PR #30319 完整报告

sgl-project/sglang

[NPU] Add mxfp4-w4a4 MOE Quantization Support for NPU

合并时间：2026-08-19 00:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30319

执行摘要

- 一句话：NPU 新增 ModelSlim W4A4 MXFP4 MoE 量化路径
- 推荐动作：值得 NPU 量化相关开发者精读。本 PR 展示了三个值得借鉴的设计决策：一是在 dispatcher 能力受限时选择 BF16 dispatch + 紧邻 GMM 前动态量化的延迟量化策略；二是通过 use_mx_quant 开关把 FP4 这类 torch dtype 对象无法直接识别的 MX 数据类型收敛进公共量化器；三是面对性能优化建议时以 profiling 数据为依据、明确拒绝并建议后续独立 PR，保持改动范围可控。

功能与动机

PR body 明确说明：SGLang 已在 Ascend NPU 上支持 dense 线性层的 ModelSlim W4A4_MXFP4 量化，但对应的 MoE 路径缺失，导致专家权重以 W4A4_MXFP4 方案导出的 MoE 模型无法通过 ModelSlim 量化后端加载和执行。同时 Ascend MoE 实现已重构为模块化 AscendRunner 架构，因此 MXFP4 MoE 支持需要集成新的 per-weight-group 量化方法和 runner 流程，而不是恢复旧的 fused MoE 实现。

实现拆解

变更从量化方案注册开始，到权重创建、权重后处理、动态激活量化和文档配套共五步：

1. 方案注册与导出：在 python/sglang/srt/layers/quantization/modelslim/modelslim.py 的 get_moe_scheme 的 moe_quant_schemes 列表中新增 ("W4A4_MXFP4", ModelSlimW4A4MXFP4MoE)，并在 schemes/__init__.py 中导入和加入 __all__，使 MoE 量化检测能识别 W4A4_MXFP4 权重组。这是整个功能的检测入口。
2. 权重创建（数据契约）：新增 python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_w4a4_mxfp4_moe.py，按 w13/w2 两个权重组分别创建参数：权重张量用 torch.uint8 容器（对应 checkpoint 中 FP4 值装在 float8_e4m3fn 容器），权重 scale 用 uint8 的 UE8M0 block scale（block size 32），并通过 FusedMoeWeightScaleSupported.BLOCK 标记量化是 block 级。
3. 权重后处理与布局转换：NPUW4A4MXFP4MoEMethod.process_weights_after_loading 将 FP4 权重通过 npu_format_cast 转成 NPU 需要的 NZ 布局并转置，将 block scale reshape 为 [E, K//32, N, 2] 供 grouped matmul 使用；由于重构后的 Ascend dispatcher 只支持 BF16/INT8 输出，w13 的 dispatcher 输出显式固定为 BF16。

4. 动态激活量化：python/sglang/srt/hardware_backend/npu/moe/quant.py 的 HiddenStatesDynamicQuant 新增 use_mx_quant 参数，使 FP4 等 MX dtype 也能选择 torch.ops.npu.npu_dynamic_mx_quant；apply 中在每次 expert GMM 之前把 BF16 激活动态量化成 MXFP4，并组装 scale/per_token_scale/x_dtype/weight_dtype 参数调用 GroupedMatmul.forward。
5. 文档配套：在 docs/docs/hardware-platforms/ascend-npus/optimization/quantization.mdx 增加支持矩阵行、示例命令与实现说明，并在 docs/docs/advanced_features/quantization.mdx 更新支持范围和 ModelSlim 支持清单。测试方面：该 PR 未新增任何单元测试文件，精度验证以 end-to-end eval 形式在 PR body 中给出。

关键文件：

- python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_w4a4_mxfp4_moe.py（模块 量化方案；类别 source；类型 data-contract；符号 ModelSlimW4A4MXFP4MoE, create_weights, process_weights_after_loading）：新增的 ModelSlim W4A4_MXFP4 MoE 方案类，是权重创建的入口，定义 w13/w2 权重组的参数形状与 block scale 契约。
- python/sglang/srt/hardware_backend/npu/quantization/moe_methods.py（模块 MoE 算子；类别 source；类型 core-logic；符号 NPUW4A4MXFP4MoEMethod, process_weights_after_loading, apply）：NPUW4A4MXFP4MoEMethod 是本功能的核心执行逻辑：权重布局转换、block scale reshape、dispatcher 输出控制与 GMM 前动态激活量化都在这里完成。
- python/sglang/srt/hardware_backend/npu/moe/quant.py（模块 激活量化；类别 source；类型 core-logic；符号 HiddenStatesDynamicQuant, init）：HiddenStatesDynamicQuant 新增 use_mx_quant 参数，使 FP4 等 MX dtype 也能走 npu_dynamic_mx_quant，是本功能激活量化能力的公共基础设施。
- python/sglang/srt/layers/quantization/modelslim/modelslim.py（模块 方案注册；类别 source；类型 data-contract）：get_moe_scheme 中注册 W4A4_MXFP4 到 MoE 量化方案列表，是功能被自动检测的入口。
- python/sglang/srt/layers/quantization/modelslim/schemes/__init__.py（模块 量化导出；类别 source；类型 data-contract）：导出新方案类 ModelSlimW4A4MXFP4MoE，维持 schemes 包的导入契约。
- docs/docs/hardware-platforms/ascend-npus/optimization/quantization.mdx（模块 文档；类别 other；类型 documentation）：补充 Ascend NPU 量化支持矩阵、MXFP4 MoE 启动命令与实现说明，明确硬件要求 Ascend 950+。
- docs/docs/advanced_features/quantization.mdx（模块 文档；类别 other；类型 documentation）：全局量化支持表与 ModelSlim 支持清单同步标记 W4A4_MXFP4 MoE 可用。

关键符号：ModelSlimW4A4MXFP4MoE.init, ModelSlimW4A4MXFP4MoE.create_weights, ModelSlimW4A4MXFP4MoE.process_weights_after_loading, NPUW4A4MXFP4MoEMethod.init, NPUW4A4MXFP4MoEMethod.process_weights_after_loading, NPUW4A4MXFP4MoEMethod.apply, HiddenStatesDynamicQuant.init, ModelSlimConfig.get_moe_scheme

关键源码片段

python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_w4a4_mxfp4_moe.py

新增的 ModelSlim W4A4_MXFP4 MoE 方案类，是权重创建的入口，定义 w13/w2 权重组的参数形状与 block scale 契约。

```
# python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_w4a4_mxfp4_moe.py
# ModelSlim W4A4_MXFP4 MoE scheme: 负责按 w13 / w2 分组创建 expert 权重参数
class ModelSlimW4A4MXFP4MoE(ModelSlimMoEScheme):
    """Create one ModelSlim MXFP4 expert-weight group (w13 or w2)."""

    def __init__(self, quant_config, weight_prefix):
        if weight_prefix not in ("w13", "w2"):
            raise ValueError(
                f"weight_prefix must be 'w13' or 'w2', got '{weight_prefix}'"
            )
        self.quant_config = quant_config
        self.weight_prefix = weight_prefix
        # 直接绑定 NPU 侧的量化执行方法，后续 process_weights 会转给它
        self.kernel = NPUW4A4MXFP4MoEMethod()

    def create_weights(self, layer, num_experts, hidden_size,
                       intermediate_size_per_partition, **extra_weight_attrs):
        # 标记为 block 级量化，供上层 MoE 权重处理逻辑识别
        extra_weight_attrs.update(
            {"quant_method": FusedMoeWeightScaleSupported.BLOCK.value}
        )
        if self.weight_prefix == "w13":
            output_size = 2 * intermediate_size_per_partition
            input_size = hidden_size
        else:
            output_size = hidden_size
            input_size = intermediate_size_per_partition

        # checkpoint 中 FP4 值实际存放在 float8_e4m3fn 容器（uint8 存储），
        # 因此先用 uint8 创建空参数，加载后再 repack 成 FP4 布局
        weight = torch.nn.Parameter(
            torch.empty(num_experts, output_size, input_size // 2, dtype=torch.uint8),
            requires_grad=False,
        )
        layer.register_parameter(f"{self.weight_prefix}_weight", weight)
        set_weight_attrs(weight, extra_weight_attrs)

        # UE8M0 block scale：block size 为 32，先按 K 方向 ceil 分配
        weight_scale = torch.nn.Parameter(
            torch.zeros(
                num_experts,
```

```
                output_size,
                (input_size + MXFP4_BLOCK_SIZE - 1) // MXFP4_BLOCK_SIZE,
                dtype=torch.uint8,
            ),
            requires_grad=False,
        )
        layer.register_parameter(f"{self.weight_prefix}_weight_scale", weight_scale)
        set_weight_attrs(weight_scale, extra_weight_attrs)

    def process_weights_after_loading(self, layer):
        # 布局转换与 repack 全部下沉到 NPU kernel method 里完成
        self.kernel.process_weights_after_loading(layer, self.weight_prefix)
```

python/sglang/srt/hardware_backend/npu/quantization/moe_methods.py

NPUW4A4MXFP4MoEMethod 是本功能的核心执行逻辑：权重布局转换、block scale reshape、dispatcher 输出控制与 GMM 前动态激活量化都在这里完成。

```
# python/sglang/srt/hardware_backend/npu/quantization/moe_methods.py
# NPUW4A4MXFP4MoEMethod：ModelSlim W4A4 MXFP4 MoE 在 Ascend NPU 上的核心执行方法
class NPUW4A4MXFP4MoEMethod(_NPUMoEMethodBase):
    """ModelSlim W4A4 MXFP4 MoE with single-level FP4 weights and activations."""

    def __init__(self):
        super().__init__(quant_config=None)
        self.matmul = GroupedMatmul()
        # FP4 权重 dtype 是 torch_npu 的 float4_e2m1fn_x2（int 枚举），
        # 不能直接用 torch.float4_e2m1fn_x2，需要从枚举中解析
        fp4_dtype = _get_float4_e2m1fn_x2_dtype()
        if fp4_dtype is None:
            raise RuntimeError("NPU W4A4 MXFP4 MoE requires float4 support.")
        # use_mx_quant=True 强制走 npu_dynamic_mx_quant（MX 块缩放路径）
        self.hidden_states_quantizer = HiddenStatesDynamicQuant(
            quant_dtype=fp4_dtype,
            use_mx_quant=True,
        )

    def process_weights_after_loading(self, layer, weight_prefix):
        self._validate_weight_prefix(layer, weight_prefix)

        # offline checkpoint 的 FP4 值装在 float8_e4m3fn 容器里，
        # 这里转成 NPU 的 NZ 布局并转置，供 grouped matmul 使用
        weight = getattr(layer, f"{weight_prefix}_weight")
        weight.data = npu_format_cast(weight.data).transpose(-1, -2)

        # UE8M0 block scale 从 [E, N, K // 32] reshape 成
        # [E, K // 32, N, 2]，匹配 grouped matmul 的 scale 布局
        weight_scale = getattr(layer, f"{weight_prefix}_weight_scale")
        scale = weight_scale.data.reshape(
            weight_scale.shape[0],
```

```
                weight_scale.shape[1],
                weight_scale.shape[2] // 2,
                2,
            ).transpose(1, 2)
            weight_scale.data = scale

            # 重构后的 Ascend dispatcher 目前只输出 BF16 或 INT8，
            # 所以 w13 的 routing 输出保持 BF16，激活量化放到 apply 里做
            if weight_prefix == "w13":
                self._set_dispatcher_output_dtype(layer, "bf16")

        def apply(self, quant_info, hidden_states, expert_tokens, pertoken_scale,
                  output_dtype, weight_prefix, group_list_type):
            fp4_dtype = self.hidden_states_quantizer.quant_dtype
            e8m0_dtype = _require_e8m0_dtype()

            # BF16 激活在两次 expert GMM 之前动态量化成 MXFP4
            if pertoken_scale is None:
                hidden_states, pertoken_scale = self.hidden_states_quantizer(hidden_states)
            elif pertoken_scale is not None:
                pertoken_scale = pertoken_scale.reshape(
                    hidden_states.shape[0], hidden_states.shape[1] // 32, 2
                )

            scale_args = {
                "scale": [getattr(quant_info, f"{weight_prefix}_weight_scale", None)],
                "scale_dtype": e8m0_dtype,
                "per_token_scale": [pertoken_scale],
                "per_token_scale_dtype": e8m0_dtype,
                "x_dtype": fp4_dtype,
                "weight_dtype": fp4_dtype,
            }
            scale_args.update(self._get_bias_args(quant_info, weight_prefix))
            return self.matmul.forward(
                quant_info,
                weight_prefix,
                hidden_states,
                expert_tokens.to(torch.int64),
                output_dtype,
                group_list_type=group_list_type,
                transposed=True,
                **scale_args,
            )
```

python/sglang/srt/hardware_backend/npu/moe/quant.py

HiddenStatesDynamicQuant 新增 use_mx_quant 参数，使 FP4 等 MX dtype 也能走 npu_dynamic_mx_quant，是本功能激活量化能力的公共基础设施。

```
# python/sglang/srt/hardware_backend/npu/moe/quant.py
```

```
# HiddenStatesDynamicQuant：NPU 动态量化激活的封装
class HiddenStatesDynamicQuant(BaseHiddenStatesQuant):
    """Dynamic per-token quantisation of hidden states."""

    def __init__(self, quant_dtype, use_mx_quant=False):
        super().__init__(quant_dtype)
        # 新增 use_mx_quant 开关：FP4 等 MX dtype 的 torch 对象无法直接
        # 与 float8_e4m3fn 比较，需要显式指定走 npu_dynamic_mx_quant
        if use_mx_quant or quant_dtype == torch.float8_e4m3fn:
            self._op = torch.ops.npu.npu_dynamic_mx_quant
        elif quant_dtype in (torch.int8, torch.quint4x2):
            self._op = torch.ops.npu.npu_dynamic_quant
        else:
            raise ValueError(f"Unsupported dynamic quant dtype: {quant_dtype}")

    def __call__(self, hidden_states):
        quantized, scale = self._op(hidden_states, dst_type=self.quant_dtype)
        return quantized, scale
```

评论区精华

Review 由 OrangeRedeng（NPU 维护者）主导，讨论聚焦在性能与正确性两点：

- 性能建议被拒绝：OrangeRedeng 两次建议使用 GroupedMatmulSwigluQuant（fused SwiGLU量化）以及在init_moe_routing中实现MXFP4量化（aclnnMoeInitRoutingV3）。作者 LinyuanLi0046 回复称 profiling 算子延迟数据显示两者均无明显性能收益，为保持改动精简不予采纳，并建议后续单独提交 PR。
- contiguous() 正确性争论：OrangeRedeng 建议对权重调用 .continues()（实际为 .contiguous()），作者指出对 FP4 NZ 张量调用会导致错误；reviewer 随后自答说明在 w4a8 mxfp 场景下对 continues() 张量也有过问题，同意保持现状。
- 量化逻辑收敛到公共类：OrangeRedeng 要求把激活量化的 reshape 逻辑移入 HiddenStatesDynamicQuant（与 mxfp8 的做法一致），作者以新增 use_mx_quant 参数的方式修复。
  - 是否改用 GroupedMatmulSwigluQuant 提升性能 (performance)：维持 GroupedMatmul 实现，作者建议后续如有收益场景再单独提交 PR。
  - 是否在 init_moe_routing 中实现 MXFP4 量化 (performance)：不实现，作者建议后续单独 PR 探索。
  - FP4 NZ 张量调用 .contiguous() 的正确性 (correctness)：不调用 .contiguous()，维持 npu_format_cast + transpose 的现状。
  - 激活量化逻辑是否收敛到 HiddenStatesDynamicQuant (design)：已修复，HiddenStatesDynamicQuant 增加 use_mx_quant 开关。

风险与影响

- 风险：主要风险集中在：

1. 缺少单元测试覆盖：本 PR 未新增任何 test 文件，实现正确性仅依赖端到端精度验证；FP4 布局转换、block scale reshape 这类极易出错的逻辑缺乏回归保护。
2. FP4 布局转换敏感：moe_methods.py 中 npu_format_cast + transpose(-1, -2) 以及 scale 的三段 reshape/transpose 强依赖 checkpoint 布局；review 中作者明确说明 FP4 NZ 张量上调用 contiguous() 会报错，说明该路径对内存布局非常敏感。
3. 潜在的量化开销：dispatcher 输出 BF16 后再在每次 GMM 前动态量化，相比直接 MXFP4 dispatch 多一次量化算子；PR 未附速度测试数据，无法确认性能影响。
4. 硬件范围限制：文档明确该路径要求 Ascend 950+ 产品，低端 NPU 上会直接不适用；CI 分析显示各平台失败均与本改动无关，但 NPU 专项回归仍有限。 - 影响：对用户：Qwen3 系列等 MoE 模型的 W4A4_MXFP4 离线量化 checkpoint 现在可以在 Ascend NPU 上自动检测并加载，无需显式传 --quantization 参数，扩大了 NPU 上的低比特 MoE 推理能力。对系统：复用 refactored AscendRunner 的 routing、activation 与 finalization 流程，仅新增 per-weight-group 量化方法与 GMM 前置量化，不影响现有 dense W4A4_MXFP4 线性层实现。对团队：确立了 ModelSlim MoE 量化的扩展模式（scheme 类 + kernel method + 公共量化器），后续 W4A8/W4A4 系列方案可以复用 HiddenStatesDynamicQuant 的 use_mx_quant 机制。 - 风险标记：缺少独立单元测试，FP4 布局转换敏感，GMM 前动态量化额外开销，仅支持 Ascend 950+，无速度基准数据

关联脉络

- 暂无明显关联 PR