑，包括缓冲区准备、前向传播、像素打包以及新的捕获轴配置。
- `vllm/v1/worker/encoder_cudagraph.py`（模块 CUDA Graph 管理器；类别 source；类型 dependency-wiring；符号 `_num_graphs_for_budgets`, `_get_graph_set`, `_capture_budget_graph`）：编码器 CUDA Graph 管理器的核心逻辑，被扩展以支持捕获轴机制，负责图的捕获、查找和回放调度。
- `vllm/model_executor/models/idefics2_vision_model.py`（模块 Idefics2 编码器；类别 source；类型 data-contract）：MiniCPM-V 使用的 Idefics2 视觉编码器的位置编码计算被重构以适应 CUDA Graph 捕获，从 CPU 循环改为 GPU 向量化操作。
- `tests/models/multimodal/generation/test_vit_cudagraph.py`（模块 编码器 CUDA Graph 测试；类别 test；类型 test-coverage；符号 `minicpmv_25_chat_template`, `minicpmv_chat_template`）：测试文件，为 MiniCPM-V 2.5、2.6、4.0 添加了端到端的编码器 CUDA Graph 测试配置。
- `vllm/v1/worker/encoder_cudagraph_defs.py`（模块 数据结构定义；类别 source；类型 data-contract）：定义了编码器 CUDA Graph 的核心数据结构，被扩展以支持捕获轴配置。

关键符号：`_MiniCPMVEncoderCudaGraphMixin.get_encoder_cudagraph_config`,
`_MiniCPMVEncoderCudaGraphMixin.prepare_encoder_cudagraph_capture_inputs`,
`_MiniCPMVEncoderCudaGraphMixin.encoder_cudagraph_forward`,
`_mcpmv_pack_flat_pixels`, `_mcpmv_normalize_tgt_sizes`,
`Idefics2VisionEmbeddings.get_position_ids`, `EncoderCudaGraphManager.capture`,
`EncoderCudaGraphManager._capture_budget_graph`,
`EncoderCudaGraphManager._run_budget_graph`

关键源码片段

`vllm/model_executor/models/minicpmv.py`

核心模型实现文件，包含 MiniCPM-V 编码器 CUDA Graph 的全部逻辑，包括缓冲区准备、前向传播、像素打包以及新的捕获轴配置。

--- MiniCPM-V 编码器 CUDA Graph 核心实现 ---

```
class _MiniCPMVEncoderCudaGraphMixin(MiniCPMVBaseModel, SupportsEncoderCudaGraph):  
    """为 MiniCPM-V 2.5/2.6/4.0 提供编码器 CUDA Graph 支持的 mixin 类。"""
```

```
    # 标记该模型支持编码器 CUDA Graph  
    supports_encoder_cudagraph: ClassVar[Literal[True]] = True
```

```
    def get_encoder_cudagraph_config(self) -> EncoderCudaGraphConfig:
```

```
        """返回编码器 CUDA Graph 配置，包括捕获轴。"""
```

```
        # MiniCPM-V 的捕获轴是 patch-grid 尺寸 (nb_h, nb_w)
```

```
        capture_axes = (  
            (  
                (32, 32),  
                (32, 64),  
                (64, 32),  
                # ... 其他可能的 grid 尺寸  
            ),  
        ),
```

```
    )  
    return EncoderCudaGraphConfig(  
        modalities=["image", "video"],  
        buffer_keys=[  
            _MINICPMV_CUDAGRAPH_BUF_KEY_TGT_SIZES,  
            _MINICPMV_CUDAGRAPH_BUF_KEY_PATCH_MASK,  
        ],  
        out_hidden_size=..., # 模型隐藏层维度  
        capture_axes=capture_axes,  
        # ... 其他配置  
    )
```

```
    def prepare_encoder_cudagraph_capture_inputs(  
        self,
```

```
        token_budget: int,  
        max_batch_size: int,  
        max_frames_per_batch: int,  
        device: torch.device,  
        dtype: torch.dtype,  
        path: str = "default",  
        axis_keys: tuple[Hashable, ...] | None = None,  
    ) -> EncoderCudaGraphCaptureInputs:
```

```
        """为 CUDA Graph 捕获准备输入缓冲区。
```

```
        当提供 axis_keys（来自捕获轴）时，缓冲区大小将根据  
        该 patch-grid 尺寸精确调整，避免填充浪费。
```

```
        """
```

```
        # 根据 axis_keys 计算像素高度、宽度和最大补丁数
```

```

pixel_h, pixel_w = self._mcpmv_patch_grid_pixel_hw(axis_keys)
max_num_patches = self._mcpmv_patch_grid_num_patches(axis_keys)

# 创建固定形状的缓冲区，用于记录到 CUDA Graph 中
values = {
    _MINICPMV_CUDAGRAPH_BUF_KEY_TGT_SIZES: torch.zeros(
        (max_batch_size, 2), device=device, dtype=torch.long
    ),
    # ... 其他缓冲区
}
return EncoderCudaGraphCaptureInputs(values=values)

# ... 其他接口方法

```

vllm/v1/worker/encoder_cudagraph.py

编码器 CUDA Graph 管理器的核心逻辑，被扩展以支持捕获轴机制，负责图的捕获、查找和回放调度。

--- 捕获轴机制在管理器中的集成 ---

```

class EncoderCudaGraphManager:
    def __init__(self, ...):
        # ... 初始化 token_budgets, max_batch_size 等

        # 从模型配置中读取捕获轴
        self._capture_axes: tuple[CaptureAxisKeys, ...] = tuple(
            tuple(axis) for axis in self.config.capture_axes
        )
        if any(len(axis) == 0 for axis in self._capture_axes):
            raise ValueError("Encoder cudagraph capture axes must be non-empty.")

        # 图的查找键类型扩展为 : int (token_budget) 或 tuple[int, CaptureAxisKeys]
        self.budget_graphs: dict[str, dict[BudgetGraphMapKey, BudgetGraphMetadata]] = {}

    def capture(self, graph_pool: Any):
        """为所有配置的路径和 token 预算捕获 CUDA 图。

        如果配置了捕获轴，则为每个预算和每个轴键组合捕获一个图。"""
        for path, budgets in self.path_token_budgets.items():
            for token_budget in sorted(budgets, reverse=True):
                if token_budget == 0:
                    continue
                # 为每个捕获轴键组合生成一个图
                for axis_keys in itertools.product(*self._capture_axes):
                    self._capture_budget_graph(
                        token_budget,
                        path=path,
                        axis_keys=axis_keys,
                    )

```

```

def _capture_budget_graph(self, token_budget: int, path: str = "default",
                          axis_keys: CaptureAxisKeys = ()):
    """为单个 token 预算和轴键组合捕获一个 CUDA 图。"""
    # 构建图的查找键
    graph_map_key: BudgetGraphMapKey = (
        token_budget if not axis_keys else (token_budget, axis_keys)
    )
    # ... 捕获图并存入 graph_set[graph_map_key]

def _run_budget_graph(self, mm_kwargs, token_budget, path, axis_keys=()):
    """执行预算图。"""
    # 构建查找键
    graph_map_key = (
        token_budget if not axis_keys else (token_budget, axis_keys)
    )
    if graph_map_key not in self._get_graph_set(path):
        self.graph_misses += num_items
        return None
    # ... 找到图并执行回放

```

vllm/model_executor/models/idefics2_vision_model.py

MiniCPM-V 使用的 Idefics2 视觉编码器的位置编码计算被重构以适应 CUDA Graph 捕获，从 CPU 循环改为 GPU 向量化操作。

--- Idefics2VisionEmbeddings.get_position_ids 重构 ---

```

class Idefics2VisionEmbeddings(nn.Module):
    def get_position_ids(
        self,
        patch_attention_mask: torch.BoolTensor,
        tgt_sizes: torch.IntTensor | None = None,
    ) -> torch.Tensor:
        batch_size = patch_attention_mask.shape[0]
        device = patch_attention_mask.device
        boundaries = torch.arange(
            1 / self.num_patches_per_side,
            1.0,
            1 / self.num_patches_per_side,
            device=device,
        )

        flat_patch_attention_mask = patch_attention_mask.view(batch_size, -1)
        max_patches = flat_patch_attention_mask.shape[-1]

        if tgt_sizes is not None:
            # tgt_sizes 可能来自 CPU 端的 mm_kwargs; 使用 pinned async
            # 复制以保持与 VLLM_GPU_SYNC_CHECK 的兼容性。
            if tgt_sizes.device != device:
                tgt_sizes = async_tensor_h2d(tgt_sizes, device)

```

```

        nb_patches_h = tgt_sizes[:, 0]
        nb_patches_w = tgt_sizes[:, 1]
    else:
        # ... 回退逻辑

    # 完全向量化的 GPU 计算
    nb_h = nb_patches_h.clamp(min=1).unsqueeze(1)
    nb_w = nb_patches_w.clamp(min=1).unsqueeze(1)
    i = torch.arange(max_patches, device=device)

    # ... 计算 h_idx, w_idx, fractional_h, fractional_w
    bucket_h = torch.bucketize(fractional_h, boundaries, right=True)
    bucket_w = torch.bucketize(fractional_w, boundaries, right=True)
    pos_ids = bucket_h * self.num_patches_per_side + bucket_w

    position_ids = torch.where(flat_patch_attention_mask, pos_ids.to(torch.long), 0)

    target_device = self.position_embedding.weight.device
    if position_ids.device != target_device:
        position_ids = async_tensor_h2d(position_ids, target_device)
    return position_ids

```

评论区精华

Review 讨论主要集中在接口设计和实现细节上：

1. 捕获轴机制设计：审阅者 shen-shanshan 建议将二级捕获轴的启用逻辑下沉到 EncoderCudaGraphConfig 中（如 enable_secondary_capture_axis=True），并让 prepare_encoder_cudagraph_capture_inputs 接口直接接收 secondary_capture_axis_key 参数，从而避免在 EncoderCudaGraphManager 中增加额外的包装方法。最终实现采纳了更通用的 capture_axes 机制。
 2. 接口一致性：审阅者 Isotr0py 探讨了是否支持任意数量的捕获轴，并推动了将实现从固定的“二级轴”泛化为可配置的 capture_axes 元组。这提升了框架的扩展性。
 3. 文档与代码风格：审阅者 shen-shanshan 指出文档表格中可合并同架构的 MiniCPM-V 2.6 和 4.0 行，以保持简洁。
 4. CI 问题解决：讨论中发现并修复了 tgt_sizes 的 GPU 同步阻塞问题（因 #52827 引入的 keep_on_cpu 导致），通过使用 async_tensor_h2d 解决。还修复了因方法返回类型注解意外触发 test_tensor_schema 测试失败的问题。
- 捕获轴机制的接口设计 (design): 采用可配置的 capture_axes 元组，每个轴是一个可能的键值集合。管理器根据预算和轴键的笛卡尔乘积捕获多个图。
 - tgt_sizes 的 GPU 同步阻塞 (performance): 在 minicpmv.py 的 forward 和 idfics2_vision_model.py 的 get_position_ids 中使用 async_tensor_h2d 替代阻塞拷贝。
 - 空缓冲区的数据类型 (correctness): 已修复，使用模型参数的实际 dtype。
 - select_encoder_cudagraph_items 返回格式 (correctness): 通过扩展 _mcpmv_normalize_tgt_sizes 以处理 list[torch.Tensor] 来修复。

风险与影响

- 风险:

1. 回归风险: 对 Idefics2 视觉编码器 (Idefics2VisionEmbeddings.get_position_ids) 的重构改变了其位置编码计算从 CPU 循环到 GPU 向量化操作的路径, 可能影响共享此编码器的其他模型 (如 InternVL、Gemma3 等) 的行为和性能。需要确保所有相关模型在 eager 和 CUDA Graph 模式下均通过测试。
2. 兼容性风险: SupportsEncoderCudaGraph 接口的 prepare_encoder_cudagraph_capture_inputs 方法新增了 axis_keys 参数。所有已实现此接口的模型文件都需要同步更新其签名 (本 PR 已完成), 否则可能导致调用错误。未来新增模型需遵循此契约。
3. 性能风险: 引入捕获轴机制后, 每个 token budget 的图捕获数量可能倍增。代码中已设置 _CAPTURE_GRAPH_COUNT_WARNING_THRESHOLD = 64 进行告警, 并在文档中提醒用户合理设置 capture_axes 以避免内存爆炸。
4. 平台兼容性: async_tensor_h2d 的使用假定存在异步 H2D 传输通道。在非 CUDA/ROCm 平台上可能需要回退到同步传输, 但当前实现依赖于 current_platform。

- 影响:

1. 用户影响: MiniCPM-V 2.5、2.6、4.0 的用户现在可以通过启用 --compilation-config '{ "cudagraph_mm_encoder": true }' 来获得编码器推理的性能提升 (延迟降低, 吞吐增加)。视频输入在 MiniCPM-V 2.6/4.0 上也得到支持。MiniCPM-V 4.5 因动态帧融合不兼容静态图捕获而未包含, 可能成为后续优化点。
2. 系统影响: 为多模态编码器 CUDA Graph 框架引入了一个重要的泛化设计 (捕获轴), 为未来支持更多具有复杂输入形状依赖的模型 (如混合分辨率、动态 patching) 奠定了基础。这增强了 vLLM 在多模态推理优化方面的基础设施。
3. 团队影响: 本 PR 是 tracker #38175 的重要组成部分, 展示了社区协作模式 (原作者 YunzhuLu 与维护者 Isotr0py 合作)。它与其他模型贡献者提供了实现编码器 CUDA Graph 的清晰范例和文档。- 风险标记: 核心编码器逻辑重构, 接口契约变更影响多模型, 捕获轴机制可能增加图捕获数量与内存占用

关联脉络

- PR #38061 [MM][Perf][CG] Support ViT full CUDA graph for Qwen3-VL video inference: 本 PR 是其直接后续, 遵循其引入的编码器 CUDA Graph 工作流程, 并在此基础上引入了捕获轴机制以适应 MiniCPM-V 的复杂输入形状。
- PR #38175 [RFC]: Support ViT Full CUDA Graph (Tracker): 本 PR 是此 RFC 追踪 issue 的一部分 (PR#42785 在其中被标记), 旨在为更多多模态模型启用编码器 CUDA Graph。
- PR #52827 [Core] Introduce keep_on_cpu for tensor metadata: 此 PR 引入了 keep_on_cpu 标志, 导致 tgt_sizes 等元数据在某些路径下驻留 CPU, 从而触发了本 PR 中修复的 GPU 同步阻塞问题。