# PR #34004 完整报告

sgl-project/sglang

[diffusion] FLUX.1 fused adaLN modulate (bit-exact) + RoPE cache hoist, LN-affine folding behind quality=high (H200 e2e -3.5% lossless / -6.9% high)

合并时间：2026-08-08 13:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34004

## 执行摘要

- 一句话：FLUX.1 融合调制与 RoPE 缓存，提速 3.5%/6.9%
- 推荐动作：值得精读。核心看三点：一是 bit-exact 融合方法论——逐算子舍入复现 + 显式阻止 FMA 收缩 + 守卫回退 + torch.compile 图内禁静默回退，可推广到任意 elementwise 链融合；二是 quality 分层 + mount/unmount 协议，这是仓库在 #28708 回退教训后沉淀的"近似加速落地"标准答案；三是 PR body 的数值论证格式（HBM pass 数、microbench 归因 vs 端到端实测、md5/PSNR 三件套），是高质量性能 PR 的范本。建议抽读 flux.py 的 _flux_modulate / _flux_norm_modulate 与 modulate_scale_shift.cuh 的 modulate_value。

## 功能与动机

FLUX.1 在 H200 上已跑到约 97.8% GPU 占用，剩余 eager 逐元素链（占 step kernel 时间 13.4%）的每一毫秒都直接转化为墙钟，PR body 原话："FLUX.1 runs at ~97.8% GPU busy, so the remaining eager elementwise chains (13.4% of step kernel time) convert directly into wall-clock." 具体账目：adaLN modulate 链 norm(x) * (1 + scale) + shift 每 step 有 114 个站点（19 个 dual block × 4 + 38 个 single block × 1），每个站点三个 eager elementwise kernel，其中 mul 与 add 是对 [1, L, 3072] 激活的两次完整 HBM 遍历（S1 trace 实测 mul 3.09 ms/step + add 3.14 ms/step）；RoPE 缓存重建则让每个 attention 都重复执行 torch.cat([cos.float(), sin.float()], -1)，57 次启动 /step、每图 2,850 次，几乎全部产出同一份 tensor。这是 #33819 之后既定清理路线的第三波，目标是在默认路径输出逐位不变的前提下，把这些浪费直接换成吞吐。

## 实现拆解

1. RoPE cos/sin 缓存提升（commit 1，lossless）。在 flux.py 新增 _rope_cos_sin_cache()，把 FluxAttention.forward 中每次调用都执行的 fp32 拼接（torch.cat([cos.float(), sin.float()], -1)）上移到 FluxTransformer2DModel.forward：每个去噪 step 只构建一次（SP 文本分片激活时为 single-block 重排单独再构建一份），以 tensor 形式下传给各 attention。FluxAttention 仍兼容接收原始 (cos, sin) 元组的调用方（内部同样走 _rope_cos_sin_cache，已构建的 tensor 直接透传）。每 step 57 次启动降为 1 次，所有 attention 消费的数值与改动前完全相同，属构造性 lossless。

2. bit-exact 融合 modulate（commit 2，lossless）。内核层新增 python/sglang/kernels/jit/csrc/diffusion/modulate_scale_shift.cuh（+221），与 residual_gate_add.cuh 同为 vec-8 行广播 tile 结构（kVec = 16 / sizeof(T)），用 grid-stride 覆盖 row-tile 超过硬件 gridDim.y（65535）上限的情形；核心 modulate_value 按 round(round(x * round(1 + scale)) + shift) 逐步舍入，复现 eager 链的逐算子 fp32-opmath/ 舍入到存储精度边界，并结构性阻止 FMA 收缩。Python 封装新增 python/sglang/kernels/ops/diffusion/modulate_scale_shift.py（+95），cache_once 按 dtype 缓存 JIT 模块，register_custom_op 注册 diffusion_modulate_scale_shift 自定义算子（含 _fake_impl 供 torch.compile 做形状推导）；can_use_modulate_scale_shift_cuda 守卫半精度 dtype、同设备、3D/2D 形状匹配、连续性、非空、D 为向量宽度倍数与 16 字节对齐。模型接线在 flux.py（+149/-22）：新增 _flux_modulate（进程级一次性禁用开关 _FLUX_MODULATE_CUDA_DISABLED + torch.compiler.is_compiling() 图内抛错的逃生口）与本地子类 FluxAdaLayerNormZero / FluxAdaLayerNormZeroSingle（参数与 state-dict 与 diffusers 父类一致），替换 dual-block 的 norm1/norm1_context 与 single-block 的 norm；norm2/norm2_context 的调制也统一改经 _flux_norm_modulate 路由，Nunchaku 分支保持原样。一个值得注意的细节：守卫的 is_contiguous() 检查与 PyTorch 跳过 size-1 维的 contiguous 语义一致，因此 emb.chunk(6) 产生的 [1, D] 视图无需 .contiguous() 拷贝即可进内核（测试 test_modulate_scale_shift_adaln_chunk_views 专门覆盖）。
3. LN affine 折叠（commit 3，quality="high" 门控）。新增 python/sglang/kernels/ops/diffusion/fused_ln_modulate.py（+85）：mark_fused_ln_modulate_site 给站点打默认关闭的属性，mount/unmount_fused_ln_modulate 沿 root.modules() 递归开关，fused_ln_modulate 实现 F.layer_norm(x, weight=(1 + scale).reshape(-1), bias=shift.reshape(-1), eps=eps) 的单 kernel 折叠；can_fuse_ln_modulate 逐调用守卫要求 B == 1（折叠 affine 是 [D] 行）。denoising.py 的 _maybe_toggle_quality_fusions 在 batch 边界按 quality == "high" 挂载并打印 Mounted fused LN+modulate (affine folding) for quality=high。该路径非 bit-exact（bf16 舍入序差异，max abs diff ~0.05）故必须门控；PR body 论证了 lossless 路径不融合 LN 归约的原因——aten LayerNorm 用逐元素 Welford 更新 + count-weighted cuWelfordCombine 合并 + 多指令 rsqrtf，Triton 逐位复刻风险高（#33819 的审计也排除了现有候选）。
4. 测试与验证配套。新增两个注册 CUDA 单测：test_modulate_scale_shift.py（FLUX 真实形状 [1, L, 3072] + batched + 奇数长度共 5 形状 × bf16/fp16 的 torch.equal 位级契约，注册 1-gpu-large 与 4-gpu-b200 两个 runner）与 test_fused_ln_modulate.py（assert_close(atol=0.0625, rtol=0.05) 接近性契约 + 挂载协议与 B==1 守卫）。端到端：lossless 主路径 32 张图（main / commit-1 / PR / final 各档）md5 全同 cad50fb5...；high 档 PSNR 35.48–37.95 dB、SSIM 0.9785–0.9916；相邻套件 45 passed。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/flux.py（模块 模型实现；类别 source；类型 core-logic；符号 _flux_modulate, _flux_norm_modulate, FluxAdaLayerNormZero, init）：主战场（+149/-22）：新增 _flux_modulate / _flux_norm_modulate 分发、FluxAdaLayerNormZero{,Single} 本地子类替换 diffusers 父

类、norm2/norm2_context 调制路由改造与 _rope_cos_sin_cache 缓存提升；freqs_cis参数契约从仅元组扩展为 tensorl 元组。

- python/sglang/kernels/ops/diffusion/modulate_scale_shift.py（模块 内核封装；类别 infra；类型 core-logic；符号 _jit_modulate_scale_shift_module, _fake_impl, _modulate_scale_shift_custom_op, _aligned）：新内核的 Python 侧封装（+95）：JIT 加载、register_custom_op 注册（含 fake_impl 供 torch.compile 形状推导）与 can_use_modulate_scale_shift_cuda 守卫，是 bit-exact 契约的入口。
- python/sglang/kernels/jit/csrc/diffusion/modulate_scale_shift.cuh（模块 CUDA 内核；类别 source；类型 core-logic；符号 modulate_value, modulate_scale_shift_vec_kernel, ModulateScaleShiftKernel）：bit-exact 契约的物理实现（+221）：vec-8 行广播 tile 结构、grid-stride 行循环、逐步舍入的 modulate_value，是性能与数值正确性的交汇点。
- python/sglang/kernels/ops/diffusion/fused_ln_modulate.py（模块 挂载协议；类别 infra；类型 infrastructure；符号 mark_fused_ln_modulate_site, fused_ln_modulate_active, iter_fused_ln_modulate_sites, mount_fused_ln_modulate）：quality=high 的 LN affine 折叠与挂载协议（+85）：mark/mount/unmount/active 四件套与 can_fuse_ln_modulate 的 B==1 守卫，是把非 bit-exact 加速安全落到请求级的关键机制。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 去噪阶段；类别 source；类型 dependency-wiring；符号 _maybe_toggle_quality_fusions）：DenoisingStage 的 _maybe_toggle_quality_fusions 增加 mount/unmount_fused_ln_modulate（+9），使 LN 折叠与 GELU epilogue 同生命周期，按 batch 边界统一挂载。
- test/registered/kernels/ops/diffusion/test_modulate_scale_shift.py（模块 内核测试；类别 test；类型 test-coverage；符号 cuda_setup, _eager, test_modulate_scale_shift_matches_eager, test_modulate_scale_shift_adaln_chunk_views）：bit-exact 契约的注册单测（+56）：FLUX 真实形状的 torch.equal 位级断言、adaLN chunk 视图直通与 fp32 拒绝守卫，注册 1-gpu-large 与 4-gpu-b200 双 runner。
- test/registered/kernels/ops/diffusion/test_fused_ln_modulate.py（模块 内核测试；类别 test；类型 test-coverage；符号 cuda_setup, test_fused_ln_modulate_matches_reference, test_fused_ln_modulate_guards_and_mount_protocol）：LN 折叠的接近性契约与挂载协议测试（+56）：断言 bf16 舍入序级差异（atol=0.0625）与 mount/unmount 全流程、B==1 守卫。

关键符号：_flux_modulate, _flux_norm_modulate, _rope_cos_sin_cache, FluxAdaLayerNormZero.forward, FluxAdaLayerNormZeroSingle.forward, modulate_scale_shift_cuda, can_use_modulate_scale_shift_cuda, _modulate_scale_shift_custom_op, _jit_modulate_scale_shift_module, fused_ln_modulate, can_fuse_ln_modulate, mark_fused_ln_modulate_site, mount_fused_ln_modulate, unmount_fused_ln_modulate, _maybe_toggle_quality_fusions, modulate_value, modulate_scale_shift_vec_kernel

## 关键源码片段

## python/sglang/multimodal_gen/runtime/models/dits/flux.py

主战场（+149/-22）：新增 _flux_modulate / _flux_norm_modulate 分发、FluxAdaLayerNormZero{,Single} 本地子类替换 diffusers 父类、norm2/norm2_context 调制路由改造与 _rope_cos_sin_cache 缓存提升；freqs_cis 参数契约从仅元组扩展为 tensorl 元组。

```
# FLUX.1 adaLN modulate 的统一分发入口：
# • 默认（lossless）：无 affine LayerNorm + bit-exact 融合 modulate，输出与 eager 逐位一致；
# • quality="high" 且守卫通过：把 modulate 折叠进 LN 的 elementwise affine（单 kernel，非 bit-exact）。
# 挂载状态是模块上的开关属性，由 DenoisingStage 按 batch
边界统一切换，站点默认关闭（参考路径）。

_FLUX_MODULATE_CUDA_DISABLED = False


def _flux_modulate(x, scale, shift):
    # 单 CUDA kernel 完成 x * (1 + scale[:, None]) + shift[:, None]。
    # kernel 复现 eager 链的逐算子 fp32-opmath / 舍入到存储精度（bf16/fp16）边界，
    # 因此 bit-exact（torch.equal 验证），无需 quality 门控；守卫失败回退 eager。
    global _FLUX_MODULATE_CUDA_DISABLED

    if not _FLUX_MODULATE_CUDA_DISABLED and can_use_modulate_scale_shift_cuda(
        x, scale, shift
    ):
        try:
            return modulate_scale_shift_cuda(x, scale, shift)
        except Exception as exc:
            # 编译图内禁止静默回退：一旦回退分支被固化进图就永远失去 fast path；
            # 图外首次失败则进程内一次性禁用，避免每步重复告警与重试。
            if torch.compiler.is_compiling():
                raise
            logger.warning_once(f"Disabling FLUX modulate CUDA fast path: {exc}")
            _FLUX_MODULATE_CUDA_DISABLED = True

    return x * (1 + scale[:, None]) + shift[:, None]


def _flux_norm_modulate(site, norm, x, scale, shift):
    # FLUX adaLN 站点的统一入口：norm(x) * (1 + scale) + shift。
    # 默认 = 无 affine 的 LayerNorm + bit-exact 融合 modulate（两趟 HBM）；
    # 站点被挂载（quality="high"）且逐调用守卫通过时，改为单 kernel 的
    # F.layer_norm(x, weight=1 + scale, bias=shift) 折叠（一趟 HBM，非 bit-exact）。
    # scale/shift 是 emb.chunk 产生的 [1, D] 视图时，PyTorch 的 is_contiguous()
    # 会跳过 size-1 维，因此能直接通过内核守卫，无需 .contiguous() 拷贝。
    if fused_ln_modulate_active(site) and can_fuse_ln_modulate(x, scale, shift):
        return fused_ln_modulate(x, scale, shift, norm.eps)
    return _flux_modulate(norm(x), scale, shift)
```

```
class FluxAdaLayerNormZero(AdaLayerNormZero):
    # diffusers AdaLayerNormZero 的本地子类：参数与 state-dict 与父类完全一致，
    # 仅把 modulate 路由到 _flux_norm_modulate，并在构造时标记为可折叠站点。
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        mark_fused_ln_modulate_site(self) # 默认关闭，quality="high" 时统一挂载

    def forward(self, x, timestep=None, class_labels=None, hidden_dtype=None, emb=None):
        if self.emb is not None:
            emb = self.emb(timestep, class_labels, hidden_dtype=hidden_dtype)
        emb = self.linear(self.silu(emb))
        shift_msa, scale_msa, gate_msa, shift_mlp, scale_mlp, gate_mlp = emb.chunk(6, dim=1)
        x = _flux_norm_modulate(self, self.norm, x, scale_msa, shift_msa)
        return x, gate_msa, shift_mlp, scale_mlp, gate_mlp
```

## python/sglang/kernels/ops/diffusion/modulate_scale_shift.py

新内核的 Python 侧封装（+95）：JIT 加载、register_custom_op 注册（含 fake_impl 供 torch.compile 形状推导）与 can_use_modulate_scale_shift_cuda 守卫，是 bit-exact 契约的入口。

```
# 融合 adaLN modulate 的 Python 侧封装（新文件 +95 行）：
# JIT 加载 CUDA 内核 + 守卫校验 + 自定义算子注册三件事。
# 数值契约：内核复现 eager 链的逐算子 fp32-opmath / 舍入到存储类型边界
# （仅 fp16/bf16），输出与 eager 完全 bit-exact，因此不需要 quality 门控。

_SUPPORTED_DTYPES = (torch.float16, torch.bfloat16)
_ALIGN_BYTES = 16


@cache_once
def _jit_modulate_scale_shift_module(dtype: torch.dtype) -> Module:
    args = make_cpp_args(dtype)
    return load_jit(
        "diffusion_modulate_scale_shift",
        *args,
        cuda_files=["diffusion/modulate_scale_shift.cuh"],
        cuda_wrappers=[
            (
                "modulate_scale_shift",
                "sglang_modulate_scale_shift::" f"ModulateScaleShiftKernel<{args}>::run",
            ),
        ],
    )


def _fake_impl(x, scale, shift):
    # torch.compile 下自定义算子不透明，fake 实现负责形状 / 设备推导
    return torch.empty_like(x)
```

```
@register_custom_op(
    op_name="diffusion_modulate_scale_shift",
    mutates_args=[],
    fake_impl=_fake_impl,
)
def _modulate_scale_shift_custom_op(x, scale, shift):
    out = torch.empty_like(x)
    module = _jit_modulate_scale_shift_module(x.dtype)
    module.modulate_scale_shift(out, x, scale, shift)
    return out


def _aligned(t):
    return t.data_ptr() % _ALIGN_BYTES == 0


def can_use_modulate_scale_shift_cuda(x, scale, shift) -> bool:
    # 守卫语义与调用场景严格对应：
    #・只服务半精度、CUDA、同设备、3D/2D 形状匹配、非空、连续输入；
    #・D 须为向量宽度（16 字节 / 元素大小）整数倍，且三个指针 16 字节对齐；
    #・PyTorch 的 is_contiguous() 会跳过 size-1 维，因此 emb.chunk() 产生的
    # [1, D] 视图可直接进内核而无需 .contiguous() 拷贝。
    if (
        x.dtype not in _SUPPORTED_DTYPES
        or scale.dtype != x.dtype
        or shift.dtype != x.dtype
        or not (x.is_cuda and scale.is_cuda and shift.is_cuda)
        or not (x.device == scale.device == shift.device)
        or x.dim() != 3
        or scale.dim() != 2
        or shift.shape != scale.shape
        or scale.shape != (x.shape[0], x.shape[-1])
        or not (x.is_contiguous() and scale.is_contiguous() and shift.is_contiguous())
        or x.numel() == 0
    ):
        return False
    vec = _ALIGN_BYTES // x.element_size()
    return (
        x.shape[-1] % vec == 0 and _aligned(x) and _aligned(scale) and _aligned(shift)
    )


def modulate_scale_shift_cuda(x, scale, shift):
    # Fused x * (1 + scale[:, None]) + shift[:, None]（与 eager 完全 bit-exact）
    if not can_use_modulate_scale_shift_cuda(x, scale, shift):
        raise RuntimeError("unsupported input for modulate_scale_shift CUDA")
    return _modulate_scale_shift_custom_op(x, scale, shift)
```

## 评论区精华

PR 无外部 review 评论：唯一的 comment 是作者 BBuf 附的 CI 运行链接，且由作者自行合并（merged_by: BBuf）。有效的"讨论"全部沉淀在 PR body 的决策自述里：一是为何 lossless 路径不融合 LN 归约——"aten's LayerNorm uses per-element Welford updates (mean + delta * (1.f/new_count)), count-weighted cuWelfordCombine merges, and the accurate multi-instruction rsqrtf — with the effective FMA-contraction choices baked into shipped libtorch SASS. Reproducing that bit-for-bit in Triton is a full task with real failure risk, so the LN fusion ships as the gated affine-folding instead (the fallback the plan anticipated)"；二是对 #28708 历史回退教训的回应——当年回退原因是"the fused path can move generated images away from the original model behavior"，而 quality 分层（#33453）让这类近似加速以 opt-in 方式重新落地："sites default off, lossless requests keep the bit-exact reference path"。

- 暂无高价值评论线程

## 风险与影响

- 风险：1）lossless 位级契约现在依赖运行时 JIT 内核与守卫：守卫失败会静默回退 eager（正确性不变），但 _FLUX_MODULATE_CUDA_DISABLED 在首次异常后进程内永久禁用 fast path（性能劣化），且 torch.compiler.is_compiling() 图内异常会直接抛出，CUDA graph 捕获场景需留意。2）挂载 / 卸载与禁用开关都是进程级可变状态，正确性依赖"quality 参与 dynamic-batch 签名、单 worker batch 内 quality 均匀"的批注不变量；若未来放开混批，存在 wrong-path 执行风险。3）守卫覆盖有限：B > 1、非对齐、非连续的批量请求自动退回 eager（正确但不加速）；FLUX.1-dev 的 embedded guidance 场景恰好 B=1 且 gate 稠密，实测全命中。4）SP 文本分片下 single-block 的 RoPE 缓存独立重排没有专门单测，md5 验证主要在单卡 H200 完成。5）high 档数值差异（max abs diff ~0.05）是契约内的，但换模型或换 torch 版本时 PSNR 需重新验证。
- 影响：对 FLUX.1 系用户：默认（lossless）路径输出逐位不变而端到端快约 3.5%（denoise -3.0%），quality="high" 用户相对 main lossless 总收益 -6.9%（denoise -5.9%），图像质量保持 PSNR 35+ dB。对系统：新增 modulate_scale_shift JIT 内核与 fused_ln_modulate 挂载协议两个可复用构件，GLM-Image 等同族模型可直接套用；CI 增加两个注册 kernel 单测套件。对团队：确立了"lossless 位级一致 + high 档 opt-in 近似"验收模板（microbench 归因 + 端到端 md5 + PSNR/SSIM 三件套），后续同类融合 PR 可照此评审。影响面限于 diffusion 路径与 FLUX.1 模型家族，不触及 SRT 核心推理路径。
- 风险标记：lossless 位级契约依赖运行时 JIT 内核，进程级全局状态（禁用开关与挂载标志），单 worker batch 内 quality 均匀性假设，SP 文本分片 RoPE 重排路径缺独立测试，B>1 批量请求自动退回 eager

## 关联脉络

- PR #33819 [diffusion] FLUX.1 bit-exact residual-gate fast path + tanh-GELU epilogue behind quality=high (H200 e2e -1.1% lossless / -4.3% high): 本 PR 的直接前驱（同作者、同一 mount 协议与拨测 harness），本 PR 是同一清理路线的第三波；residual-gate

的位级契约与 GELU epilogue 门控模式在此确立。

- PR #33536 [diffusion] Fuse DiT FFN tanh-GELU into up-proj GEMM (cublasLt epilogue) behind quality=high (Qwen-Image 1024^2 denoise 12.36 -> 12.05 s on H200): quality=high 挂载协议（DenoisingStage mount/unmount）与 cublasLt GELU epilogue 机制的源头，本 PR 的 LN affine 折叠沿用同一模式。
- PR #33451 [diffusion] FLUX.2 VAE decoder fast path behind quality=high (H200: 1024^2 97.6->29.2 ms, 2048^2 437.2->168.5 ms): quality 门控在 diffusion 路径的另一个应用（FLUX.2 VAE 解码快路径），同属 #33453 分层体系。
- PR #34008 [diffusion] GLM-Image bit-exact fused aten LayerNorm+modulate / qk-LN (H200 30-step denoise -8.1%): 同期并行的 GLM-Image bit-exact 融合 LayerNorm+modulate，与本 PR 构成 adaLN 链融合的两种实现路线对照（复用单个 aten kernel vs 自定义 CUDA 内核）。