PR #27096 完整报告

sgl-project/sglang

[diffusion] Cosmos3 fused qknorm rope

合并时间：2026-06-06 09:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27096

执行摘要

- 一句话：融合 QK-norm 与 RoPE，Cosmos3 推理加速 4 倍
- 推荐动作：此 PR 值得精读，尤其推荐给以下读者：
  - 关注文生视频模型推理性能优化
  - 想了解如何将特定模型组件（如 Qwen3 half-split RoPE）映射到通用融合 kernel
  - 需要学习 GQA 场景下 partial rope 的 triton 实现模式
  - 研究 DiT 架构注意力层加速的工程师

功能与动机

Cosmos3 的注意力层同时应用了 Q/K RMSNorm 和 Qwen3 风格的 half-split mRoPE。先前这两个操作为分离的 kernel 调用，存在较大 kernel launch 和带宽开销。本 PR 复用 sglang 已有 fused QK-norm+RoPE 内核，通过封装适配 Qwen3 的 half-split 约定，将两步合并为一步，大幅减少耗时。

实现拆解

实现分为以下步骤：

1. 封装 Qwen3 风格 RoPE 函数（cosmos3video.py）：新增 _apply_qwen3_qk_norm_rope（调用融合的 apply_qk_norm_rope）、_apply_qwen3_rope_from_cache（基于预计算 cos/sin 做 half-split 旋转）和 _apply_qwen3_qk_norm_rope_split（先 norm 再单独 rope，作为回退路径）。注意力层 forward 根据计算模式选择融合或分离路径。
2. 扩展 Qwen3VLTextRotaryEmbedding（mrope.py）：提取 _normalize_position_ids 和 _compute_interleaved_freqs 方法重构 forward 逻辑；新增 build_rope_cache_inputs 方法，直接生成可用于 fused kernel 的连续 cos/sin 缓存张量和位置索引，避免在每次注意力调用时重复计算频率。
3. 增强 apply_flashinfer_rope_qk_inplace（utils.py）：支持 q 和 k 头数不相等（GQA 场景）；增加设备一致性检查和 rope_dim 校验；新增局部函数 apply_rope_prefix，只对部分维度（前 rope_dim）应用旋转，其余维度保持原样，满足 half-split rope 仅作用于前半部分的需求。
4. 加固 apply_qk_norm_rope（layernorm.py）：添加 cos_sin_cache 类型、维度、设备一致性检查；对 positions 添加显式 device/dtype 转换；放宽形状检查以支持 GQA 中的不同头数。

5. vocoder_loader.py 微小修复：类名空值时使用 pipeline 配置中指定的架构名作为默认值。
测试与基准：提供了 kernel 微基准（4x 加速）和 e2e 性能数据（端到端 -3.27%），未新增单元测试。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/cosmos3video.py（模块 DiT 模型；类别 source；类型 data-contract；符号 _apply_qwen3_qk_norm_rope, _apply_qwen3_rope_from_cache, _apply_qwen3_qk_norm_rope_split, _compute_rope_freqs）：模型主文件，新增 Qwen3 风格 RoPE 封装函数，修改注意力层 forward，串联融合 norm+rope 或分离路径。
- python/sglang/multimodal_gen/runtime/layers/rotary_embedding/mrope.py（模块 旋转编码；类别 source；类型 core-logic；符号 forward, _normalize_position_ids, _compute_interleaved_freqs, build_rope_cache_inputs）：Qwen3 旋转嵌入类重构，提取公共子方法并新增 build_rope_cache_inputs 用于 fused kernel 缓存生成。
- python/sglang/multimodal_gen/runtime/layers/rotary_embedding/utils.py（模块 旋转编码工具；类别 source；类型 core-logic；符号 apply_rope_prefix）：增强 apply_flashinfer_rope_qk_inplace，支持 q_heads != k_heads（GQA），引入 apply_rope_prefix 局部函数处理部分维度的旋转。
- python/sglang/multimodal_gen/runtime/layers/layernorm.py（模块 归一化层；类别 source；类型 core-logic）：加固 apply_qk_norm_rope 的输入校验和设备一致性检查。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/vocoder_loader.py（模块 加载器；类别 source；类型 core-logic）：微小修复：class_name 为 None 时使用 pipeline 配置中的架构名作为默认值。

关键符号：_apply_qwen3_qk_norm_rope, _apply_qwen3_rope_from_cache, _apply_qwen3_qk_norm_rope_split, build_rope_cache_inputs, apply_rope_prefix

关键源码片段

python/sglang/multimodal_gen/runtime/models/dits/cosmos3video.py

模型主文件，新增 Qwen3 风格 RoPE 封装函数，修改注意力层 forward，串联融合 norm+rope 或分离路径。

```
def _apply_qwen3_qk_norm_rope(
    q: torch.Tensor,
    k: torch.Tensor,
    q_norm: RMSNorm,
    k_norm: RMSNorm,
    head_dim: int,
    cos_sin_cache: torch.Tensor,
    rope_cache_positions: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor]:
    # 调用融合的 apply_qk_norm_rope，is_neox=True 对应 half-split
    return apply_qk_norm_rope(
        q=q.contiguous(),
        k=k.contiguous(),
```

```
            q_norm=q_norm,
            k_norm=k_norm,
            head_dim=head_dim,
            cos_sin_cache=cos_sin_cache,
            is_neox=True,
            positions=rope_cache_positions,
        )


    def _apply_qwen3_rope_from_cache(
        q: torch.Tensor, k: torch.Tensor, cos_sin_cache: torch.Tensor
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # 直接从预计算的 cos/sin 缓存应用 half-split RoPE
        batch_size, seq_len = q.shape[:2]
        half = q.shape[-1] // 2
        cos = cos_sin_cache[:, :half].view(batch_size, seq_len, 1, half).to(q.dtype)
        sin = cos_sin_cache[:, half:].view(batch_size, seq_len, 1, half).to(q.dtype)
        q1, q2 = q[..., :half], q[..., half:]
        k1, k2 = k[..., :half], k[..., half:]
        q_out, k_out = torch.empty_like(q), torch.empty_like(k)
        q_out[..., :half] = q1 * cos - q2 * sin
        q_out[..., half:] = q2 * cos + q1 * sin
        k_out[..., :half] = k1 * cos - k2 * sin
        k_out[..., half:] = k2 * cos + k1 * sin
        return q_out, k_out


    def _apply_qwen3_qk_norm_rope_split(
        q: torch.Tensor,
        k: torch.Tensor,
        q_norm: RMSNorm,
        k_norm: RMSNorm,
        head_dim: int,
        cos_sin_cache: torch.Tensor,
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # 分离路径：先 norm，再应用 rope
        q, k = apply_qk_norm(q.contiguous(), k.contiguous(), q_norm, k_norm, head_dim)
        return _apply_qwen3_rope_from_cache(q, k, cos_sin_cache)
```

python/sglang/multimodal_gen/runtime/layers/rotary_embedding/mrope.py

Qwen3 旋转嵌入类重构，提取公共子方法并新增 build_rope_cache_inputs 用于 fused kernel 缓存生成。

```
    @torch.no_grad()
    def build_rope_cache_inputs(
        self, position_ids: torch.Tensor, *, cache_dtype: torch.dtype | None = None
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # 计算 interleaved mRoPE 频率，然后拼成 cos/sin 缓存
        freqs = self._compute_interleaved_freqs(position_ids)
```

```
    cos = freqs.cos() * self.attention_scaling
    sin = freqs.sin() * self.attention_scaling
    # 若指定 cache_dtype，先转换再转回 float 保持精度
    if cache_dtype is not None and cache_dtype != torch.float32:
        cos = cos.to(cache_dtype).float()
        sin = sin.to(cache_dtype).float()
    # 拼接为 [total_positions, head_dim] 的连续缓存
    cos_sin_cache = torch.cat((cos, sin), dim=-1).reshape(-1, self.head_dim)
    cos_sin_cache = cos_sin_cache.contiguous()
    cache_positions = torch.arange(
        cos_sin_cache.shape[0], device=cos_sin_cache.device, dtype=torch.long
    )
    return cos_sin_cache, cache_positions
```

python/sglang/multimodal_gen/runtime/layers/rotary_embedding/utils.py

增强 apply_flashinfer_rope_qk_inplace，支持 q_heads != k_heads（GQA），引入 apply_rope_prefix 局部函数处理部分维度的旋转。

```
def apply_rope_prefix(x: torch.Tensor, num_heads: int) -> torch.Tensor:
    # 将 x 展平为 [bsz*seqlen, num_heads, d]
    x_flat = x.reshape(bsz * seqlen, num_heads, d)
    # 仅对前 rope_dim 维应用旋转
    x_rot = x_flat[..., :rope_dim]
    out_rot = torch.empty_like(x_rot)
    cos_b = cos.unsqueeze(-2) # [bsz*seqlen, 1, half_size]
    sin_b = sin.unsqueeze(-2)
    if is_neox:
        # half-split 风格：平分维度
        x1, x2 = torch.chunk(x_rot, 2, dim=-1)
        out_rot[..., :half_size] = x1 * cos_b - x2 * sin_b
        out_rot[..., half_size:] = x2 * cos_b + x1 * sin_b
    else:
        # 交替风格
        x1 = x_rot[..., ::2]
        x2 = x_rot[..., 1::2]
        out_rot[..., ::2] = x1 * cos_b - x2 * sin_b
        out_rot[..., 1::2] = x2 * cos_b + x1 * sin_b
    if rope_dim == d:
        return out_rot.view(bsz, seqlen, num_heads, d)
    # 仅替换前 rope_dim 维，后半部分保持不变
    out = x_flat.clone()
    out[..., :rope_dim] = out_rot
    return out.view(bsz, seqlen, num_heads, d)
```

评论区精华

PR 评审期间，reviewer (mickqian) 在 Issue 评论中提出确认新实现与官方实现的接近程度："could you help confirm that the new implementation is closer to official?"（#27096#issuecomment-...）。作者未直接回复，但后续 CI 中的精度测试通过并获 approval

，说明差异在可接受范围内。此外，多次 rerun CI 表明部分测试失败为 flaky，与本次变更无关。

- 新实现与官方对齐的确认 (correctness): 作者未直接回复，但后续 CI 中精度测试通过且 PR 获 approval，表明差异在可接受范围内。

风险与影响

- 风险:
  1. 精度风险：融合 norm+rope 虽然复用了已有 kernel，但 Qwen3 half-split 的角度计算和位置编码顺序必须与原始分离实现完全对齐。若存在细微差异，可能影响视频生成质量（如画面连贯性）。PR 提供了视觉对比结果，但未用数值指标（如 PSNR）量化。
  2. 非CUDA回退路径：新增的apply_rope_prefix逻辑在FlashInfer不可用时（如AMD、CPU）被激活。该路径虽经过重构但未在非 CUDA 设备上测试，可能存在数值或性能退化。
  3. GQA 支持的不完全兼容：apply_flashinfer_rope_qk_inplace 中当 q_heads != k_heads 时强制走 Triton 回退，不再调用 FlashInfer 原生实现，可能丢失原生的性能优势。
  4. 配置项依赖：融合开关通过环境变量 SGLANG_ENABLE_FUSED_QKNORM_ROPE 控制，默认开启。如果用户显式关闭，将使用分离路径，性能下降但不影响正确性。 - 影响: 用户影响：Cosmos3 模型推理速度提升约 3-4%，无功能变化，无需修改配置文件。系统影响：仅影响 sglang/multimodal_gen 模块中的 Cosmos3 相关代码，其他模型（如 Qwen2-VL、Ideogram）不受影响。团队影响：展示了复用融合 kernel 的技术路径，为未来其他模型类似优化提供参考。 - 风险标记：精度敏感，非 CUDA 回退未测试，依赖融合 kernel CUDA 兼容性

关联脉络

- 暂无明显关联 PR