PR #37129 完整报告

sgl-project/sglang

[Diffusion] Fuse Qwen-Image residual norm and NVFP4 quantization

合并时间：2026-09-01 08:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37129

执行摘要

- 一句话：Qwen-Image 残差归一化与 NVFP4 量化融合，端到端提速约 1.2%
- 推荐动作：值得精读。重点关注四个设计决策：(1) 用生产量化 helper（TensorRT-LLM 的 cvt_warp_fp16_to_fp4 / get_sf_out_offset_128x4）保证字节一致性而非自研打包；(2) 独立 JIT 模块隔离 FlashInfer 内部头依赖，避免污染既有 BF16 norm kernel 的构建；(3) 完整 gate 矩阵与三个环境变量逃生舱，保障与生产量化器语义对齐；(4) 从 kernel launch 计数到 PNG SHA256 的多层验证方法论。若团队后续推进 Blackwell diffusion 模型的 kernel 融合，本 PR 是理想范本。评分：整体重要度 6（有意义的性能优化，但影响面窄、增益温和），洞察价值 6（依赖隔离与字节一致性设计值得借鉴）。

功能与动机

PR 实现的是 Baseten 博客「Agentic Kernels in Production」中描述的 Qwen-Image NVFP4 resnorm_quant 优化：在 Blackwell 上用单个 kernel 连续完成 residual_out = residual + gate * (attention_output + output_bias)、FP32 累加的 LayerNorm、BF16 scale/shift 调制、E2M1 激活打包与 128x4 swizzled E4M3 scale 写入，直接为下一个 ModelOpt FP4 FC1 GEMM 产出输入。PR body 明确动机：'This removes the large BF16 modulation intermediate and the following flashinfer.fp4_quantize launch.'，即消除每 transformer block 一次的全尺寸 BF16 中间张量（1×T×3072）与一次额外 launch；同时 'The conversion uses the same FlashInfer/TensorRT-LLM helper as the production quantizer, so packed values, scales, and residual output are byte-exact'——融合不得引入任何数值漂移，这是该优化区别于单纯加速的关键约束。

实现拆解

实现按 5 步拆解：

1. 新增 NVFP4 专用 JIT 模块与依赖定位（norm_scale_shift_jit.py + deps.py）：新增 norm_scale_shift_nvfp4_module，以 ENABLE_BF16 / ENABLE_FP4 宏和 extra_dependencies=["flashinfer", "flashinfer_nv_internal"] 独立编译，使现有 BF16 norm JIT 不获得 FlashInfer 构建依赖；deps.py 注册 get_flashinfer_nv_internal_include_paths，定位 flashinfer/data/csrc/nv_internal 头目录。新增入口 try_fused_scale_residual_norm_scale_shift_nvfp4，gate 矩阵包括：非 torch.compile、非 CUDA graph capture、纯 LayerNorm（无仿射 weight/bias）、BF16 激活、SM10x、global_scale 为 FP32 CUDA 标量，且未被 FLASHINFER_DISABLE_FP4_QUANT_FAST_

MATH / TRTLLM_DISABLE_FP4_QUANT_FAST_MATH / FLASHINFER_NVFP4_4OVER6 环境变量禁用；不满足即返回 None 由调用方回退。

2. CUDA kernel 扩展（norm_scale_shift.cuh）：NormScaleShiftParams 增加 quant_scales / global_scale / num_rows；norm_scale_shift_kernel 增加 kQuantizeNvfp4 模板分支，复用 TensorRT-LLM 的 cvt_warp_fp16_to_fp4 与 get_sf_out_offset_128x4 完成 E2M1 打包与 128x4 swizzled E4M3 scale 写入，越界（padding）行写入 0 scale；新增 ScaleResidualNormScaleShiftNvfp4Kernel host wrapper 校验 quantized 行数与 scale 行 128 填充几何。所有既有 launch 站点（FP8 / 普通 norm）补齐新字段，行为不变。
3. 模型侧集成（qwen_image.py）：QwenImageTransformerBlock.__init__ 在「非 nunchaku、dim == 3072、SM10x、图像 / 文本 FC1 均为 ModelOptFp4LinearMethod」时置位 _enable_nvfp4_resnorm_quant；新增 _try_nvfp4_resnorm_quant，在 forward 图像 / 文本两流中先于 FP8 路径尝试，失败依次回退 _try_fp8_residual_norm_quant 与 _modulate。QwenImageGELU.forward 与 QwenImageFeedForward 支持接收（packed, scales）元组直接喂给预量化 GEMM，普通 tensor 输入路径不变。
4. 预量化 GEMM 助手（modelopt_quant.py）：新增 apply_nvfp4_gemm_prequantized，从 ModelOptFp4LinearMethod.apply 提取 GEMM 段，接受已打包的 x_fp4 / x_scale_interleaved，复用权重 padding、E4M3 视图转换、_get_fp4_gemm_op 后端选择与 slice_nvfp4_output；apply 本身保持不变，普通线性调用保持引用行为。
5. 测试与基准：test_diffusion_nvfp4_scaled_mm.py 新增 test_qwen_image_fused_resnorm_nvfp4_quant_is_exact（17 / 1024 行，17 行覆盖 padding 敏感场景，fused 与 baseline 三路输出 torch.equal）与 test_prequantized_input_matches_regular_apply（default/flashinfer_trtllm两个后端）；新增独立 benchmark bench_qwen_image_resnorm_nvfp4_quant.py 覆盖 17 / 1024 / 4096 / 4608 行并同时断言字节一致。__init__.py 注册 diffusion.scale_residual_norm_scale_shift_nvfp4 backend 并导出 try_fused_scale_residual_norm_scale_shift_nvfp4。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py（模块 扩散模型；类别 source；类型 data-contract；符号 forward, _try_nvfp4_resnorm_quant, _enable_nvfp4_resnorm_quant）：融合路径的模型侧入口：新增 _enable_nvfp4_resnorm_quant 标志与 _try_nvfp4_resnorm_quant，并在 forward 中把 NVFP4 尝试置于 FP8 路径之前，同时让 QwenImageGELU / QwenImageFeedForward 支持接收预量化元组。
- python/sglang/kernels/ops/diffusion/norm/norm_scale_shift_jit.py（模块 内核 JIT；类别 infra；类型 infrastructure；符号 _blackwell_sm10x, norm_scale_shift_nvfp4_module, _env_enabled, try_fused_scale_residual_norm_scale_shift_nvfp4）：提供独立 norm_scale_shift_nvfp4_module 与门控入口 try_fused_scale_residual_norm_scale_shift_nvfp4，用环境变量与运行时条件严格限制快路径适用范围，避免污染现有 BF16 norm JIT 依赖。

- python/sglang/multimodal_gen/runtime/layers/quantization/modelopt_quant.py（模块 量化层；类别 source；类型 data-contract；符号 apply_nvfp4_gemm_prequantized）：新增 apply_nvfp4_gemm_prequantized，抽取 ModelOptFp4LinearMethod.apply 的 GEMM 段以直接消费已打包激活；apply 保持引用行为不变，普通线性调用不受影响。
- python/sglang/kernels/jit/csrc/diffusion/norm_scale_shift.cuh（模块 CUDA 内核；类别 other；类型 dependency-wiring；符号 ScaleResidualNormScaleShiftNvfp4Kernel, norm_scale_shift_kernel, NormScaleShiftParams）：kernel 本体：扩展参数结构，新增 kQuantizeNvfp4 模板分支，复用 TensorRT-LLM cvt_warp_fp16_to_fp4 与 get_sf_out_offset_128x4 完成字节一致的 E2M1 打包和 128x4 swizzled scale 写入，并处理 padding 行清零。
- test/registered/kernels/ops/quantization/test_diffusion_nvfp4_scaled_mm.py（模块 量化测试；类别 test；类型 test-coverage；符号 _qwen_resnorm_nvfp4_supported, test_qwen_image_fused_resnorm_nvfp4_quant_is_exact, test_prequantized_input_matches_regular_apply）：字节级一致性契约的固化：融合结果与 baseline（既有 norm+modulation+flashinfer.fp4_quantize）在 packed 值、scale（按 uint8 视图）、residual 三路逐位相等，覆盖 17 行 padding 敏感场景；预量化 GEMM 与常规 apply 等价。
- test/registered/kernels/benchmark/diffusion/bench_qwen_image_resnorm_nvfp4_quant.py（模块 基准测试；类别 test；类型 test-coverage；符号 _benchmark, _run_case, baseline, fused）：提交生产组合微基准，覆盖 17 / 1024 / 4096 / 4608 行，同时断言 fused 与 baseline 字节一致并输出 speedup（约 3x）。
- python/sglang/kernels/jit/utils/deps.py（模块 依赖解析；类别 source；类型 core-logic；符号 get_flashinfer_nv_internal_include_paths）：注册 flashinfer_nv_internal 依赖定位器，让 NVFP4 JIT 能定位 data/csrc/nv_internal 头文件，且不影响其他 kernel 的依赖解析。
- python/sglang/kernels/ops/diffusion/__init__.py（模块 内核注册；类别 infra；类型 infrastructure；符号 try_fused_scale_residual_norm_scale_shift_nvfp4）：注册新 kernel backend 并导出 try_fused_scale_residual_norm_scale_shift_nvfp4，接通外部调用面。

关键符号：QwenImageTransformerBlock._try_nvfp4_resnorm_quant, QwenImageTransformerBlock.forward, QwenImageGELU.forward, apply_nvfp4_gemm_prequantized, try_fused_scale_residual_norm_scale_shift_nvfp4, norm_scale_shift_nvfp4_module, get_flashinfer_nv_internal_include_paths, ScaleResidualNormScaleShiftNvfp4Kernel::run, test_qwen_image_fused_resnorm_nvfp4_quant_is_exact, test_prequantized_input_matches_regular_apply

关键源码片段

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

融合路径的模型侧入口：新增 _enable_nvfp4_resnorm_quant 标志与 _try_nvfp4_resnorm_quant，并在 forward 中把 NVFP4 尝试置于 FP8 路径之前，同时让

QwenImageGELU / QwenImageFeedForward 支持接收预量化元组。

```
    # 仅在非 nunchaku、hidden size 3072、Blackwell SM10x 且图像 / 文本 FC1 均为
    # ModelOpt FP4 线性层时开启融合快路径；其余情况全程走原逻辑。
    self._enable_nvfp4_resnorm_quant = False
    capability = current_platform.get_device_capability()
    if (
        not nunchaku_enabled
        and dim == 3072
        and capability is not None
        and capability.major == 10
    ):
        img_fc1 = self.img_mlp.net[0].proj
        txt_fc1 = self.txt_mlp.net[0].proj
        self._enable_nvfp4_resnorm_quant = isinstance(
            img_fc1.quant_method, ModelOptFp4LinearMethod
        ) and isinstance(txt_fc1.quant_method, ModelOptFp4LinearMethod)


    # 一次调用完成 residual 合并、LayerNorm、scale/shift 调制与 FC1 输入 NVFP4 打包。
    def _try_nvfp4_resnorm_quant(
        self,
        norm_module: ScaleResidualLayerNormScaleShift,
        mlp: QwenImageFeedForward,
        *,
        residual: torch.Tensor,
        x: torch.Tensor,
        x_bias: Optional[torch.Tensor],
        residual_gate: torch.Tensor,
        mod_params: torch.Tensor,
        modulate_index: Optional[torch.Tensor],
        use_bcg_helpers: bool,
    ) -> Optional[tuple[tuple[torch.Tensor, torch.Tensor], torch.Tensor, torch.Tensor]]:
        # 动态 CFG modulation 需逐 batch 重选参数，融合 kernel 不支持，直接返回 None
        # 由调用方回退到 FP8 norm+quant 或普通 _modulate 路径，保证行为不漂移。
        if (
            not self._enable_nvfp4_resnorm_quant
            or modulate_index is not None
            or use_bcg_helpers
        ):
            return None

        # adaLN 参数按 shift / scale / gate 切分；FC1 的 input_scale_inv 直接作为
        # NVFP4 打包的 global_scale，保证量化口径与生产量化器 flashinfer.fp4_quantize 一致。
        shift, scale, gate = mod_params.chunk(3, dim=-1)
        fc1 = mlp.net[0].proj
        result = try_fused_scale_residual_norm_scale_shift_nvfp4(
            residual,
            x,
```

```
        x_bias,
        residual_gate,
        getattr(norm_module.norm, "weight", None),
        getattr(norm_module.norm, "bias", None),
        scale.unsqueeze(1),
        shift.unsqueeze(1),
        fc1.input_scale_inv,
        norm_module.norm_type,
        norm_module.eps,
    )
    if result is None:
        return None
    # packed 为（E2M1 激活，E4M3 scale）元组，直接喂给后续 FC1 的预量化 GEMM；
    # gate 与 residual_out 原样返回，FFN 之后的 gate 乘加语义保持不变。
    packed, residual_out = result
    return packed, residual_out, gate.unsqueeze(1)
```

python/sglang/kernels/ops/diffusion/norm/norm_scale_shift_jit.py

提供独立 norm_scale_shift_nvfp4_module 与门控入口 try_fused_scale_residual_norm_scale_shift_nvfp4，用环境变量与运行时条件严格限制快路径适用范围，避免污染现有 BF16 norm JIT 依赖。

```
def try_fused_scale_residual_norm_scale_shift_nvfp4(
    residual, x, input_bias, gate, weight, bias, scale, shift,
    global_scale, norm_type, eps,
):
    """融合 Qwen residual LayerNorm / modulation 与 FC1 输入的 NVFP4 量化。"""
    # 硬门控：任何不满足的条件一律返回 None，由调用方走原路径，保证行为不漂移。
    # 兼容性逃生舱：FlashInfer / TRT-LLM 若被外部配置为禁用 FP4 fast-math 或
    # 使用 4-over-6 编码，这里必须回避，否则打包字节与生产量化器不一致。
    if (
        torch.compiler.is_compiling()
        or _env_enabled("FLASHINFER_DISABLE_FP4_QUANT_FAST_MATH")
        or _env_enabled("TRTLLM_DISABLE_FP4_QUANT_FAST_MATH")
        or _env_enabled("FLASHINFER_NVFP4_4OVER6")
    ):
        return None
    # 仅支持纯 LayerNorm（无仿射 weight/bias）、BF16 激活与 Blackwell SM10x。
    if norm_type != "layer" or weight is not None or bias is not None:
        return None
    if not (
        _nss_activation(x)
        and _nss_activation(residual, x)
        and _blackwell_sm10x(x.device)
    ):
        return None
    # CUDA 图捕获期间禁止在图中分配输出张量，融合路径直接退出。
    if torch.cuda.is_current_stream_capturing():
        return None
```

```
    gate = _row_bf16(gate, x.device)
    input_bias = _row_bf16(input_bias, x.device)
    scale = _row_bf16(scale, x.device)
    shift = _row_bf16(shift, x.device)
    if input_bias is None or gate is None or scale is None or shift is None:
        return None
    # global_scale 必须是 CUDA 上的 FP32 单元素连续张量（FC1 的 input_scale_inv）。
    if not (
        isinstance(global_scale, torch.Tensor)
        and global_scale.is_cuda
        and global_scale.device == x.device
        and global_scale.dtype == torch.float32
        and global_scale.numel() == 1
        and global_scale.is_contiguous()
    ):
        return None

    # 输出布局与 flashinfer.fp4_quantize 对齐：quantized 每行 1536 B，
    # quant_scales 行数按 128 填充（内核要求 kHidden=3072 特化）。
    rows = x.numel() // _HIDDEN
    padded_rows = (rows + 127) // 128 * 128
    quantized = torch.empty((rows, _HIDDEN // 2), dtype=torch.uint8, device=x.device)
    quant_scales = torch.empty(
        (padded_rows, _HIDDEN // 16), dtype=torch.uint8, device=x.device
    )
    residual_out = torch.empty_like(x)
    _nvfp4_module().srnss_nvfp4_row(
        quantized,
        quant_scales,
        residual_out.view(-1, _HIDDEN),
        residual.view(-1, _HIDDEN),
        x.view(-1, _HIDDEN),
        input_bias,
        gate,
        scale,
        shift,
        global_scale.reshape(1),
        float(eps),
    )
    return (quantized, quant_scales), residual_out
```

python/sglang/multimodal_gen/runtime/layers/quantization/modelopt_quant.py

新增 apply_nvfp4_gemm_prequantized，抽取 ModelOptFp4LinearMethod.apply 的 GEMM 段以直接消费已打包激活；apply 保持引用行为不变，普通线性调用不受影响。

```
  def apply_nvfp4_gemm_prequantized(
      layer: torch.nn.Module,
      x_fp4: torch.Tensor,
```

```
    x_scale_interleaved: torch.Tensor,
    output_dtype: torch.dtype,
    bias: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    """基于已打包 FP4 激活与 scale 直接执行 ModelOpt NVFP4 GEMM。

    与 ModelOptFp4LinearMethod.apply 共享同一套权重 padding、E4M3 视图转换与
    cutlass / FlashInfer 后端选择，只是跳过 apply 内部的 fp4_quantize 阶段，
    从而保证融合路径与常规量化路径在 GEMM 输出上逐位一致。
    """
    weights_padding_cols = getattr(layer, "weights_padding_cols", 0)
    x_fp4 = pad_nvfp4_activation_for_cutlass(x_fp4, weights_padding_cols)

    w = layer.weight
    w_scale_interleaved = layer.weight_scale_interleaved
    # scale 可能是 uint8 存储，统一转为 E4M3 视图再参与 GEMM。
    if x_scale_interleaved.dtype == torch.uint8:
        x_scale_interleaved = x_scale_interleaved.view(torch.float8_e4m3fn)
    if w_scale_interleaved.dtype == torch.uint8:
        w_scale_interleaved = w_scale_interleaved.view(torch.float8_e4m3fn)

    fp4_gemm, flashinfer_backend = _get_fp4_gemm_op()
    if fp4_gemm is None:
        raise RuntimeError("No FP4 GEMM kernel available. Install flashinfer.")
    out = fp4_gemm(
        x_fp4,
        w.T,
        x_scale_interleaved,
        w_scale_interleaved.T,
        layer.alpha,
        output_dtype,
        backend=flashinfer_backend,
    )
    # 切掉为 cutlass 对齐补的 padding 列，并把推迟的 bias 加回。
    out = slice_nvfp4_output(out, layer.output_size_per_partition)
    return out + bias if bias is not None else out
```

评论区精华

本 PR 没有代码 review 评论，讨论集中在 PR body 与 issue 评论：

1. 独立化决策：BBuf 在 issue 评论中说明已 restack 到当前 main，#37096 不是祖先，diff 不含 FLUX.2 融合代码，只保留窄的 apply_nvfp4_gemm_prequantized 助手与 Qwen 路径；GB300 上 12/12 NVFP4 测试、9/9 import-surface 测试通过。
2. 依赖隔离决策：PR body 声明 NVFP4 wrapper 使用独立 JIT 模块与可选内部头依赖，现有 BF16 norm JIT 不获得 FlashInfer 构建依赖——避免把重型依赖扩散到共享 kernel。
3. 一致性契约：复用 FlashInfer/TensorRT-LLM 生产量化 helper，保证 packed 值、scale 布局与 residual 输出字节精确，并用 PNG SHA256 相同、SSIM=1.0、PSNR=inf、像素

差 0/0 闭环验证。

4. CI 噪音：AMD ROCm 7.2 运行标红，BBuf 两次 /rerun-failed-ci 后合并，未留下失败根因说明。

- PR 独立化与 FLUX.2 代码剔除 (design): 以独立变更合并，支持与 FLUX.2 优化分开演化、独立验证。
- NVFP4 JIT 与 BF16 norm JIT 的依赖隔离 (design): 通过 norm_scale_shift_nvfp4_module + extra_dependencies=[flashinfer, flashinfer_nv_internal] 落地，BF16 norm JIT 依赖面不变。
- 字节级一致性与验证方法论 (testing): 新增两个 SM10x 测试固化契约；benchmark 对每个形状同时断言 exact。
- AMD ROCm 7.2 CI 失败 (other): 未记录根因；融合路径仅限 CUDA SM10x，AMD 失败大概率与功能无关，但结论缺乏证据支撑。

风险与影响

- 风险：
  1. JIT 依赖风险：norm_scale_shift_nvfp4_module 通过 extra_dependencies=["flashinfer", "flashinfer_nv_internal"] 加载，get_flashinfer_nv_internal_include_paths 在 data/csrc/nv_internal 缺失时 raise RuntimeError。若用户 FlashInfer wheel 不含该目录，在 SM10x + ModelOpt FP4 模型上会启动失败而非回退（所有 gate 检查位于模块加载之前）；环境变量可整体禁用，但默认路径仍暴露该风险。
  2. 字节一致性契约风险：打包布局与 128x4 scale swizzle 依赖 TensorRT-LLM cvt_warp_fp16_to_fp4 / get_sf_out_offset_128x4 与 FlashInfer 生产量化器契约；未来任一方调整布局，融合路径将静默产生不同字节，而测试仅覆盖 17 / 1024 行两种形状。
  3. 回归风险：forward 分支被重构（FP8 尝试移入 else，文本流顺序调整），逻辑等价依赖测试与代码审查；QwenImageGELU.forward 新增 tuple 分支，普通 tensor 路径不变；ModelOptFp4LinearMethod.apply 未改动。
  4. 覆盖局限：快路径仅在 eager、非 CUDA 图捕获、batch-1、非动态 CFG、纯 LayerNorm、SM10x + dim 3072 下生效，其余场景无收益但行为不变；torch.compile 下 kernel 直接禁用，PR 验证过 breakable CUDA graph 兼容性。 - 影响：用户侧：GB300 / B200 上运行 Qwen-Image-2512 ModelOpt NVFP4 的用户，denoise/step 中位数降约 0.84%（均值约 1.20%），端到端约 0.81%–1.19%；目标 norm+FC1 量化链 GPU 时间降 23.6%，每两步少 224 次 kernel launch（56 个量化 transformer block × 双流 × 两趟），输出与基线逐字节一致，无精度影响。系统侧：新增一个 lazily-loaded JIT 模块与一个新依赖定位器，仅 SM10x + FP4 场景触发构建；共享 norm_scale_shift.cuh 内核家族参数结构扩展，所有旧 launch 站点补齐字段，行为不变。团队侧：确立「融合 kernel + 字节级契约测试 + env 逃生舱 + 硬件 / 形状门控」的可复制模式，apply_nvfp4_gemm_prequantized 可供其他扩散模型复用预打包激活路径。 - 风险标记：SM10x 限定快路径，依赖 FlashInfer 内部头文件，字节一致性契约，CUDA 图捕获排除，环境变量逃生舱

关联脉络

- PR #37156 [Diffusion] Fuse Qwen-Image FP8 norm and activation quantization: 同属 qwen_image.py 的 norm+quant 融合系列；本 PR 的 forward 回退链在 _try_fp8_residual_norm_quant 之前新增 NVFP4 分支，并共享 norm_scale_shift.cuh 内核家族。
- PR #37144 [Diffusion] Fuse Qwen-Image final adaptive LayerNorm: 同一模型文件的 adaLN 融合先行工作，确立 BitExactFusionGate 与字节级自校验方法，本 PR 延续该模式到 NVFP4 量化路径。
- PR #37141 [Diffusion] Fuse FLUX.2 token concatenation and NVFP4 quantization: 同一 NVFP4 融合项目线；PR body 声明本 PR 不含、不依赖、不改动 #37096 的 FLUX.2 优化，已 restack 为独立变更，两者可分开演进。