PR #37043 完整报告

sgl-project/sglang

[vlm] fix: preserve per-request vit graph metadata for qwen-vl

合并时间：2026-08-30 21:02

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37043

执行摘要

- 一句话：修复 Qwen-VL ViT CUDA graph 按布局复用与工作区地址绑定
- 推荐动作：值得精读。重点学习两个设计决策：一是 CUDA graph key 必须包含请求级元数据（仅按长度复用是典型的 graph 陷阱）；二是用“退役旧分配 + 按 graph 绑定地址”解决 capture 后地址失效问题。_sequence_layout_key 把同步从查找路径挪到 CPU 预处理的做法，也值得其他 CUDA graph runner 复用。

功能与动机

PR body 明确指出：Equal-token VLM batches can have different image boundaries，仅按总 token 数复用 graph 会重放 stale cu_seqlens；同时 Qwen3-VL 可能在较小 graph 捕获后增长共享 rotary workspace，使旧 graph 保留旧地址而 replay 更新替换分配。两种情况都会把 request-stale vision metadata 送入 attention，属于静默输出错误而非崩溃。

实现拆解

1. graph key 语义扩展（python/sglang/srt/multimodal/vit_cuda_graph_runner.py）：_get_graph_key 从仅返回 x_3d.shape[0] 改为返回 (x_3d.shape[0], attention_layout_key) 元组；新增 _sequence_layout_key 静态方法，把 cu_seqlens / cu_window_seqlens 转为 CPU tuple。调用方不显式传 key 时，runner 内部用两个 seqlens 计算布局，使等 token 数、不同 image 边界的 batch 不再命中同一 graph。
2. rotary workspace 地址绑定：原 _ensure_sin_cos_ws 重构为 _get_sin_cos_ws(graph_key, seq_len, head_dim)，新增 _sin_cos_ws_by_graph 记录每个 graph 绑定的 workspace 切片，新增 _retired_sin_cos_ws 列表挂起旧分配。workspace 增长时旧地址 buffer 不再释放，旧 graph 重放仍写回捕获时的地址，消除“replay 更新替换分配”的数据错位。
3. 模型侧传递布局 key（python/sglang/srt/models/qwen2_5_vl.py、qwen3_vl.py）：Qwen2.5-VL 的 forward_with_cuda_graph 在 CPU 侧通过 grid_thw.tolist() 累计 full_layout，并在 torch.unique_consecutive 之前抽取去重后的 cu_window_layout，显式传入 attention_layout_key；Qwen3-VL 用 tuple(cu_seqlens.tolist()) 构造 key，使 graph 查找路径不触碰 GPU tensor，避免设备同步。
4. 接口契约同步：create_graph / replay / run 签名同步更新，graph_key 类型从 int 改为 Hashable，cu_window_seqlens 改为 Optional[torch.Tensor]，并新增 attention_layout_key 参数；现有 InternVL 等其他调用方不受影响。

5. 测试配套：新增 test/registered/unit/multimodal/test_vit_cuda_graph_metadata_cuda.py，用真实 CUDA graph 验证等 token 不同布局、workspace 增长后小 graph 重放、输出依赖 metadata 三个场景；test_vit_cuda_graph_runner.py 增加 graph key 布局区分与 workspace 地址保持两个 CPU 单元测试。

关键文件：

- python/sglang/srt/multimodal/vit_cuda_graph_runner.py（模块 视觉编码；类别 source；类型 core-logic；符号 _get_sin_cos_ws, _sequence_layout_key, _get_graph_key, create_graph）：核心修复点：graph key 纳入 attention 布局元组、rotary workspace 按 graph 绑定并保留旧分配，所有 capture/replay 路径的签名同步更新。
- python/sglang/srt/models/qwen2_5_vl.py（模块 模型适配；类别 source；类型 data-contract；符号 forward_with_cuda_graph）：Qwen2.5-VL 的 forward_with_cuda_graph 从 CPU 元数据构造 full/window 布局 key 并显式传入 runner，是布局 key 落地到模型侧的关键接入点。
- python/sglang/srt/models/qwen3_vl.py（模块 模型适配；类别 source；类型 data-contract；符号 forward_with_cuda_graph）：Qwen3-VL 的 forward_with_cuda_graph 用 tuple(cu_seqlens.tolist()) 构造 layout key，保证无 window attention 场景下同样避免按长度误复用。
- test/registered/unit/multimodal/test_vit_cuda_graph_metadata_cuda.py（模块 测试；类别 test；类型 test-coverage；符号 test_vit_graph_replays_current_attention_and_position_metadata）：新增真实 CUDA graph 集成测试，覆盖等 token 不同布局、workspace 增长后小 graph 重放、输出依赖当前 metadata 三个核心回归场景。
- test/registered/unit/multimodal/test_vit_cuda_graph_runner.py（模块 测试；类别 test；类型 test-coverage；符号 test_vit_graph_key_includes_full_and_window_attention_boundaries, test_vit_graph_keeps_rotary_workspace_address_after_growth）：补充两个轻量 CPU 单元测试，直接锁定 graph key 布局区分与 workspace 地址保持两个回归点。

关键符号：_get_sin_cos_ws, _sequence_layout_key, _get_graph_key, create_graph, replay, run, forward_with_cuda_graph

关键源码片段

python/sglang/srt/multimodal/vit_cuda_graph_runner.py

核心修复点：graph key 纳入 attention 布局元组、rotary workspace 按 graph 绑定并保留旧分配，所有 capture/replay 路径的签名同步更新。

```
def _get_sin_cos_ws(
    self, graph_key: Hashable, seq_len: int, head_dim: int
) -> Tuple[torch.Tensor, torch.Tensor]:
    # 已经绑定过的 graph 直接返回旧切片，保证 replay 写回的是捕获时的地址
    graph_ws = self._sin_cos_ws_by_graph.get(graph_key)
    if graph_ws is not None:
        return graph_ws

    # 当前 workspace 长度或 head_dim 不足时，分配更大的缓冲
```

```
        needs_new_workspace = self.sin_cos_ws is None or (
            self.sin_cos_ws[0].size(0) < seq_len
            or self.sin_cos_ws[0].size(1) < head_dim
        )
        if needs_new_workspace:
            previous = self.sin_cos_ws
            previous_seq_len = previous[0].size(0) if previous is not None else 0
            previous_head_dim = previous[0].size(1) if previous is not None else 0
            max_shape = max(self.max_context_len or 0, previous_seq_len * 2, seq_len)
            max_head_dim = max(previous_head_dim, head_dim)
            cos_ws = torch.empty(max_shape, max_head_dim, dtype=self.dtype, device=self.device)
            sin_ws = torch.empty(max_shape, max_head_dim, dtype=self.dtype, device=self.device)
            if previous is not None:
                # CUDA graph 捕获后内部保存的是旧地址，旧分配只能挂起、不能释放
                self._retired_sin_cos_ws.append(previous)
            self.sin_cos_ws = (cos_ws, sin_ws)

        # 每个 graph 绑定这份 workspace 的 [0:seq_len, 0:head_dim] 切片
        graph_ws = (
            self.sin_cos_ws[0][:seq_len, :head_dim],
            self.sin_cos_ws[1][:seq_len, :head_dim],
        )
        self._sin_cos_ws_by_graph[graph_key] = graph_ws
        return graph_ws

    @staticmethod
    def _sequence_layout_key(cu_seqlens: Optional[torch.Tensor]) -> Optional[tuple]:
        # GPU tensor 转 CPU tuple，graph 查找路径上不触发设备同步
        if cu_seqlens is None:
            return None
        return tuple(int(value) for value in cu_seqlens.tolist())

    def _get_graph_key(
        self,
        x_3d: torch.Tensor,
        cu_seqlens: torch.Tensor,
        cu_window_seqlens: Optional[torch.Tensor],
        attention_layout_key: Optional[Hashable] = None,
    ) -> Hashable:
        # 只按总 token 数会复用不同 image 边界的 graph，重放出陈旧的 cu_seqlens
        if attention_layout_key is None:
            attention_layout_key = (
                self._sequence_layout_key(cu_seqlens),
                self._sequence_layout_key(cu_window_seqlens),
            )
        return (x_3d.shape[0], attention_layout_key)
```

python/sglang/srt/models/qwen2_5_vl.py

Qwen2.5-VL 的 forward_with_cuda_graph 从 CPU 元数据构造 full/window 布局 key 并显式传入 runner，是布局 key 落地到模型侧的关键接入点。

```
# 从 CPU 侧 grid_thw 顺序累计 token 数，构造 full attention 布局
full_layout = [0, 0]
total_tokens = 0
for temporal, height, width in grid_thw.tolist():
    total_tokens += temporal * height * width
    full_layout.append(total_tokens)

# window 布局在 unique_consecutive 去重之前先抽取出连续值序列，
# 保证与 run 内部实际使用的 cu_window_seqlens 语义一致
cu_window_layout = tuple(
    value
    for index, value in enumerate(cu_window_seqlens)
    if index == 0 or value != cu_window_seqlens[index - 1]
)

return self.cuda_graph_runner.run(
    x=x,
    position_embeddings=position_embeddings,
    cu_seqlens=cu_seqlens,
    cu_window_seqlens=cu_window_seqlens,
    output_indices=reverse_indices,
    # 显式传入布局 key，等 token 数但不同 image 边界的 batch 会被区分开
    attention_layout_key=(tuple(full_layout), cu_window_layout),
)
```

test/registered/unit/multimodal/test_vit_cuda_graph_metadata_cuda.py

新增真实 CUDA graph 集成测试，覆盖等 token 不同布局、workspace 增长后小 graph 重放、输出依赖当前 metadata 三个核心回归场景。

```
def test_vit_graph_replays_current_attention_and_position_metadata():
    runner = ViTCudaGraphRunner(_VisionTower())

    def run(seq_len, boundaries, position):
        x = torch.zeros(seq_len, 1, device="cuda")
        cu_seqlens = torch.tensor(boundaries, dtype=torch.int32, device="cuda")
        positions = torch.full((seq_len, 1), position, device="cuda")
        output = runner.run(x, cu_seqlens, None, (positions, positions))
        torch.cuda.synchronize()
        return output.cpu()

    # 首次捕获 4 token、边界 [0, 2, 4] 的 graph，输出应为 x + boundary + position
    first = run(4, [0, 2, 4], 1)
    # 等 token 数但不同边界，必须命中不同 graph，输出随新边界变化
    different_layout = run(4, [0, 1, 4], 1)
    # 更大的 8 token 请求触发 rotary workspace 增长
    run(8, [0, 8], 7)
```

```
# 小 graph 重放必须仍写回自己捕获时的 workspace 地址，输出携带新 position
small_after_growth = run(4, [0, 2, 4], 5)

torch.testing.assert_close(first, torch.full_like(first, 3))
torch.testing.assert_close(different_layout, torch.full_like(different_layout, 2))
torch.testing.assert_close(small_after_growth, torch.full_like(small_after_growth, 7))
```

评论区精华

该 PR 没有产生技术性 review 评论。唯一一条 issue 评论是作者触发的 /tag-and-rerun-ci 指令，属于 CI 重跑操作；Review 审核与评论均为空。设计权衡主要体现在 PR body 与测试意图中：布局 key 从 CPU 元数据构造以避免 graph 查找路径上的 GPU 同步，以及用退役 workspace 列表保证 CUDA graph 地址稳定。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. graph 缓存膨胀：graph key 从“总 token 数”变为“token 数 + 布局元组”，同长度不同布局的 batch 会各自捕获新 graph，block_input / block_output / block_ws 等缓存数量随布局组合增多，可能带来显存与首次捕获延迟开销。
  2. 退役 workspace 不释放：_retired_sin_cos_ws 永久持有旧 buffer，多次 workspace 增长会阶梯式累积显存，长生命周期服务需关注。
  3. capture/replay 契约依赖：_get_sin_cos_ws 依赖“capture 与 replay 传入相同 embedding 参数”的隐式约定；若某调用方 capture 时不传 position embeddings 而 replay 时传，新分配 workspace 与 graph 捕获地址可能不一致。当前两个 Qwen-VL 调用方行为一致，风险较低。
  4. fallback 路径仍有同步：未显式传 attention_layout_key 的调用方会走 _sequence_layout_key 对 GPU tensor 调用 .tolist()，仍可能引入设备同步；模型侧已通过 CPU 元数据绕过，但 runner 的 fallback 未完全消除该开销。 - 影响：用户侧：修复 Qwen2.5-VL / Qwen3-VL 在启用 CUDA graph（默认）时，多图批次等 token 数但 image 边界不同的场景下偶发静默错误输出的问题。系统侧：graph 缓存从“按长度共享”变为“按长度 + 布局共享”，新布局首次出现时需额外捕获，graph 数量可能增长；rotary workspace 退役机制使显存占用小幅上升。团队侧：ViTCudaGraphRunner 公开方法签名变化，未来新增 VLM 模型需按此契约在 forward_with_cuda_graph 中构造布局 key；新增 CUDA 集成测试挂到 base-b / 1-gpu-large CI 网格。 - 风险标记：核心路径变更，graph 缓存膨胀风险，capture/replay 契约依赖，退役 workspace 不释放

关联脉络

- PR #35588 [Bugfix] Fix full prefill CUDA graph padding and EAGLE capture: 同为 CUDA graph 捕获 / 重放正确性修复，可对照 prefill 与 ViT 两条路径对 shape 与 buffer 地址的处理方式。

- PR #36248 [PP] Support prefill CUDA graph proxy tensors: 涉及 CUDA graph proxy tensor 与 buffer 地址稳定性设计，与本 PR 的 rotary workspace 地址保留问题同源。