PR #23650 完整报告

sgl-project/sglang

:sparkles: [llm][npu][quant] Add W4A8 MXFP quantization support for Qwen3 Dense on Ascend NPU

合并时间：2026-07-07 00:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23650

执行摘要

- 一句话：为 Ascend NPU Qwen3 Dense 添加 W4A8 MXFP 量化和离线方案
- 推荐动作：该 PR 是 Ascend NPU 量化路线中的重要里程碑，值得所有 NPU 用户和关注硬件后端量化的开发者精读。重点关注：
  - 在线与离线量化路径的设计区别及复用模式。
  - npu_format_cast 的扩展方式（FP4 参数传递与 ND 回退跳过逻辑）。
  - 性能测试中 .contiguous() vs strided view 的实测决策，体现了硬件相关的优化取舍。
  - 建议后续作者或团队补充 E2E 测试用例，并跟踪统一基类的重构进展。

功能与动机

作为 Issue #21584（Ascend NPU A5 MXFP8/MXFP4 量化）的一部分，旨在为 Qwen3 Dense 模型提供更激进的权重量化（4-bit）方案，以降低内存带宽压力并提升推理吞吐量。PR 依赖前置 W8A8 实现（#22352），并复用了相同的 NPU 量化基础设施。

实现拆解

1. 在线量化配置与注册：新增 python/sglang/srt/layers/quantization/npu_mxfp4.py，定义 Mxfp4W4A8Config（注册为 "mxfp_w4a8"），通过 get_quant_method 为 Linear 层分发 NPU 后端线性方法 NPUMXFP4W4A8LinearMethod，并跳过 MoE 层（fallback 到未量化）。
2. 在线核心逻辑：在 python/sglang/srt/hardware_backend/npu/quantization/linear_method_npu.py 中新增 NPUMXFP4W4A8LinearMethod。process_weights_after_loading 调用 npu_dynamic_mx_quant(dst=float4_e2m1fn_x2) 将 BF16 权重量化为 packed FP4，再通过 npu_format_cast 转 FRACTAL_NZ 格式。apply 中激活动态量化为 MXFP8 后执行 npu_quant_matmul(x2_dtype=float4_e2m1fn_x2, group_sizes=[0,0,32])。
3. 离线量化 Scheme：新增 python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_mxfp4_w4a8.py，定义 ModelSlimMXFP4W4A8Scheme。create_weights 创建 uint8 packed FP4 权重和 UE8M0 scale 参数；process_weights_after_loading 和 apply 委托给 NPUMXFP4W4A8OfflineLinearMethod（与在线方法共享推理内核，但权重来源不同）。

4. 基础设施增强：扩展 python/sglang/srt/hardware_backend/npu/utils.py 中的 npu_format_cast，新增 customize_dtype/input_dtype 参数以支持 packed FP4 的 unpack 转换，并跳过 FP4 场景的 ND fallback 检查（防止对齐错误导致结果损坏）。
5. 注册与 CLI：在 layers/quantization/__init__.py 中注册 Mxfp4W4A8Config，在 layers/quantization/modelslim/schemes/__init__.py 中导出 ModelSlimMXFP4W4A8Scheme，并在 server_args.py 的 QUANTIZATION_CHOICES 中添加 "mxfp_w4a8"。
6. 文档：更新 docs_new/docs/advanced_features/quantization.mdx 和 ascend_npu_quantization.mdx，添加 W4A8 使用说明。

关键文件：

- python/sglang/srt/hardware_backend/npu/quantization/linear_method_npu.py（模块 量化层；类别 source；类型 core-logic；符号 _get_float4_e2m1fn_x2_dtype, NPUMXFP4W4A8LinearMethod, NPUMXFP4W4A8OfflineLinearMethod, create_weights）：核心实现文件，新增 NPUMXFP4W4A8LinearMethod 和 NPUMXFP4W4A8OfflineLinearMethod，包含在线权重量化和推理全流程，是最关键的变更。
- python/sglang/srt/layers/quantization/npu_mxfp4.py（模块 量化配置；类别 source；类型 dependency-wiring；符号 Mxfp4W4A8Config, get_name, get_quant_method, from_config）：新增在线量化配置类 Mxfp4W4A8Config，负责注册和分发 NPUMXFP4W4A8LinearMethod，是用户通过 CLI 触发在线模式的入口。
- python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_mxfp4_w4a8.py（模块 量化配置；类别 source；类型 data-contract；符号 ModelSlimMXFP4W4A8Scheme, create_weights, process_weights_after_loading, apply_weights）：新增离线 W4A8 方案 ModelSlimMXFP4W4A8Scheme，是 msmodelslim 预量化权重的加载和推理入口，与在线路径共享内核但数据结构不同。
- python/sglang/srt/hardware_backend/npu/utils.py（模块 工具函数；类别 source；类型 core-logic；符号 npu_format_cast）：增强 npu_format_cast 以支持 FP4 的 unpack 参数，并跳过 ND 对齐检查，是基础设施层级的关键修改，确保 FP4 权重正确转换为 FRACTAL_NZ。
- python/sglang/srt/layers/quantization/__init__.py（模块 注册入口；类别 source；类型 dependency-wiring）：注册 Mxfp4W4A8Config 到量化框架的映射字典，使 --quantization mxfp_w4a8 能被识别。
- python/sglang/srt/layers/quantization/modelslim/schemes/__init__.py（模块 注册入口；类别 source；类型 data-contract）：导出 ModelSlimMXFP4W4A8Scheme，使离线方案在 modelslim 注册表中可见。
- python/sglang/srt/layers/quantization/modelslim/modelslim.py（模块 量化配置；类别 source；类型 data-contract）：在 _get_scheme_from_parts 中添加 W4A8_MXFP 分支，分发到 ModelSlimMXFP4W4A8Scheme，同时添加 W4A8_DYNAMIC 分支（INT4 离线路径）。
- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 configuration）：将 "mxfp_w4a8" 加入 QUANTIZATION_CHOICES，使其可作为有效 CLI 选项。

- docs_new/docs/hardware-platforms/ascend-npus/ascend_npu_quantization.mdx（模块 文档；类别 other；类型 documentation）：更新 Ascend NPU 量化文档，添加 W4A8 量化使用指南和性能数据。
- docs_new/docs/advanced_features/quantization.mdx（模块 文档；类别 other；类型 documentation）：在量化总览文档中增加 mxfp_w4a8 选项的支持说明。

关键符号：_get_float4_e2m1fn_x2_dtype, NPUMXFP4W4A8LinearMethod.create_weights, NPUMXFP4W4A8LinearMethod.process_weights_after_loading, NPUMXFP4W4A8LinearMethod.apply, NPUMXFP4W4A8OfflineLinearMethod.process_weights_after_loading, NPUMXFP4W4A8OfflineLinearMethod.apply, Mxfp4W4A8Config.get_quant_method, ModelSlimMXFP4W4A8Scheme.create_weights, npu_format_cast (enhanced)

关键源码片段

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

核心实现文件，新增 NPUMXFP4W4A8LinearMethod 和 NPUMXFP4W4A8OfflineLinearMethod，包含在线权重量化和推理全流程，是最关键的变更。

```
# python/sglang/srt/hardware_backend/npu/quantization/linear_method_npu.py
# 新增的在线 W4A8 线性方法（NPUMXFP4W4A8LinearMethod）
# 在 process_weights_after_loading 中将 BF16 权重在线量化为 packed FP4，
# 在 apply 中动态量化激活为 MXFP8 并调用 FP4 matmul。

def _get_float4_e2m1fn_x2_dtype():
    # FP4 dtype 必须从 torch_npu 取（int enum，如 296），
    # torch.float4_e2m1fn_x2 对象在 op-plugin 中会导致 "output y must be
    # same shape as input x" 错误。这里惰性加载并 fallback，
    # 确保模块在非 NPU 环境可导入。
    from sglang.srt.utils import is_npu
    if is_npu():
        import torch_npu
        npu_dtype = getattr(torch_npu, "float4_e2m1fn_x2", None)
        if npu_dtype is not None:
            return npu_dtype
    return getattr(torch, "float4_e2m1fn_x2", None)


class NPUMXFP4W4A8LinearMethod(_NPULinearMethodBase):
    """Ascend NPU W4A8 在线量化：MXFP4 权重 + MXFP8 激活。"""

    def create_weights(self, layer, ...):
        # 创建 BF16 权重缓冲区（完整精度），供后续在线量化。
        # 与离线方案不同，这里不是预量化 buffer。
        weight = Parameter(torch.empty(output_size, input_size, dtype=params_dtype),
                           requires_grad=False)
        layer.register_parameter("weight", weight)
```

```
    def process_weights_after_loading(self, layer):
        # 在线量化权重：BF16 → packed FP4 + UE8M0 scale
        weight = layer.weight.data
        # 调用 NPU 原生双级量化 API（MXFP4：FP4 + L0 FP32 scale + L1 FP8_E8M0 scale）
        quant_weight, l0_scale, l1_scale = \
            torch.ops.npu.npu_dynamic_mx_quant(weight, dst_type=_get_float4_e2m1fn_x2_dtype())
        # 转 FRACTAL_NZ 格式（NPU 高效计算所需）
        quant_weight = npu_format_cast(quant_weight.view(torch.int8), acl_format=29,
                                       customize_dtype=torch.float8_e4m3fn,
                                       input_dtype=_get_float4_e2m1fn_x2_dtype())
        # 转置并存储
        layer.weight = Parameter(quant_weight.transpose(0, 1).contiguous(), requires_grad=False)
        # 存储双级 scale（处理转置和 contiguous）
        layer.weight_scale_0 = Parameter(l0_scale.squeeze().transpose(0, 1).contiguous(),
                                         requires_grad=False)
        layer.weight_scale_1 = Parameter(...) # 类似处理

    def apply(self, layer, x, bias):
        original_dtype = x.dtype
        # 动态量化激活为 MXFP8（FP8 + UE8M0 scale）
        qx, qscale = torch.ops.npu.npu_dynamic_mx_quant(x, dst_type=torch.float8_e4m3fn)
        # 执行 W4A8 matmul（实际是 W4A4 compute 但 A8 指 scale 精度）
        out = torch.ops.npu.npu_quant_matmul(
            qx, layer.weight, layer.weight_scale_0, layer.weight_scale_1,
            pertoken_scale=qscale,
            bias=bias,
            output_dtype=original_dtype,
            x2_dtype=_get_float4_e2m1fn_x2_dtype(),
            group_sizes=[0, 0, 32] # 块大小 32
        )
        return out


class NPUMXFP4W4A8OfflineLinearMethod(_NPULinearMethodBase):
    """离线 W4A8 核：加载预量化 packed FP4 权重，推理时激活动态 MXFP8。"""

    def process_weights_after_loading(self, layer):
        # 预量化权重已是 packed FP4（uint8, shape [out, in//2]），
        # 转换为 FRACTAL_NZ 并加载双级 scale。注意不做 .contiguous(),
        # 因为预量化数据的块尺度映射依赖于非连续视图。
        w = layer.weight.data # uint8
        # 转 NZ 格式（通过 npu_format_cast 带 unpack 参数）
        w_nz = npu_format_cast(w, customize_dtype=torch.float8_e4m3fn,
                               input_dtype=_get_float4_e2m1fn_x2_dtype())
        # 转置但不 contiguous（关键优化：stride view 在 NPU matmul 中更快）
        layer.weight.data = w_nz.transpose(0, 1).data
        # 类似的 scale 转换，均使用 .data 赋值保持非连续
        ...
```

```
def apply(self, layer, x, bias):
    # 与在线方法的 apply 完全一致：激活动态 MXFP8，matmul 调用相同 API。
    ...
```

评论区精华

1. 复用 npu_format_cast 而非直接调用 torch_npu 原语：ping1jing2 建议重用现有工具函数，作者扩展了 npu_format_cast 以支持 FP4 unpack 参数，并在所有 W4A8 路径中统一使用，避免了代码散布和手动格式转换。
2. 在线配置是否需要独立文件：OrangeRedeng 询问是否可将在线逻辑合并到 modelslim.py 中以减少重复。作者解释在线（--quantization mxfp_w4a8）和离线（modelslim）是两套注册机制，各自独立，不宜合并。reviewer 提出后续可用统一基类（NPUUnquantLinearMethod）减少代码重复。
3. .contiguous() 的使用策略：AI 审查建议在离线路径预转置权重并调用 .contiguous()，但作者通过实测发现 strided view 在 NPU 上反而更慢，因此仅在线路径使用 .contiguous()（安全），离线路径保持非连续视图以保持预量化数据的块尺度映射正确。这一讨论体现了性能测量驱动决策的原则。
4. FP4 dtype 的来源：作者识别出 torch.float4_e2m1fn_x2 与 torch_npu.float4_e2m1fn_x2 在不同版本下的兼容性问题，采用惰性解析 + fallback 机制（_get_float4_e2m1fn_x2_dtype），避免对 torch_npu 版本的硬依赖。
5. 缺少测试文件：虽未构成明确讨论，但发现 PR 未包含对应的端到端测试注释或回归测试文件，评审中未质疑但可能造成潜在覆盖风险。

- 复用 npu_format_cast 而非直接调用 torch_npu 原语 (design): 采用统一入口 npu_format_cast 并新增 FP4 参数，所有 W4A8 路径均通过它转换权重格式。
- 是否需要单独在线配置文件 vs 合并到 modelslim.py (design): 保持独立文件，因为这是在线 vs 离线两种不同的注册 / 触发机制；后续重构可考虑统一基类减少重复。
- .contiguous() 的使用策略（在线 vs 离线） (performance): 保留两套做法：在线 .contiguous() 安全，离线不 .contiguous() 以匹配预量化数据布局。这纠正了“预转置 + contiguous 总是更快”的通用假设。
- FP4 dtype 的惰性解析与版本兼容 (correctness): 实现惰性解析 + 双来源 fallback，确保 FP4 dtype 始终正确且模块可跨后端导入。
- 缺少 E2E 测试文件 (testing): 未在本次 PR 中添加测试，建议后续补充 NPU-specific E2E 回归测试。

风险与影响

- 风险：
  1. torch_npu 版本依赖：FP4 的 NPU op（npu_dynamic_mx_quant(dst=float4_e2m1fn_x2)) 在较旧的 torch_npu 构建上可能失败。代码已通过惰性 dtype 解析 + torch 退避缓解，但仍需 Ascend 950/A5 及特定 CANN 版本。
  2. 缺少测试文件：PR 未包含相应的单元测试或 E2E 回归测试。量化路径的正确性仅依赖作者的临时验证，后续重构或版本升级可能引入回归，且 CI 无法自动拦截。

3. 离线权重格式耦合：ModelSlimMXFP4W4A8Scheme 当前假设 msmodelslim 导出的是 packed FP4（uint8, shape [out, in//2]），但 PR body 提到旧版本可能导出 float8_e4m3fn（与 MXFP8 相同布局），若 msmodelslim 导出格式变更，该 scheme 将失效。目前文档已标注这一限制。
4. 代码复杂度增加：新增两个 LinearMethod 子类 + 一个 Config + 基础设施改动，增加了 NPU 后端的维护成本。后续若推统一基类可部分缓解。 - 影响：用户影响：使用 Ascend NPU 运行 Qwen3/3.5 Dense 模型的用户可通过 --quantization mxfp_w4a8（在线）或 --quantization modelslim 配合预量化权重获得 11-13% 吞吐提升及约 10% 时延降低，TTFT 改善尤为显著（-32%）。在线模式无需预量化权重，上手更简便。系统影响：仅影响启用指定量化选项的模型加载和推理路径，未启用时完全无性能损耗。新增文件独立且不影响其他后端（CUDA/ROCm/CPU）。团队影响：需要维护新的量化组件和对应的文档；后续需在重构中统一 LinearMethod 基类以降低维护成本。

- 风险标记：缺少测试覆盖，硬件版本依赖（torch_npu），离线权重格式耦合，路径分歧增加维护成本

关联脉络

- PR #22352 [NPU] Add W8A8 MXFP8 quantization support for Qwen3 Dense on Ascend NPU: 本 PR 的前置依赖，本 PR 复用了 #22352 中建立的 NPU 量化基础设施（_NPULinearMethodBase、ModelSlimConfig dispatch 等），并在其基础上扩展 W4A8。
- PR #21584 [RFC][NPU] Ascend NPU A5 Support for MXFP8/MXFP4 Quantization: 关联的 Issue（已转化为关联 PR 引用），本 PR 是其中 LLM 阶段的一部分，实现了 W4A8 MXFP 量化，对应路线图中的 Qwen3 Dense W4A8。
- PR #25663 [refactor] Unify NPU linear method with base class and parameter extraction: 在评论中 OrangeRedeng 提及此 PR 中的重构思路（统一基类），为本 PR 的后续演进提供了方向。