# PR #35994 完整报告

sgl-project/sglang

[diffusion] Load serialized Comfy ConvRot INT8 DiTs

合并时间：2026-08-23 09:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35994

## 执行摘要

- 一句话：加载序列化 Comfy INT8 DiT，复用 Kitchen 融合内核
- 推荐动作：值得精读。几个值得关注的设计决策： 1) 不引入新 CLI flag，通过 layer_markers 自动检测与现有 config/method 复用完成双路径分派； 2) 对 checkpoint 数据契约做严格前置校验，坏权重在模型构造前被拒绝而不是静默产出错误； 3) 用基类字段 checkpoint_uses_native_qkv_layout 表达 QKV 布局契约，避免硬编码模型名； 4) PR body 中通过 ConvRot 行范数不变性做数值溯源验证 QKV 布局的方法非常巧妙。建议结合 #36023 看同一契约在编码器侧的落地。

## 功能与动机

PR body 明确这是 #35979 之后的“next small slice”：通用组件加载路径已能解析本地文件与 Hub 文件引用，本 PR 补上第一个真实的序列化物化后端。动机有三层：

1) 省去在线重量化开销——现有 --quantization kitchen_int8 需要先物化完整 BF16 DiT 再逐层量化，加载已序列化 checkpoint 可直接跳过； 2) 修复静默错误——PR body 用数值实验证明 Comfy 张量已按 [q_all, k_all, v_all] 排列，“applying H3's official grouped-QKV reorder again would silently corrupt conditioning rather than necessarily crash”； 3) 确立格式契约——文档新增 comfy-int8-convrot 条目，明确 TP 限制、FSDP 拒绝与已验证的 H3 用法。

## 实现拆解

1. 配置层双路径分派（kitchen_int8_config.py）：KitchenInt8Config 新增 layer_markers 参数，构造时逐层校验 format=int8_tensorwise、convrot=true、convrot_groupsize，并据此设置 is_checkpoint_int8_serialized 与 checkpoint_uses_native_qkv_layout。get_quant_method 按 marker 分派 serialized 或 online 的 KitchenInt8LinearMethod，未标记层回退 UnquantizedLinearMethod。
2. 线性方法支持序列化权重（kitchen_int8.py）：create_weights 对 serialized 路径直接分配 INT8 权重并注册 F32 [out_features, 1] 的 weight_scale；process_weights_after_loading 对序列化权重直接跳过在线量化；前向 apply 保持融合 ConvRot 内核不变。
3. Checkpoint 契约校验（minimax_h3_weights.py）：inspect_minimax_h3_safetensors 开始收集所有 weight/weight_scale 的 dtype 与形状，对 int8_tensorwise 层校验 I8 权重、

F32 缩放、2D 形状、scale 形状 [out, 1]，不匹配即在模型构造前抛 ValueError。resolve_minimax_h3_checkpoint_quantization 不再对 INT8 抛 NotImplementedError，改为返回 KitchenInt8Config。

4. QKV 布局修复（base_config.py、comfy_fp8.py、minimax_h3.py）：QuantizationConfig 基类新增 checkpoint_uses_native_qkv_layout=False，ComfyFp8Config 与 serialized Kitchen 置为 True；H3 模型据此决定是否安装官方 grouped-QKV 重排 loader。
5. 加载器与配套（transformer_load_utils.py、transformer_loader.py）：TransformerQuantLoadSpec 新增 is_serialized_kitchen_int8 与 uses_comfy_layer_markers；_needs_device_weight_postprocess 对 serialized kitchen 返回 False；FSDP 拒绝与 .comfy_quant key 过滤从仅 FP8 扩展到所有 comfy marker 路径，serialized INT8 允许 CPU 加载。文档同步更新至 docs/docs/sglang-diffusion/quantization.mdx 与 MiniMax-H3 cookbook。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/quantization/kitchen_int8.py（模块 量化内核；类别 source；类型 core-logic；符号 init, create_weights, process_weights_after_loading）：核心线性方法改造：KitchenInt8LinearMethod 新增 group_size 与 is_checkpoint_serialized 参数，serialized 路径直接分配 INT8 权重与 F32 行缩放并跳过在线量化，前向仍走融合 ConvRot 内核。
- python/sglang/multimodal_gen/runtime/layers/quantization/configs/kitchen_int8_config.py（模块 量化配置；类别 source；类型 core-logic；符号 init, get_quant_method）：配置分派核心：KitchenInt8Config 新增 layer_markers 参数并校验序列化契约，get_quant_method 按 marker 分派 serialized 或 online 量化方法。
- python/sglang/multimodal_gen/runtime/loader/minimax_h3_weights.py（模块 权重契约；类别 source；类型 data-contract；符号 inspect_minimax_h3_safetensors, resolve_minimax_h3_checkpoint_quantization）：Checkpoint 契约校验：inspect_minimax_h3_safetensors 收集权重元数据并对 int8_tensorwise 层做 I8/F32/ 形状校验，resolve_minimax_h3_checkpoint_quantization 改为返回 KitchenInt8Config。
- python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py（模块 模型定义；类别 source；类型 data-contract；符号 init）：QKV 布局修复落点：按 checkpoint_uses_native_qkv_layout 决定是否安装官方 grouped-QKV 重排 loader，Comfy/GGUF checkpoint 不再被错误重排。
- python/sglang/multimodal_gen/runtime/loader/transformer_load_utils.py（模块 加载规格；类别 source；类型 core-logic；符号 is_serialized_kitchen_int8, uses_comfy_layer_markers, _needs_device_weight_postprocess）：加载规格扩展：TransformerQuantLoadSpec 新增 is_serialized_kitchen_int8 与 uses_comfy_layer_markers，_needs_device_weight_postprocess 对 serialized kitchen 返回 False。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/transformer_loader.py（模块 加载器；类别 source；类型 core-logic；符号 load_customized）：加载器分派：FSDP 拒绝与 .comfy_quant key 过滤从仅 FP8 扩展到所有 comfy marker 路径，并允许

serialized INT8 CPU 加载。

- python/sglang/multimodal_gen/test/unit/test_transformer_quant.py（模块 量化测试；类别 test；类型 test-coverage；符号 test_minimax_h3_comfy_int8_resolves_serialized_kitchen, test_serialized_kitchen_constructs_int8_weight_and_row_scale, test_serialized_kitchen_rejects_non_convrot_marker）：测试配套：覆盖 serialized kitchen INT8 的解析、参数构造、非 convrot 拒绝，并扩展 FP8 native QKV 断言。
- python/sglang/multimodal_gen/runtime/layers/quantization/comfy_fp8.py（模块 FP8 量化；类别 source；类型 core-logic；符号 checkpoint_uses_native_qkv_layout）：标记 Comfy FP8 checkpoint 为 native QKV 布局，与 INT8 路径共用同一修复。
- python/sglang/multimodal_gen/runtime/layers/quantization/configs/base_config.py（模块 配置基类；类别 source；类型 data-contract；符号 checkpoint_uses_native_qkv_layout）：配置基类新增 checkpoint_uses_native_qkv_layout 字段，为所有量化配置提供布局契约默认值。
- docs/docs/sglang-diffusion/quantization.mdx（模块 量化文档；类别 docs；类型 documentation）：文档定义 comfy-int8-convrot 格式契约、TP 限制与 Kitchen INT8 双路径用法。
- docs/cookbook/diffusion/MiniMax/MiniMax-H3.mdx（模块 模型文档；类别 docs；类型 documentation）：cookbook 更新序列化 INT8 checkpoint 的启动命令与验证说明。

关键符号：KitchenInt8LinearMethod.init, KitchenInt8LinearMethod.create_weights, KitchenInt8LinearMethod.process_weights_after_loading, KitchenInt8Config.init, KitchenInt8Config.get_quant_method, inspect_minimax_h3_safetensors, resolve_minimax_h3_checkpoint_quantization, TransformerQuantLoadSpec.is_serialized_kitchen_int8, TransformerQuantLoadSpec.uses_comfy_layer_markers, _needs_device_weight_postprocess

## 关键源码片段

### python/sglang/multimodal_gen/runtime/layers/quantization/kitchen_int8.py

核心线性方法改造：KitchenInt8LinearMethod 新增 group_size 与 is_checkpoint_serialized 参数，serialized 路径直接分配 INT8 权重与 F32 行缩放并跳过在线量化，前向仍走融合 ConvRot 内核。

```
# KitchenInt8LinearMethod 同时服务两条路径：
# 1) online：BF16 checkpoint + `--quantization kitchen_int8`，加载后在线量化；
# 2) serialized：Comfy 导出的 INT8 权重 + F32 行缩放，加载后直接使用。
# 两条路径的差异集中在权重分配与后处理，前向统一走融合 ConvRot 内核。
def create_weights(
    self,
    layer: torch.nn.Module,
    input_size_per_partition: int,
    output_partition_sizes: list[int],
    input_size: int,
    output_size: int,
```

```
    params_dtype: torch.dtype,
    **extra_weight_attrs,
) -> None:
    # get_quant_method 已筛过未分片输入尺寸；TP > 1 时 row-parallel
    # 层会切分旋转所基于的维度，分片尺寸必须能被组大小整除
    if input_size_per_partition % self.group_size:
        raise ValueError(
            f"kitchen_int8 needs input_size_per_partition "
            f"({input_size_per_partition}) divisible by group_size "
            f"{self.group_size}"
        )

    # 在线路径先按 UnquantizedLinearMethod 分配 BF16 占位，加载后再量化；
    # serialized 路径直接分配 INT8 最终形态，省去物化 BF16 再重量化
    weight = Parameter(
        torch.empty(
            sum(output_partition_sizes),
            input_size_per_partition,
            dtype=(torch.int8 if self.is_checkpoint_serialized else params_dtype),
        ),
        requires_grad=False,
    )
    set_weight_attrs(weight, {"input_dim": 1, "output_dim": 0})
    layer.register_parameter("weight", weight)
    set_weight_attrs(weight, extra_weight_attrs)

    # serialized checkpoint 自带每输出通道 F32 行缩放 [out_features, 1]；
    # 在线路径的缩放由 process_weights_after_loading 量化后生成
    if self.is_checkpoint_serialized:
        weight_scale = Parameter(
            torch.empty(
                sum(output_partition_sizes),
                1,
                dtype=torch.float32,
            ),
            requires_grad=False,
        )
        set_weight_attrs(weight_scale, {"output_dim": 0})
        set_weight_attrs(weight_scale, extra_weight_attrs)
        layer.register_parameter("weight_scale", weight_scale)


def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
    weight = layer.weight.data
    # serialized 权重已是最终形态，直接跳过在线量化；
    # online 路径若已被处理过（weight 已变为 int8）也直接返回
    if self.is_checkpoint_serialized or weight.dtype == torch.int8:
        return
```

```
from comfy_kitchen.tensor.int8 import TensorWiseINT8Layout

# 在线量化在 CUDA 上进行，但模型可能仍暂存在 CPU（offload 场景）；
# 逐层往返而不是等 loader 整体搬移，避免一次放不下全部权重
home = weight.device
qdata, params = TensorWiseINT8Layout.quantize(
    weight.to("cuda", non_blocking=True),
    is_weight=True,
    per_channel=True,
    convrot=True,
    convrot_groupsize=self.group_size,
    stochastic_rounding=0,
)
layer.weight = Parameter(qdata.to(home), requires_grad=False)
layer.register_parameter(
    "weight_scale",
    Parameter(
        params.scale.to(device=home, dtype=torch.float32),
        requires_grad=False,
    ),
)
self.quant_config.note_quantized(weight.numel() * weight.element_size())
```

## python/sglang/multimodal_gen/runtime/loader/minimax_h3_weights.py

Checkpoint 契约校验：inspect_minimax_h3_safetensors 收集权重元数据并对 int8_tensorwise 层做 I8/F32/ 形状校验，resolve_minimax_h3_checkpoint_quantization 改为返回 KitchenInt8Config。

```
# inspect_minimax_h3_safetensors 的 INT8 契约校验部分：
# 前置逻辑已收集所有 .weight / .weight_scale 的 dtype 与形状到 checkpoint_meta，
# 并对 comfy_quant marker 做了 JSON 解析与冲突检查。下面按 marker 做最终校验，
# 任何不匹配都在模型构造前抛 ValueError，避免坏权重静默产出错误结果。
for prefix, marker in layer_markers.items():
    marker_format = marker.get("format")
    required = {f"{prefix}.weight", f"{prefix}.weight_scale"}
    # FP8 且非 full_precision 的层还需要 input_scale（此处略）
    if marker_format not in ("float8_e4m3fn", "int8_tensorwise"):
        continue
    missing = required - checkpoint_keys
    if missing:
        raise ValueError(
            f"MiniMax-H3 Comfy layer {prefix!r} is missing checkpoint "
            f"tensors: {sorted(missing)}"
        )
    if marker_format == "int8_tensorwise":
        weight_dtype, weight_shape = checkpoint_meta[f"{prefix}.weight"]
        scale_dtype, scale_shape = checkpoint_meta[f"{prefix}.weight_scale"]
        # 契约：I8 权重 + F32 行缩放，权重必须 2D，
        # scale 形状必须严格等于 [out_features, 1]
```

```
        if weight_dtype != "I8" or scale_dtype != "F32":
            raise ValueError(
                f"MiniMax-H3 Comfy INT8 layer {prefix!r} needs I8 weights "
                f"and F32 scales, got {weight_dtype} and {scale_dtype}"
            )
        if len(weight_shape) != 2:
            raise ValueError(
                f"MiniMax-H3 Comfy INT8 layer {prefix!r} needs a 2D weight, "
                f"got {weight_shape}"
            )
        expected_scale_shape = (weight_shape[0], 1)
        if scale_shape != expected_scale_shape:
            raise ValueError(
                f"MiniMax-H3 Comfy INT8 layer {prefix!r} needs scale shape "
                f"{expected_scale_shape}, got {scale_shape}"
            )
```

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

QKV 布局修复落点：按 checkpoint_uses_native_qkv_layout 决定是否安装官方 grouped-QKV 重排 loader，Comfy/GGUF checkpoint 不再被错误重排。

```
# MiniMax H3 注意力模块构造：qkv_proj 使用 MergedColumnParallelLinear，
# checkpoint 中存储单个融合 qkv 张量。官方 safetensors 按 head 交错 Q/K/V，
# 需要自定义 loader 重排；Comfy 与 GGUF checkpoint 已按
# [q_all, k_all, v_all] 排列，若继续套官方重排会静默破坏条件编码。
checkpoint_qkv_is_native = quant_config is not None and (
    quant_config.get_name() == "gguf"
    or quant_config.checkpoint_uses_native_qkv_layout
)
if not checkpoint_qkv_is_native:
    self._install_qkv_weight_loader(arch)
```

## 评论区精华

本 PR 无实质 review 讨论（review_comments_count=0），唯一 issue 评论是 mintlify[bot] 的文档预览部署通知。PR body 自带详尽的 checkpoint 数值验证说明：官方 Comfy-Org pruned FL2VA DiT 含 200 个 int8_tensorwise 标记层，每层均为 I8 2D 权重 + F32 [out_features, 1] 缩放 + convrot=true + convrot_groupsize=256；Ref2VA 文件契约相同。PR body 还利用 ConvRot 保持输出行范数的特性，将 Comfy 行号与官方 BF16 权重源行一一对应（如 Comfy row 128 -> official row 384，norm error 0.00010），证明 QKV 已是 native 布局。

- 暂无高价值评论线程

## 风险与影响

- 风险：

1. 数据契约依赖：全路径依赖 Comfy marker 的规范性，本 PR 对 format、convrot、convrot_groupsize、权重 dtype/ 形状、scale 形状均做前置 fail-fast 校验，不匹配即拒绝加载；但 Comfy 生态若出现未声明的新格式仍需后续跟进。
2. TP 限制：TP8 被显式拒绝，因为 row-parallel 输入分片会切开 256 元素的旋转组边界，用户只能使用 TP1/2/4 并配合 --ulysses-degree 补足并行度。
3. QKV 布局是基类级约定：checkpoint_uses_native_qkv_layout 挂在 QuantizationConfig 上默认 False，未来新量化格式若为 native 布局但漏设该标志，H3 会错误套用官方 grouped 重排。
4. CI 状态：主 CI 通过；Extra 测试失败、AMD ROCm 为 pending；PR body 自述 runtime 测试未在本地运行，NVIDIA 覆盖由 PR CI 提供。
5. 回归面：改动集中在 multimodal_gen 的 MiniMax-H3 加载路径，在线 BF16 + kitchen_int8 路径保持兼容，SRT 核心不受影响。 - 影响：用户影响：MiniMax-H3 用户可直接加载 Comfy-Org 的 pruned INT8 ConvRot DiT（FL2VA E2E 验证、Ref2VA 契约验证），省去本地重新量化；TP1/2/4 可用，TP8 被拒，FSDP 不可用。 系统影响：变更限于 multimodal_gen 下 MiniMax-H3 的量化加载链路，前向内核未改动，性能特征与在线路径一致；序列化加载省去 BF16 物化与重量化开销。 团队影响：建立了可复用的序列化量化 checkpoint 契约（marker 校验 + native QKV 布局标志），后续其他扩散模型可沿用；同时文档明确了格式边界。

- 风险标记：数据契约强校验，fail-fast 拒绝，TP8 被拒（旋转组边界），QKV 布局依赖基类新字段，Extra CI 未通过

## 关联脉络

- PR #36023 [diffusion] Load serialized Comfy ConvRot INT8 native encoders: 同一 Comfy INT8 序列化系列在编码器侧的落地，复用同一 marker 契约与量化工具链。
- PR #36060 [Diffusion] Infer Comfy FP8 activation scaling: Comfy 序列化量化加载支持线的一部分，处理 FP8 激活缩放的推断。
- PR #36067 [Diffusion] Load Diffusers MiniMax H3 components natively: MiniMax H3 组件原生加载支持，涵盖 QKV 融合与 SwiGLU 重排，与本 PR 同模型同链路。
- PR #36080 [Diffusion] Support hybrid MiniMax H3 conditioning: MiniMax H3 流水线能力增强，与本 PR 同属 MiniMax H3 功能演进线。