PR #35698 完整报告

sgl-project/sglang

[diffusion] Fuse LTX-2.5 decoder 3D RoPE

合并时间：2026-08-21 10:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35698

执行摘要

- 一句话：融合 LTX-2.5 decoder 3D RoPE，单 kernel 旋转 Q/K 并缓存表格，整网提速约 17%
- 推荐动作：值得精读。这个 PR 展示了在不改变数值语义的前提下做 kernel 融合的完整套路：先构造共享的 compact 表、再写一次性旋转 Q/K 的 CUDA kernel、用预检函数拦截一切不支持输入、用 bit-exact gate 在首次调用时自动验证并永久回退。对比数据（10 倍级加速 + 输出 SHA256 一致 + 显存不变）是很有说服力的验收标准。建议重点看 ltx_2_5_diffusion_decoder.py 的 _tables/_apply_rope/forward_pair 与 ltx25_decoder_rope.cuh 的 pair 旋转循环，理解 "eager 顺序被严格保留 " 在 kernel 里是如何体现的（__fmul_rn/__fsub_rn 组合、避免 FMA）。唯一需要留意的是 gate 与全局缓存的生命周期管理，可作为后续讨论点。

功能与动机

LTX-2.5 decoder 有 24 个 attention block，每个块都要对 (T,H,W) 网格做 3D RoPE；原来的 eager 实现在每次 Q/K 旋转时独立重建 cos/sin 表，并产生大量 kernel launch 和中间张量。PR body 给出的数据是 paired RoPE 在 stage 0 / stage 4 tile / stage 5 tile 上分别有 9.87x / 10.32x / 10.85x 的提升空间，而 decoder 整体中位延迟为 5379.54 ms → 4463.80 ms（-17.0%）。性能之外，作者特别强调要保持 eager fp32 运算顺序，使 JIT 结果与 baseline 输出 SHA256 完全一致（b3a693021e1251d69847903a772e67dca4046425c576c60ce87bfee6689a6a68），这是该融合能够安全落地的核心约束。

实现拆解

实现分五步完成：

1. RoPE 表构建改为可缓存的表格生成（ltx_2_5_diffusion_decoder.py）：把原来 _inv_freqs + 每次旋转现场算角度的逻辑，拆成 _axis_tables(length, dim, device) 直接产出 (cos, sin) 表；_tables(hidden_states) 以 (num_frames, height, width, rope_dim_split, base, device) 为 key 查 _ROPE_TABLE_CACHE，命中则复用，否则生成后写入，缓存上限 _ROPE_TABLE_CACHE_MAX = 16，超限采用简单的先入先出淘汰。表格是纯函数且体积很小，所有 decoder block 在同一网格下共享，避免 24 个块各自重复构造。
2. 保留 eager 运算顺序的 _apply_rope：T/H/W 三个轴各自取 chunk、reshape 成 pair、在 float32 下做 even*cos ± odd*sin 后再转回原 dtype，并在最后 torch.cat 拼回。这个逐

位不变的 eager 表达式既作 CPU/fallback 参考实现，也是 JIT 快速路径 bit-exact 校验的基准。

3. 新增 fused JIT kernel 入口（python/sglang/kernels/ops/diffusion/rope/ltx25_decoder_rope_jit.py）：通过 register_custom_op 注册 diffusion_ltx25_decoder_rope，内部用 load_jit 编译 diffusion/ltx25_decoder_rope.cuh；fused_ltx25_decoder_rope 一次 launch 同时写 Q_out 和 K_out；can_use_ltx25_decoder_rope 做形状、dtype、连续性、指针 4 字节对齐、表格形状 / 设备一致等全面预检；_fake_impl 服务于 torch.compile 等元数据流通场景。
4. 接入解码器并加 gate：模块新增 BitExactFusionGate("LTX-2.5 decoder fused RoPE") 实例 _LTX25_DECODER_ROPE，forward_pair 在预检通过时先走 JIT，并在"首次遇到该配置"时同 eager 做逐位比较；验证通过则信任快路径，不通过则永久回退 eager。forward 被改写为 forward_pair 的薄封装，保持原有调用语义不变。
5. 测试、基准与文档配套：新增 CUDA 单测 test_ltx25_decoder_rope.py（bit-exact、预检拒绝非法输入、指针别名断言）、基准 bench_ltx25_decoder_rope.py（stage0/4/5 三档几何，CI 跑 stage5_tile）、CPU 回退 / 表格共享单测 test_ltx2_5_config.py；导出 sglang.kernels.ops.diffusion 新符号，并在 fused_kernels.mdx、README.md 记录用法与注意事项。

关键文件：

- python/sglang/multimodal_gen/runtime/models/decoders/ltx_2_5_diffusion_decoder.py（模块 扩散解码；类别 source；类型 core-logic；符号 _axis_tables, _tables, _apply_rope, forward）：LTX-2.5 decoder 的 RoPE 主实现：新增跨 block 共享的 cos/sin 表缓存、bit-exact 的 eager 参考路径，以及接入 fused kernel 的 forward_pair 入口，是本 PR 的核心数据契约变更点。
- python/sglang/kernels/ops/diffusion/rope/ltx25_decoder_rope_jit.py（模块 JIT 内核；类别 source；类型 infrastructure；符号 _jit_ltx25_decoder_rope_module, _fake_impl, fused_ltx25_decoder_rope, can_use_ltx25_decoder_rope）：新增 JIT kernel 的 Python 封装，注册 custom op、定义输入预检函数、声明 fake impl，是快速路径能否被安全启用的关键边界。
- python/sglang/kernels/jit/csrc/diffusion/ltx25_decoder_rope.cuh（模块 CUDA 内核；类别 other；类型 core-logic；符号 rotate_pair, ltx25_decoder_rope_kernel）：新增 CUDA kernel 本体，呈现如何用一次 launch 完成 Q/K 的旋转、如何用 SGLANG_LDG 读取表格以及如何保持 eager 运算顺序。
- test/registered/kernels/ops/diffusion/test_ltx25_decoder_rope.py（模块 内核测试；类别 test；类型 test-coverage；符号 make_tables, eager_rope, test_ltx25_decoder_rope_is_bit_exact, test_ltx25_decoder_rope_predicate_rejects_unsupported_inputs）：CUDA 单测的核心验证：断言 fused 输出与 eager 逐位相同、输出不是输入原地别名、预检会拒绝各种非法输入，是快速路径安全性的第一道防线。
- test/registered/kernels/benchmark/diffusion/bench_ltx25_decoder_rope.py（模块 基准测试；类别 test；类型 test-coverage；符号 Case, make_tables, eager_rope, benchmark）：可复现的 paired RoPE 基准，按 stage0/4/5 三档 shape 对比 eager 与 JIT，是 PR 声称 10 倍级加速的直接证据来源，且已接入 CI。

- python/sglang/multimodal_gen/test/unit/test_ltx2_5_config.py（模块 配置测试；类别 test；类型 test-coverage；符号 test_rotary_pair_cpu_fallback_matches_original_expression, test_rotary_tables_are_shared_across_decoder_blocks）：补充 CPU 回退路径与表格共享的单测，确保无 CUDA 环境或 JIT 不可用时 forward_pair 仍与原实现完全一致，并验证缓存确实跨 decoder block 共享。
- python/sglang/kernels/ops/diffusion/__init__.py（模块 内核导出；类别 source；类型 infrastructure）：导出新加入的 fusion API 与 BitExactFusionGate，是上层 decoder 模块能够 import 上述符号的接线点。
- docs/docs/sglang-diffusion/fused_kernels.mdx（模块 文档；类别 docs；类型 documentation）：向用户说明 fused LTX-2.5 decoder RoPE 的存在与使用约束，属于对外可观察行为的文档化。
- python/sglang/kernels/ops/diffusion/README.md（模块 文档；类别 docs；类型 documentation）：维护 kernel 目录自身的维护者文档，提醒新 kernel 的注册方式。

关键符号：_axis_tables, _tables, _apply_rope, forward_pair, fused_ltx25_decoder_rope, can_use_ltx25_decoder_rope, _jit_ltx25_decoder_rope_module, rotate_pair, ltx25_decoder_rope_kernel

关键源码片段

python/sglang/kernels/ops/diffusion/rope/ltx25_decoder_rope_jit.py

新增 JIT kernel 的 Python 封装，注册 custom op、定义输入预检函数、声明 fake impl，是快速路径能否被安全启用的关键边界。

```
# python/sglang/kernels/ops/diffusion/rope/ltx25_decoder_rope_jit.py

@cache_once
def _jit_ltx25_decoder_rope_module(dtype: torch.dtype) -> Module:
    """按 dtype 缓存编译结果，避免每个形状都重复触发 JIT。"""
    if dtype is not torch.bfloat16:
        # 当前 kernel 只支持 BF16；其他 dtype 在预检阶段就会被拦截
        raise RuntimeError(f"Unsupported ltx25_decoder_rope dtype: {dtype}")
    args = make_cpp_args(dtype)
    return load_jit(
        "diffusion_ltx25_decoder_rope",
        *args,
        cuda_files=["diffusion/ltx25_decoder_rope.cuh"],
        cuda_wrappers=[
            (
                "ltx25_decoder_rope",
                f"ltx25_decoder_rope::LTX25DecoderRopeKernel<{args}>::run",
            ),
        ],
    )


def fused_ltx25_decoder_rope(q, k, cos_t, sin_t, cos_h, sin_h, cos_w, sin_w, dim_t, dim_h):
```

```
    """应用配对的 LTX-2.5 decoder RoPE，一次 kernel launch 同时旋转 Q 和 K。"""
    q_out = torch.empty_like(q)
    k_out = torch.empty_like(k)
    module = _jit_ltx25_decoder_rope_module(q.dtype)
    # 所有张量都 flatten 后交给 kernel，网格 / 线程划分在设备端完成
    module.ltx25_decoder_rope(
        q_out.view(-1), k_out.view(-1),
        q.view(-1), k.view(-1),
        cos_t.view(-1), sin_t.view(-1),
        cos_h.view(-1), sin_h.view(-1),
        cos_w.view(-1), sin_w.view(-1),
        q.shape[0], q.shape[1], q.shape[2], q.shape[3], q.shape[4], q.shape[5],
        dim_t, dim_h,
    )
    return q_out, k_out


def can_use_ltx25_decoder_rope(q, k, tables, dim_split) -> bool:
    """预检：任何一项不满足都直接回退 eager，避免 JIT 失败或非法内存访问。

    检查维度数、6D 形状一致、BF16 dtype、CUDA 设备一致、连续性与
    4 字节对齐，以及每个轴的 cos/sin 表形状与 dim_split 匹配。
    """
    if (
        q.dim() != 6
        or len(tables) != 3
        or any(len(pair) != 2 for pair in tables)
        or len(dim_split) != 3
    ):
        return False
    dim_t, dim_h, dim_w = dim_split
    expected_shapes = (
        (q.shape[1], dim_t // 2),
        (q.shape[2], dim_h // 2),
        (q.shape[3], dim_w // 2),
    )
    flat_tables = tuple(table for pair in tables for table in pair)
    return (
        q.dtype is torch.bfloat16
        and k.dtype is q.dtype
        and q.is_cuda
        and k.is_cuda
        and q.device == k.device
        and k.shape == q.shape
        and all(size > 0 for size in q.shape)
        and q.shape[-1] == sum(dim_split)
        and all(dim > 0 and dim % 2 == 0 for dim in dim_split)
        and q.is_contiguous()
        and k.is_contiguous()
```

```
        and q.data_ptr() % 4 == 0
        and k.data_ptr() % 4 == 0
        and all(table.device == q.device for table in flat_tables)
        # 后续还有每个 table 的形状是否等于 expected_shapes 的逐项核对，完整实现见源文件
    )
```

python/sglang/kernels/jit/csrc/diffusion/ltx25_decoder_rope.cuh

新增 CUDA kernel 本体，呈现如何用一次 launch 完成 Q/K 的旋转、如何用 SGLANG_LDG 读取表格以及如何保持 *eager* 运算顺序。

```
// python/sglang/kernels/jit/csrc/diffusion/ltx25_decoder_rope.cuh
// 核心思想：每个线程处理一组 (even, odd) 对，Q/K 共用同一份表索引计算。

template <typename T>
SGL_DEVICE device::AlignedVector<T, 2> rotate_pair(
    device::AlignedVector<T, 2> input, float cos, float sin) {
  // 严格复刻 eager 的 float32 运算顺序：先分别乘，再减 / 加，
  // 不引入 FMA，保证与 torch eager 结果逐位一致。
  const float even = static_cast<float>(input[0]);
  const float odd = static_cast<float>(input[1]);
  const float even_cos = __fmul_rn(even, cos);
  const float odd_sin = __fmul_rn(odd, sin);
  const float even_sin = __fmul_rn(even, sin);
  const float odd_cos = __fmul_rn(odd, cos);
  device::AlignedVector<T, 2> output;
  output[0] = static_cast<T>(__fsub_rn(even_cos, odd_sin));
  output[1] = static_cast<T>(__fadd_rn(even_sin, odd_cos));
  return output;
}

template <typename T>
__global__ void ltx25_decoder_rope_kernel(
    T* __restrict__ q_out, T* __restrict__ k_out,
    const T* __restrict__ q, const T* __restrict__ k,
    const float* __restrict__ cos_t, const float* __restrict__ sin_t,
    const float* __restrict__ cos_h, const float* __restrict__ sin_h,
    const float* __restrict__ cos_w, const float* __restrict__ sin_w,
    int64_t num_pairs, int64_t num_frames, int64_t height, int64_t width,
    int64_t num_heads, int64_t pairs_per_head,
    int64_t t_pairs, int64_t h_pairs) {
  using Pair = device::AlignedVector<T, 2>;
  // grid-stride 循环，一个线程负责多个 pair；
  // pair 按 (token, head, pair_in_head) 排列，先算位置再选表。
  const int64_t stride = static_cast<int64_t>(gridDim.x) * blockDim.x;
  for (int64_t pair_index = static_cast<int64_t>(blockIdx.x) * blockDim.x + threadIdx.x;
       pair_index < num_pairs; pair_index += stride) {
    const int64_t pair_in_head = pair_index % pairs_per_head;
    const int64_t row = pair_index / (num_heads * pairs_per_head);
    const int64_t token = row % (num_frames * height * width);
```

```
    const int64_t frame = token / (height * width);
    const int64_t spatial = token % (height * width);
    const int64_t y = spatial / width;
    const int64_t x = spatial % width;

    // 根据 pair 在 head 内的归属选 T/H/W 中的某一组表；
    // 使用 SGLANG_LDG 走只读缓存路径，减少显存带宽压力。
    const float* cos_table;
    const float* sin_table;
    int64_t table_index;
    if (pair_in_head < t_pairs) {
      cos_table = cos_t;
      sin_table = sin_t;
      table_index = frame * t_pairs + pair_in_head;
    } else if (pair_in_head < t_pairs + h_pairs) {
      cos_table = cos_h;
      sin_table = sin_h;
      table_index = y * h_pairs + pair_in_head - t_pairs;
    } else {
      const int64_t w_pairs = pairs_per_head - t_pairs - h_pairs;
      cos_table = cos_w;
      sin_table = sin_w;
      table_index = x * w_pairs + pair_in_head - t_pairs - h_pairs;
    }
    const float cos_value = SGLANG_LDG(cos_table + table_index);
    const float sin_value = SGLANG_LDG(sin_table + table_index);

    // Q 与 K 在同一对坐标下旋转，因此只做一次表查询，两次旋转
    Pair q_pair;
    q_pair.load(q, pair_index);
    rotate_pair(q_pair, cos_value, sin_value).store(q_out, pair_index);

    Pair k_pair;
    k_pair.load(k, pair_index);
    rotate_pair(k_pair, cos_value, sin_value).store(k_out, pair_index);
  }
}
```

评论区精华

本 PR 没有实质性的 review 评论或评审线程，主要的 " 讨论 " 体现在 PR body 与 CI 状态中：

- 作者在 body 里反复强调相同 SHA256 输出，说明 " 性能优化不得改变数值结果 " 是本次融合的硬约束，所有测试都以 torch.equal 做逐位断言，而不是宽容差。
- 由于 LTX-2.5 的 HF checkpoint 是 gated 仓库、benchmark 主机没有可用凭据，组件基准使用的是确定性初始化权重的全尺寸 decoder；作者明确指出这一局限，未把未经验证的 checkpoint 结果当作正式结论。

- 评论区只有一个 CI 失败运行链接（Run #32378347205）与 mintlify 的文档预览提示，未展开技术讨论，CI 结论需后续观察。
- 真实 checkpoint 验证缺口 (question): 作者明确标注该局限，未声称已通过官方权重验证；建议后续在具备凭证的环境补跑一次端到端验证。
- CI 失败运行链接 (question): 失败原因未在本 PR 内展开，需要关注后续 CI 状态与重跑结果。

风险与影响

- 风险：主要风险集中在四点：

1. 真实 checkpoint 未端到端验证：单测与基准都基于确定性初始化权重，gated 的官方 LTX-2.5 权重没有跑过完整链路。虽然 bit-exact 校验覆盖了数学等价性，但量化、加载路径或权重分布差异仍可能在真实推理中暴露问题。
2. JIT 编译与运行环境依赖：快速路径依赖 load_jit/TVM FFI 编译 CUDA 源码，在首次调用时有编译开销和失败可能；虽然 can_use_ltx25_decoder_rope 会预检并在不支持时回退 eager，但若 kernel 在运行时非法内存访问而被误判为 " 验证通过 "，影响会更隐蔽。
3. 全局表缓存的状态性：_ROPE_TABLE_CACHE 是模块级 dict，key 含 device 和网格尺寸，多租户或动态分辨率场景下若缓存 key 碰撞或设备切换，可能出现表复用偏差；上限 16 的过期策略也可能在极端高分辨率组合下反复重建。
4. gate 状态持久性：BitExactFusionGate 按配置记录 " 已验证一致 / 永久禁用 "，但该状态是否跨进程、跨形状转换安全迁移，PR 内没有充分说明；同一服务器上不同 batch/grid 混用时的 gate 生命周期值得关注。 - 影响：对用户而言，LTX-2.5 视频生成解码阶段的延迟明显下降（组件级约 17%），且输出数值逐位不变、峰值显存几乎不增加，属于 " 免费午餐 " 型优化。对系统而言，sglang.kernels.ops.diffusion 新增了一个带预检和回退的 JIT 自定义算子，为后续 diffusion 模块的 kernel 融合提供了可复用的入口模式；同时 BitExactFusionGate 的 " 首次验证、永久信任 / 回退 " 策略若推广，会影响整个代码库对数值一致性的态度。对团队而言，这组改动确立了 " 性能优化必须伴随 bit-exact 证据 " 的验收习惯，但也在 runtime 中引入了新的全局缓存和编译期依赖，需要持续关注其稳定性。 - 风险标记：真实 checkpoint 未验证，依赖 JIT 编译链路，新增全局表缓存状态，bit-exact gate 生命周期未明，仅覆盖 CUDA/BF16

关联脉络

- PR #35707 [diffusion] read the cgroup this process is actually in: 同一阶段对 LTX-2.5 diffusion 运行时的内存管理路径做修复，与本 PR 同属 multimodal_gen/diffusion 的稳定性与性能维护脉络。
- PR #34247 [Docs] Standardize diffusion cookbook model pages: 同期对 diffusion 文档体系做标准化，本 PR 也同步更新了 fused_kernels.mdx 文档位，两者共同完善 diffusion 功能区的用户可见描述。