# PR #35703 完整报告

sgl-project/sglang

[diffusion] fix: fix loading a block-FP8 quantized MiniMax-H3 DiT

合并时间：2026-09-01 10:33

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35703

## 执行摘要

- 一句话：修复 MiniMax-H3 DiT block-FP8 加载静默产出空白帧
- 推荐动作：值得精读。这是一个经典的量化元数据与权重布局一致性缺陷：根因定位链条完整（布局契约 → 行数门控失效 → 块计数），修复通过类型判断（BlockQuantScaleParameter）与块计数缩小重排粒度，并对无法修复的配置显式 fail-fast。实现与测试共同定义了可复用的加载契约，对后续支持其他 DiT 模型或其他 block 量化格式的 checkpoint 加载器有直接借鉴意义。

## 功能与动机

block-FP8 MiniMax-H3 DiT 加载时无任何报错——dtype、shape 全部匹配，Detected fp8 checkpoint 正常打印，服务就绪且推理速度正常——但生成的是空白帧。根因是 H3 checkpoint 按 head 分组存储 qkv 行（[h0_q, h0_k, h0_v, h1_q, ...]），DiT 在加载时已将它们重排为 [q_all, k_all, v_all]，而前置 PR #35740 的行数门控对 block-FP8 的 weight_scale_inv 失效：门控按 21504（权重行数）对比，scale 的 leading dim 只有 168（= 56 heads × 3），每个 scale 行覆盖 128 个权重行，导致 scale 未经重排直接通过。作者强调这不是任何单一量化器的问题：scale 按 checkpoint 布局产出是合理的，应用重排是运行时职责，任何标准 block-FP8 H3 checkpoint 都会踩中。

## 实现拆解

### 1. 定位根因并确定修复入口

入口是 _install_qkv_weight_loader（python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py）。关键判断：block-FP8 的 scale 也是按输出行索引的，只是索引粒度是块而非行——一个 scale 行覆盖 weight_block_size[0] 个权重行。因此重排操作本身可复用，但重排的 head_dim 参数和行数门控都要按块高缩小。

### 2. 新增 _qkv_scale_block_rows

新函数从 qkv_proj.quant_method.quant_config.weight_block_size[0] 读取块高：无块配置（BF16、per-tensor FP8、NVFP4、在线量化）返回 1，保持原有每权重行一行 scale 的语义；head_dim % block_rows != 0 时抛 ValueError——块若横跨两个 head 的 q/k/v 行，任何行置换都无法修复，必须显式拒绝而非静默错缩放。

### 3. 参数化重排并安装到各入口

把原先固定的 _reorder_checkpoint_weight 改为 _make_row_reorder(head_dim) 工厂闭包。在遍历 qkv 投影的量化参数时：BlockQuantScaleParameter 类型参数（block-FP8 的 weight_scale_inv）按 head_dim // block_rows 重排，门控行数缩放为 qkv_rows // block_rows；其余参数（含权重本身、逐通道 scale）保持原重排。param._weight_loader / param.weight_loader（加载器包装）与 param.rank_local_weight_transform（FSDP 切分前）双入口同步更新，保证 CPU 加载与 TP/FSDP 路径行为一致。GGUF 路径原本就存 [q_all, k_all, v_all]，被行数门控排除，不受影响。

### 4. 测试配套

test_minimax_h3_dit_contract.py 新增 136 行、5 个用例与 2 个 helper（_block_fp8_quant_config、_meta_h3）：

| 用例 | 覆盖点 |
| --- | --- |
| test_offline_block_fp8_checkpoint_layout_and_cpu_load | 260 个 FP8 权重与 scale 一一配对、scale 形状为 ceil(n/128)×ceil(k/128)、离线权重无需设备后处理 |
| test_block_fp8_qkv_scale_follows_its_weight_through_the_grouped_reorder | 模拟量化器在 grouped 布局下算 scale，验证重排后与权重 tile 的 max 值严格一致 |
| test_qkv_block_scale_param_is_reordered_in_blocks | 用 rank_local_weight_transform 作为契约入口验证已安装的变换 |
| test_unquantized_and_per_tensor_qkv_keep_their_loaders | 无量化与 per-tensor FP8 路径行为不变、_qkv_scale_block_rows 返回 1 |
| test_block_that_straddles_heads_is_rejected | block=256 时抛出 "divides the head dim" 错误 |

### 5. 端到端与性能验证

2× RTX 5090、TP=2、layerwise offload、seed 42/10 步 /768p 下：无修复时帧内对比度塌缩 16 倍、帧间运动塌缩 19 倍；修复后输出正常，PSNR-Y 16.5 dB / SSIM-Y 0.68 贴近 BF16 参考。本 PR 不改任何计算路径，block-FP8 对比 BF16 在 offload 下从 104.7s 降到 70.4s（1.49 倍）。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py（模块 DiT 模型；类别 source；类型 data-contract；符号 _qkv_scale_block_rows, _reorder_checkpoint_weight, _make_row_reorder, _reorder）：核心修复文件：新增 _qkv_scale_block_rows 与 _make_row_reorder，让 block-FP8 的 weight_scale_inv 按块计数并跟随 qkv 行重排，同时拒绝跨 head 的块配置。
- python/sglang/multimodal_gen/test/unit/test_minimax_h3_dit_contract.py（模块 契约测试；类别 test；类型 test-coverage；符号 _block_fp8_quant_config, _meta_h3,

test_offline_block_fp8_checkpoint_layout_and_cpu_load,test_block_fp8_qkv_scale_follows_its_weight_through_the_grouped_reorder）：新增 5个测试覆盖离线 block-FP8 布局契约、scale 随权重重排、非块量化路径不变与跨 head 块拒绝，是修复正确性的主要保障。

关键符号：_qkv_scale_block_rows, _make_row_reorder, _install_qkv_weight_loader, _reorder

## 关键源码片段

### python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py

核心修复文件：新增 _qkv_scale_block_rows 与 _make_row_reorder，让 block-FP8 的 weight_scale_inv 按块计数并跟随 qkv 行重排，同时拒绝跨 head 的块配置。

```
def _qkv_scale_block_rows(qkv_proj: nn.Module, head_dim: int) -> int:
    """返回 qkv 投影的 scale 张量每一行覆盖的权重行数。

    逐通道量化与 NVFP4 的 scale 每个权重行对应一行，返回 1；block-FP8 的
    scale 每个权重块（weight_block_size[0] 行）才对应一行，所以 qkv 行置换
    对 scale 需要按块计数。只有完整块才能移动，若块跨越两个 head 的 q/k/v
    行则无法用行置换修复，直接抛错而不是静默错缩放。
    """
    quant_config = getattr(
        getattr(qkv_proj, "quant_method", None), "quant_config", None
    )
    block_size = getattr(quant_config, "weight_block_size", None)
    if not block_size:
        # 无块配置（BF16、per-tensor、NVFP4）时每权重行一行 scale
        return 1
    block_rows = block_size[0]
    if head_dim % block_rows:
        # 块无法整除 head_dim 意味着块会跨 head，行置换无法修复
        raise ValueError(
            "block-quantized qkv needs a block size that divides the head dim: "
            f"head_dim={head_dim}, weight_block_size={block_size}."
        )
    return block_rows

def _install_qkv_weight_loader(self, arch: MiniMaxH3DiTArchConfig) -> None:
    weight = self.qkv_proj.weight
    # H3 checkpoint 按 head 交织存放 Q/K/V 行，而 QKVParallelLinear 期望
    # [q_all, k_all, v_all]，加载时必须重排
    weight.checkpoint_mapping_unsafe = True
    base_loader = weight.weight_loader

    # 工厂按 head_dim 参数化重排闭包：权重按原始 head_dim 重排，
    # block-FP8 的 scale 按 head_dim // block_rows 重排，结构同构
    def _make_row_reorder(head_dim: int) -> Callable[[torch.Tensor], torch.Tensor]:
        def _reorder(loaded_weight: torch.Tensor) -> torch.Tensor:
```

```
            return _reorder_grouped_qkv_to_qkv(
                loaded_weight,
                num_query_groups=arch.num_attention_heads,
                heads_per_group=1,
                head_dim=head_dim,
            )
        return _reorder

    _reorder_checkpoint_weight = _make_row_reorder(arch.attention_head_dim)

    def _weight_loader(param: torch.Tensor, loaded_weight: torch.Tensor) -> None:
        # TP 直拷路径：稠密 MHA checkpoint 直接写入 TP 本地 Q/K/V 行
        if _copy_grouped_qkv_tp_shard(
            param, loaded_weight,
            num_query_groups=arch.num_attention_heads,
            head_dim=arch.attention_head_dim,
            tp_rank=self.qkv_proj.tp_rank, tp_size=self.tp_size,
        ):
            return
        base_loader(param, _reorder_checkpoint_weight(loaded_weight))

    if hasattr(weight, "_weight_loader"):
        weight._weight_loader = _weight_loader
    else:
        weight.weight_loader = _weight_loader
    # rank-local FSDP 也必须先重排 grouped QKV 再切分 shard
    weight.rank_local_weight_transform = _reorder_checkpoint_weight

    # 量化 checkpoint 在权重旁存放按输出行索引的元数据（NVFP4 block scale、
    # fp8 per-channel scale）。权重行被置换后，逐行元数据必须同步置换。
    # block-FP8 的 scale 按块而非按行计数：每个块一行，置换与行数门控
    # 都按块高缩放。
    qkv_rows = 3 * arch.num_attention_heads * arch.attention_head_dim
    block_rows = _qkv_scale_block_rows(self.qkv_proj, arch.attention_head_dim)
    for name, param in self.qkv_proj.named_parameters(recurse=False):
        if name == "weight":
            continue
        # 仅 BlockQuantScaleParameter 按块计数；per-tensor 与逐通道 scale
        # 保持每权重行一行，走原有路径
        rows_per_scale_row = (
            block_rows if isinstance(param, BlockQuantScaleParameter) else 1
        )
        _install_qkv_row_reorder(
            param,
            _make_row_reorder(arch.attention_head_dim // rows_per_scale_row),
            qkv_rows // rows_per_scale_row,
        )
```

python/sglang/multimodal_gen/test/unit/test_minimax_h3_dit_contract.py

新增 5 个测试覆盖离线 block-FP8 布局契约、scale 随权重重排、非块量化路径不变与跨 head 块拒绝，是修复正确性的主要保障。

```
def test_qkv_block_scale_param_is_reordered_in_blocks():
    model = _meta_h3(_block_fp8_quant_config())
    arch = model.arch
    qkv = model.blocks[0].attn.qkv_proj
    block = 128
    # head_dim 与 block 都是 128，每个 head 的 q/k/v 恰占一个完整 scale 行
    scale_rows = 3 * arch.num_attention_heads * arch.attention_head_dim // block
    assert qkv.weight_scale_inv.shape[0] == scale_rows

    # rank_local_weight_transform 是安装后的契约入口：weight_loader 包装后
    # 先执行它再交给 base_loader，FSDP 分片前也会执行它
    loaded = torch.arange(scale_rows * 4, dtype=torch.float32).reshape(scale_rows, 4)
    expected = _reorder_grouped_qkv_to_qkv(
        loaded,
        num_query_groups=arch.num_attention_heads,
        heads_per_group=1,
        head_dim=arch.attention_head_dim // block,
    )
    got = qkv.weight_scale_inv.rank_local_weight_transform(loaded)
    assert torch.equal(got, expected)
    assert not torch.equal(got, loaded)

    # 权重本身保持逐行置换，不受块计数影响
    unblocked = torch.zeros(scale_rows * block, 4)
    assert qkv.weight.rank_local_weight_transform(unblocked).shape == unblocked.shape
```

## 评论区精华

评审流程简短：mickqian 直接 APPROVED（无附带评论），issue 评论区只有 CI 操作（/tag-and-rerun-ci、/rerun-failed-ci）、请求审核与致谢，没有展开技术辩论。作者把技术论证完整放在 PR body 中，核心观点值得引用：

> block-FP8 checkpoint 携带的 weight_scale_inv 是每个 128×128 tile 一个 scale，#35740 的行数门控按 21504（权重行数）对比，而 scale 的 leading dim 只有 168，导致 scale 未经重排直接通过。
>
> 这不是任何单一量化器的问题：scale 按 checkpoint 布局产出是合理的，应用重排是运行时的职责，任何标准 block-FP8 H3 checkpoint 都会踩中。
>
> 无修复时帧内对比度塌缩 16 倍、帧间运动塌缩 19 倍；修复后 PSNR-Y 16.5 dB / SSIM-Y 0.68，贴近 BF16 参考，残差来自 FP8 舍入导致的采样轨迹分叉而非损坏。

- 暂无高价值评论线程

## 风险与影响

- 风险：数据契约变更：block-FP8 的 weight_scale_inv 加载行为从 " 原样传入 " 变为 " 按块重排 "，这是本次修复的核心，属于预期内契约修正，但与 #35740 的逐行元数据行为保持一致。新增硬校验：head_dim % block_rows != 0 会拒绝此前能 " 静默加载 " 的配置（如 block=256），这是有意为之的 fail-fast，但若未来出现合法的大块配置需重新审视该限制。多卡路径覆盖有限：新测试在单进程 TP=1 环境运行，TP/FSDP 多卡路径依赖 rank_local_weight_transform 既有契约，未单独多卡验证；不过该入口与 weight_loader 包装共享同一实现，风险可控。契约依赖：新逻辑依赖 quant_config.weight_block_size 约定与 BlockQuantScaleParameter 类型判断，若未来新增返回块布局但参数类型不同的量化方法，可能绕过重排而静默损坏。
- 影响：影响面收敛在 MiniMax-H3 DiT 的 block-FP8 离线 checkpoint 加载路径：BF16、在线 FP8、per-tensor FP8、NVFP4 与 GGUF 路径经门控和测试确认为不变。对用户而言，直接收益是 block-FP8 权重从 " 不可用 " 变为 " 可用且更快 "：layerwise offload 下 70.4s 对比 BF16 的 104.7s（1.49 倍），对长视频生成场景有明显成本价值。对团队而言，本 PR 与 #35740 共同确立了 " 量化元数据必须跟随权重行重排 " 的加载契约，并提供了拒绝无法修复配置的范式。
- 风险标记：数据契约变更，块尺寸硬校验，TP/FSDP 覆盖有限，依赖量化配置约定

## 关联脉络

- PR #35740 （推断）[Fix] Make row-indexed quant metadata follow the qkv row reorder: PR body 明确引用 #35740 引入的 _install_qkv_row_reorder 行重排基础设施，本 PR 修复其行数门控对 block-FP8 scale（leading dim = 168）失效的缺口；两个 PR 都修改 minimax_h3.py 的 qkv 加载链。标题基于 PR body 语义推断，原标题未在材料中提供。