PR #42796 完整报告

vllm-project/vllm

[MM][CG] Avoid over-padding Qwen2.5-VL encoder cudagraph window metadata

合并时间：2026-05-29 02:22

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42796

执行摘要

- 一句话：优化 Qwen2.5-VL encoder CUDA graph 窗口序列上界，B200 性能提升 3x+
- 推荐动作：该 PR 值得精读，展示了在 CUDA graph replay 中处理变长输入的正确姿势，尤其是 padding_logics 设计模式体现了插件化思想。评审过程中对灵活性与显式性之间的权衡也值得关注。

功能与动机

PR #40830 启用 Qwen2.5-VL 的 encoder CUDA graph 后，在 B200 上使用默认的 FLASH_ATTN 后端出现严重性能退化，原因是窗口序列数上界估计过于保守，导致 FlashAttention 启动大量空计算块。同时 FlashInfer 后端使用不同的 packed cu_seqlens 布局，通用 padding 方式会写错向量偏移部分。

实现拆解

1. 新增 padding 函数：在 qwen2_5_vl.py 中定义 _pad_cumulative_seqlens_buffer（通用）和 _pad_flashinfer_cu_seqlens_buffer（FlashInfer 专用）。后者将 packed 布局的 [Q/K/O offsets, V offsets] 拆为两段独立 padding。
2. 新增窗口序列数上界计算：在 Qwen2_5_VLProcessor 中实现 get_encoder_cudagraph_max_window_seqs，基于视觉融合网格尺寸做几何推导，给出远紧于 token_budget 的上界，避免过度分配。
3. 重构 replay 拷贝逻辑：在 encoder_cudagraph.py 的 _run_budget_graph 中，将硬编码的零填充 + 切片拷贝替换为通过 EncoderCudaGraphConfig.padding_logics 查找模型特定 padding 函数，默认使用 _copy_padded_buffer。
4. 配置插件化：在 encoder_cudagraph_defs.py 的 EncoderCudaGraphConfig 中新增 padding_logics 字典字段，允许模型为每个 buffer key 注册自定义 padding 函数。
5. 模型注册：在 qwen2_5_vl.py 的 get_encoder_cudagraph_config 中，根据 attention 后端类型注册对应的 padding 函数到 padding_logics。 此变更未新增测试文件，但通过已有 benchmark 验证性能。

关键文件：

- vllm/model_executor/models/qwen2_5_vl.py（模块 视觉模型；类别 source；类型 core-logic；符号 _pad_cumulative_seqlens_buffer, _pad_flashinfer_cu_seqlens_buffer, get_encoder_cudagraph_max_window_seqs）：核

心变更，包含两种 padding 函数和窗口序列数上界计算方法，是性能提升的关键。

- vllm/v1/worker/encoder_cudagraph.py（模块 编码器 CG 管理；类别 source；类型 core-logic；符号 _copy_padded_buffer）：通用管理器中引入可插拔 padding 逻辑，通过 padding_logics 代理调用。
- vllm/v1/worker/encoder_cudagraph_defs.py（模块 配置定义；类别 source；类型 dependency-wiring）：定义 EncoderCudaGraphPaddingLogic 类型和 padding_logics 配置字段，实现插件化。

关键符号：_pad_cumulative_seqlens_buffer, _pad_flashinfer_cu_seqlens_buffer, get_encoder_cudagraph_max_window_seqs, _copy_padded_buffer

关键源码片段

vllm/model_executor/models/qwen2_5_vl.py

核心变更，包含两种 padding 函数和窗口序列数上界计算方法，是性能提升的关键。

```
def _pad_cumulative_seqlens_buffer(
    dst: torch.Tensor,
    src: torch.Tensor,
) -> None:
    # 通用 padding：将有效数据拷贝到 dst 头部，
    # 尾部用最后一个有效值填充，使 CUDA graph replay 时
    # padded 位置表示空序列（cu_seqlens 尾部重复）。
    n = src.shape[0]
    dst.zero_()
    dst[:n].copy_(src)
    if n < dst.shape[0]:
        dst[n:] = src[-1]


def _pad_flashinfer_cu_seqlens_buffer(
    dst: torch.Tensor,
    src: torch.Tensor,
) -> None:
    # FlashInfer 使用 packed 布局：[Q/K/O offsets][V offsets]，
    # 两段需要独立 padding。
    src_mid = src.shape[0] // 2
    dst_mid = dst.shape[0] // 2
    assert src_mid <= dst_mid, (
        f"FlashInfer cu_seqlens replay buffer is larger than capture buffer: "
        f"src_section={src_mid}, dst_section={dst_mid}"
    )
    dst.zero_()
    # 处理前半段（Q/K/O 偏移）
    dst[:src_mid].copy_(src[:src_mid])
    if src_mid < dst_mid:
        dst[src_mid:dst_mid] = src[src_mid - 1]
    # 处理后半段（V 偏移）
```

```
        dst[dst_mid : dst_mid + src_mid].copy_(src[src_mid:])
        if dst_mid + src_mid < dst.shape[0]:
            dst[dst_mid + src_mid :] = src[-1]


    def get_encoder_cudagraph_max_window_seqs(
        self,
        token_budget: int,
        max_batch_size: int,
        max_frames_per_batch: int,
    ) -> int:
        # 基于几何形状计算 window 序列数上界，避免用 token_budget 直接作为序列数
        # 导致过度分配。
        vit_merger_window_size = (
            self.window_size // self.spatial_merge_size // self.patch_size
        )
        max_sequence_units = max(max_batch_size, max_frames_per_batch)
        # 最坏情况：一条细条只向一个方向延伸，所需窗口数为
        # ceil(token_budget / window_side)
        max_strip_windows = (
            token_budget + vit_merger_window_size - 1
        ) // vit_merger_window_size
        # 但窗口序列数不可能超过 token_budget，取 min 收紧上界
        return min(token_budget, max_sequence_units + max_strip_windows)
```

vllm/v1/worker/encoder_cudagraph.py

通用管理器中引入可插拔 padding 逻辑，通过 padding_logics 代理调用。

```
    @staticmethod
    def _copy_padded_buffer(
        dst: torch.Tensor,
        src: torch.Tensor,
    ) -> None:
        # 默认 padding：先清零，再拷贝有效数据。
        dst.zero_()
        dst[: src.shape[0]].copy_(src)

    # 在 _run_budget_graph 中调用：
    padding_logic = self.config.padding_logics.get(
        key, self._copy_padded_buffer
    )
    padding_logic(buf, src)
```

评论区精华

- Isotr0py在 Review 中建议将 padding logic 放入 EncoderCudaGraphConfig 中提高灵活性，PR 采纳并新增 padding_logics 字段。

- johncalesp指出基于 key 名称和形状推断 FlashInfer 布局不够安全，建议显式检查 backend；PR 最终通过传递专用 padding 函数回避该问题。
- huanghua1994在 Issue 评论中详细解释了 FlashInfer packed 布局需要独立 padding Q/K/O 与 V 偏移的技术原因。
- Padding logic 设计灵活性 (design): PR 采纳建议，新增 padding_logics 字段，由模型提供专有 padding 函数。
- FlashInfer cu_seqlens 布局识别 (correctness): PR 最终未添加显式检查，但通过模型传递专有 padding 函数绕过了该问题。
- B200 性能退化 (performance): 修复有效，同时 H200 无回归。

风险与影响

- 风险：
  1. 回归风险低：默认 padding 行为（零填充 + 拷贝）保持不变，仅显式注册了特定 key 的 padding 函数，不影响未注册 key 或其他模型。
  2. 硬件特定：优化主要针对 B200 的 FLASH_ATTN 后端，H200 已测试无回归，但其他硬件需验证。
  3. 耦合局限：FlashInfer 的 padding 逻辑紧耦合于 Qwen2.5-VL 模型，若其他模型使用 FlashInfer 后端可能需要复制或抽象公共逻辑。
  4. 测试覆盖：缺乏针对 padding 逻辑的单元测试，仅依赖端到端 benchmark。 - 影响：用户影响：使用 Qwen2.5-VL 并启用 encoder CUDA graph 的用户在 B200 上获得 3.4 倍吞吐提升和 87% TTFT 降低；H100 无变化。 系统影响：引入了可插拔的 padding 机制，未来其他模型在类似场景可直接复用，降低耦合。 团队影响：需要维护 padding_logics 接口文档，并确保新模型正确定义 padding 行为。
- 风险标记：核心路径变更，缺少测试覆盖，硬件特定优化

关联脉络

- PR #42787 [MM][CG] Switch default MM encoder attention to FlashInfer on B200: 该 PR 将 B200 的默认 MM encoder attention 切换到 FlashInfer，两者配合解决性能退化。
- PR #40830 Enable encoder CUDA graph for Qwen2.5-VL: 首次启用 Qwen2.5-VL encoder CUDA graph，但引入 over-padding 问题。