PR #50929 完整报告

vllm-project/vllm

[MM][CG] Support ViT full CUDA graph for Kimi-K2.5

合并时间：2026-08-05 01:15

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50929

执行摘要

- 一句话：Kimi-K2.5 ViT 编码器接入完整 CUDA graph 捕获
- 推荐动作：值得精读：这是 vLLM v1 encoder CUDA graph 协议的一份相对完整的模型侧实现范例，特别是 pad_cu_seqlens() 对 varlen attention 行数约束的处理、RoPE 尺寸约束下的 dummy 网格构造，以及用哨兵值表达 eager fallback 的权衡。建议重点对照 Codex 三条 P1 评论理解 CG 与 encoder DP、FlashInfer 之间的交互边界；若生产环境使用 K2.5/K2.6 且会开启 flashinfer 或 mm-encoder-tp-mode data，应等待后续修复或自行回填补齐。

功能与动机

PR body 明确目标是 "Add ViT CG support (#38175) for Kimi K2.5/K2.6"，即补齐 Kimi-K2.5 视觉编码器在 vLLM v1 下的 CUDA graph 支持。在多模态场景中，ViT 前向逐请求执行会产生明显显存与内核启动开销，接入 encoder CUDA graph 后 ViT 前向可复用捕获的图来降低延迟。PR 提供了 8 卡 TP + EP 的 e2e 验证命令，并给出图片描述输出作为通过依据。

实现拆解

1. 协议接入与类型声明：KimiK25ForConditionalGeneration 类新增 SupportsEncoderCudaGraph 接口基类与 supports_encoder_cudagraph: ClassVar[Literal[True]] = True；在 TYPE_CHECKING 分支引入 vllm/v1/worker/encoder_cudagraph_defs 中的 EncoderCudaGraphConfig、EncoderCudaGraphCaptureInputs、EncoderCudaGraphReplayBuffers、EncoderItemSpec 类型，避免运行时循环导入。
2. 捕获配置：get_encoder_cudagraph_config() 声明 modalities=["vision_chunk"]、buffer_keys（pixel_values、pos_embeds、rope_freqs_cis、cu_seqlens、max_seqlen、merge_gather_idx）与 padding_logics；核心是内嵌的 pad_cu_seqlens() 闭包——捕获 buffer 按完整 token budget 分配，真实 batch 较小时用尾部 padding 序列补齐 cu_seqlens，避免 FlashAttn 读到 NaN。
3. 预算与 item 规格：get_encoder_cudagraph_budget_range() 给出 [64, min(max_num_batched_tokens, max_model_len)] 的 token 预算区间；get_encoder_cudagraph_item_specs() 按 grid_thws 将图像拆为独立 item，t > 1 的视频 chunk 用 output_tokens=2**30 哨兵强制 manager 回退 eager。

4. 捕获与回放输入构建：prepare_encoder_cudagraph_capture_inputs() 依据 RoPE 2D 预计算尺寸约束拆分 dummy 图像网格，调用 vision_tower.prepare_encoder_cudagraph_metadata() 生成 cu_seqlens 等元数据，并通过 _encoder_cudagraph_pad_totals[cu_seqlens.data_ptr()] 登记各捕获 buffer 的 patch 总数；prepare_encoder_cudagraph_replay_buffers() 用真实 mm_kwargs 生成回放 buffer。
5. 前向与选择逻辑：select_encoder_cudagraph_items() 按 patch 累计区间从 pixel_values 中切出选中的 item，encoder_cudagraph_forward() 消费捕获输入并调用 ViT 前向（vision_tower_forward 链路），剩余 kwargs 透传给 projector。
6. 测试与配置配套：本 PR 未新增自动化测试文件，仅提供手动 e2e 命令（VLLM_USE_V2_MODEL_RUNNER=0 + --compilation-config '{"cudagraph_mm_encoder": true, ...}'）；配置入口复用已有的 cudagraph_mm_encoder 编译开关，未引入新配置键。

关键文件：

- vllm/model_executor/models/kimi_k25.py（模块 模型实现；类别 source；类型 data-contract；符号 _encoder_cudagraph_pad_totals, get_encoder_cudagraph_config, pad_cu_seqlens, get_encoder_cudagraph_budget_range）：唯一变更文件，为 KimiK25ForConditionalGeneration 完整实现 SupportsEncoderCudaGraph 协议，包含捕获配置、预算、item 规格、捕获 / 回放输入构造与前向入口，是本次功能的核心载体。

关键符号：_encoder_cudagraph_pad_totals, get_encoder_cudagraph_config, pad_cu_seqlens, get_encoder_cudagraph_budget_range, _get_grid_thw_list, get_encoder_cudagraph_item_specs, select_encoder_cudagraph_items, prepare_encoder_cudagraph_capture_inputs

关键源码片段

vllm/model_executor/models/kimi_k25.py

唯一变更文件，为 KimiK25ForConditionalGeneration 完整实现 SupportsEncoderCudaGraph 协议，包含捕获配置、预算、item 规格、捕获 / 回放输入构造与前向入口，是本次功能的核心载体。

```
# 编码器 CUDA graph 配置：声明参与捕获的 buffer 键与 padding 逻辑。 defget_encoder_cudagraph_config(self) -> "EncoderCudaGraphConfig": fromvllm.v1.worker.encoder_cudagraph_defsimport EncoderCudaGraphConfig pad_totals = self._encoder_cudagraph_pad_totals defpad_cu_seqlens(dst: torch.Tensor, src: torch.Tensor) -> None: # varlen attention 要求 cu_seqlens[-1] 等于实际传入的行数。 # 捕获 buffer 按完整 token budget 分配，真实 batch 更小时需要用 # 一条尾部 padding 序列补齐，否则会出现未定义行为并让 FlashAttn # 读到 NaN。 total = pad_totals.get(dst.data_ptr()) n = min(src.shape[0], dst.shape[0]) dst[:n].copy_(src[:n]) dst[n:] = total if total is not None else src[-1] return EncoderCudaGraphConfig( # 只针对图像模态（vision_chunk）启用；视频 chunk 走 eager 回退。 modalities=["vision_chunk"], buffer_keys=[
```

"pixel_values", "pos_embeds", "rope_freqs_cis", " cu_seqlens", "max_seqlen", "merge_gather_idx",], out_hidden_size=self.config.text_config.hidden_size, padding_logics={"cu_seqlens": pad_cu_seqlens},) # 构造捕获用 dummy 输入，并登记每个捕获 buffer 的真实行数（按 cu_seqlens 指针）。 defprepare_encoder_cudagraph_capture_inputs(self, token_budget, max_batch_size, max_frames_per_batch, device, dtype, path="default",) -> "EncoderCudaGraphCaptureInputs": kh, kw = self.vision_tower.merge_kernel_size # 单个 dummy item 的输出 token 数，向上取整使总数覆盖预算。 per_item_out = (token_budget + max_batch_size - 1) // max_batch_size # 图像网格必须落在 RoPE 2D 预计算的最大尺寸内。 rope = self.vision_tower.encoder.rope_2d max_wo = rope.max_width // kw wo = min(per_item_out, max_wo) ho = (per_item_out + wo - 1) // wo assert ho * kh <= rope.max_height, (f"per_item_out={per_item_out}exceeds RoPE grid capacity " f"(max{(rope.max_height//kh)*(rope.max_width//kw)}tokens)") grid_thw_list = [[1, ho * kh, wo * kw] for _ in range(max_batch_size)] ps = self.vision_tower.patch_size if isinstance(ps, int): ps = (ps, ps) total_patches = max_batch_size * ho * kh * wo * kw dummy_pixel_values = torch.zeros(total_patches, 3, ps[0], ps[1], device=device, dtype=dtype,) # max_batch_size + 1 预留一个 cu_seqlens 槽位，回放时可追加 padding 序列； # max_seqlen_override 覆盖最坏情况：单个 item 吃掉全部预算。 metadata = self.vision_tower.prepare_encoder_cudagraph_metadata(grid_thw_list, max_batch_size=max_batch_size + 1, max_seqlen_override=token_budget * kh * kw, device=device,) values: dict[str, torch.Tensor] = {"pixel_values": dummy_pixel_values} values.update({k: v for k, v in metadata.items() if v is not None}) cu_seqlens = values.get("cu_seqlens") if cu_seqlens is not None: # 记录该 buffer 的 patch 总数，供 pad_cu_seqlens 回放时补齐。 self._encoder_cudagraph_pad_totals[cu_seqlens.data_ptr()] = total_patches return EncoderCudaGraphCaptureInputs(values=values)

评论区精华

合并者 Isotr0py 触发 @codex review，维护者对最终版直接 APPROVED（"Thanks!"），但 review 中 Codex 提出了三条 P1 级边界风险，均针对 kimi_k25.py 新增的 encoder CG 协议实现：

- FlashInfer 下 cu_seqlens padding 失效（:512）：Codex 指出 --mm-encoder-attn-backend flashinfer 时 cu_seqlens 是两段拼接、且偏移按 per-rank hidden size 缩放，pad_cu_seqlens() 的连续 copy + 尾部 padding 会让第二段错位，应参考 Qwen encoder CG 路径做 section-wise padding。
- 视频哨兵在 encoder DP 下 OOM（:561）：t > 1 时的 output_tokens=2**30 哨兵会被 EncoderCudaGraphManager._dp_shard()/_dp_gather() 当作真实输出长度，_dp_gather() 会尝试分配并 all-gather (2**30, hidden_size) 张量导致 OOM，需要在不替换真实 output token 数的前提下表达 eager-fallback 资格。

- *eager* 回退二次分片可能 hang（:717）：*encoder* DP 下若走 *eager*，manager 已在各 rank 选出不同的本地子 batch，而 _process_media_input() 内部会调用 run_dp_sharded_mrope_vision_model() 再次基于全局 batch 分片与 all-gather，可能造成 rank 间 *collective* 大小不一致而 hang。 三条评论在合并时均未见作者回复，属未闭环的已知边界问题；主路径（图像 + TP/EP、默认 flash-attn）由维护者确认可合入。
- FlashInfer backend 下 *cu_seqlens* padding 失效 (correctness): 未见作者回复，PR 仍被合并；该 padding 逻辑仅对默认（非 flashinfer）编码器注意力后端成立，属于已知兼容性缺口。
- video 哨兵在 *encoder* 数据并行下导致 OOM (correctness): 未见作者回复，合并时未解决；*eager* fallback 资格不应通过替换真实 output token 数来表达。
- *eager* 回退在 manager DP 分片后二次分片可能 hang (correctness): 未见作者回复，合并时未解决；*eager* 协议方法应直接在已分片的本地 mm_kwargs 上运行 vision tower/projector。

风险与影响

- 风险：
  1. FlashInfer 兼容性风险（kimi_k25.py:512）：pad_cu_seqlens() 假设 *cu_seqlens* 是普通累积长度向量，而 FlashInfer 编码器路径返回两段拼接且按 rank 缩放的 offsets，启用 --mm-encoder-attn-backend flashinfer 时回放会产出非法偏移。
  2. *encoder* 数据并行 OOM 风险（kimi_k25.py:561）：视频 chunk 的 2**30 哨兵在 --mm-encoder-tp-mode *data* 下会被 _dp_gather() 当作真实长度，直接导致显存申请失控，请求级 OOM。
  3. *eager* 回退死锁 / 挂起风险（kimi_k25.py:717）：DP + *eager* fallback 组合下存在二次分片与重复 all-gather 的可能，概率性 hang 且难以排查。
  4. 测试覆盖缺失：无自动化测试文件，三个 P1 场景全靠人工 *e2e* 验证，回归时缺乏护栏。
  5. 平台覆盖单一：PR 仅在 NVIDIA 上验证，未覆盖 ROCm/XPU 等平台；不过 *encoder* CG 机制本身是 v1 通用协议，平台风险相对可控。 - 影响：对用户：启用 *cudagraph_mm_encoder* 后，Kimi-K2.5/K2.6 的图片请求 ViT 前向可复用 CUDA graph，降低 *encoder* 延迟与启动开销；视频 chunk 请求自动回退 *eager*，行为不受影响。对系统：改动限定在 kimi_k25.py 单模型文件，通过 SupportsEncoderCudaGraph 协议接入 v1 *encoder* CUDA graph 管理器，不触及通用调度 / 捕获路径；但对使用 --mm-encoder-attn-backend flashinfer 或 --mm-encoder-tp-mode *data* 的 K2.5 用户存在上述边界风险。对团队：该 PR 是 *encoder* CG 协议在 kimi 模型上的首个完整实现，后续其他模型可参考同一模式接入；同时暴露了协议在 DP 与 FlashInfer 下的公共设计缺口，值得在 manager 层收敛。 - 风险标记：缺少自动化测试，FlashInfer 适配缺口，*encoder* DP 哨兵 OOM 风险，video 回退路径未验证，Codex P1 未闭环

关联脉络

- 暂无明显关联 PR