PR #18005 完整报告

sgl-project/sglang

[AMD][MXFP4] Online MXFP4 quantization 1/N - dense and MOE models w. original BF16 weight

合并时间：2026-06-04 03:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/18005

执行摘要

- 一句话：AMD MI350X 在线 MXFP4 量化，支持 Dense 和 MOE
- 推荐动作：值得精读。该 PR 展示了如何通过包装 weight_loader 实现在线量化，从而避免预量化模型的加载开销。设计模式（使用 is_checkpoint_mxfp4_serialized 控制流统一代码路径）值得借鉴。讨论中关于准确性降级和 weight loader 优化的权衡也很有教育意义。

功能与动机

允许从 BF16 模型直接在线量化成 MXFP4，无需预量化 checkpoint，节省加载内存和磁盘空间。PR 描述中说明 'This PR implements online MXFP4 quantization (targeting Instinct MI350X/MI355X). For both dense and MOE layers, online MXFP4 quantization is run in weight loaders by wrapping the original weight_loader with the online quantization step, before eventually calling it.' 该实现与现有 PR #9049 类似，但避免了 #9049 将所有高精度权重加载到设备上的高内存开销。

实现拆解

1. 配置生成与入口：在 QuarkConfig（quark.py）中新增 online_scheme 参数，当指定 --quantization quark_mxfp4 时，调用 _create_online_mxfp4_config 动态生成量化配置（权重规格、输入规格、排除层等）。配置中自动排除 lm_head、router、gate 等层，并禁止 packed_modules_mapping 的 value 键（由 per-group 量化特性决定）。
2. 密集层在线量化：在 quark_w4a4_mxfp4.py 中，create_weights 识别 is_checkpoint_mxfp4_serialized=False，将原始 weight_loader 替换为 get_online_mxfp4_weight_loader 返回的新闭包。该闭包在每个 shard 加载时：将权重转移到 device → 调用 dynamic_mxfp4_quant 进行 per-group 量化 → 将量化后权重（uint8）填充到预设的 PackedvLLMParameter。权重 scale 则使用原始 weight_loader 加载（从量化结果直接写入）。
3. MOE 层在线量化：在 quark_w4a4_mxfp4_moe.py 中实现类似逻辑。get_online_weight_loader 返回 online_mxfp4_moe_weight_loader，该闭包处理 w13_weight 和 w2_weight 的量化。注意 MOE 的权重 scale 存储模式为 uint8 per-block（OCP_MX_BLOCK_SIZE=32），量化后直接调用 original_weight_loader 将 scale 写入预注册的 scale 参数。
4. 辅助日志与统计：

- loader.py 中 load_weights_and_postprocess 在调用 model.load_weights 前后记录峰值内存和显存增量，便于测试验证（仅在 CUDA-like 设备生效）。
- model_runner.py 在模型加载后输出在线量化层统计（通过 quantized_layers 属性），告知用户哪些层已被量化。

5. 测试配套：新增 test/registered/quant/test_quark_mxfp4.py，包含 TestOnlineQuantizationMemoryLoad 和 TestOnlineQuantizationMemoryLoadDense 类，启动服务器并解析日志中的内存峰值，检查在线量化是否减少内存占用；同时运行 GSM8K 评估验证精度。

关键文件：

- python/sglang/srt/layers/quantization/quark/schemes/quark_w4a4_mxfp4_moe.py（模块 量化层；类别 source；类型 core-logic；符号 init, process_weights_after_loading, get_online_weight_loader, online_mxfp4_moe_weight_loader）：核心 MOE 在线量化实现，引入了 get_online_weight_loader 和闭包 online_mxfp4_moe_weight_loader。该文件变化最大，涉及初始化、权重创建和加载流程重构。
- test/registered/quant/test_quark_mxfp4.py（模块 测试；类别 test；类型 test-coverage；符号 TestOnlineQuantizationMemoryLoad, setUpClass, _extract_peak_memory_before_load, _extract_memory_increase_load_weights）：新增测试，覆盖在线量化的内存节省和 GSM8K 准确率，确保功能正确性。
- python/sglang/srt/layers/quantization/quark/quark.py（模块 配置；类别 source；类型 core-logic；符号 init, quantized_layers, _create_online_mxfp4_config）：核心配置类改造，支持在线方案配置生成、层排除、量化统计。
- python/sglang/srt/layers/quantization/quark/schemes/quark_w4a4_mxfp4.py（模块 量化层；类别 source；类型 core-logic；符号 get_online_mxfp4_weight_loader, online_mxfp4_weight_loader）：密集层在线量化 weight loader 实现，包含 get_online_mxfp4_weight_loader 闭包。
- python/sglang/srt/model_loader/loader.py（模块 模型加载；类别 source；类型 data-contract）：在 load_weights_and_postprocess 函数中增加 GPU 内存日志，用于测试验证。
- python/sglang/srt/model_executor/model_runner.py（模块 模型运行；类别 source；类型 data-contract）：加载完成后输出在线量化层统计信息，提升可观测性。
- python/sglang/srt/server_args.py（模块 参数配置；类别 source；类型 configuration）：在量化选择列表中增加 quark_mxfp4 并添加 TODO 注释建议清理。
- python/sglang/srt/configs/model_config.py（模块 模型配置；类别 source；类型 data-contract）：修复模型配置传递，支持 hf_config 传递给 QuarkConfig。
- python/sglang/srt/constants.py（模块 常量；类别 source；类型 core-logic）：新增 GIB_BYTES 常量，消除魔法数字。
- python/sglang/srt/model_loader/weight_utils.py（模块 权重工具；类别 source；类型 data-contract）：增加 mxfp_supported 导入检查，支持在线量化硬件判断。
- python/sglang/srt/layers/quantization/__init__.py（模块 量化入口；类别 source；类型 dependency-wiring）：导出 QuarkConfig 等类，但变更微小。

关键符号：QuarkW4A4MXFP4.init, QuarkW4A4MXFP4.process_weights_after_loading, QuarkW4A4MXFP4.create_weights, QuarkW4A4MXFP4.get_online_mxfp4_weight_loader, QuarkW4A4MXFp4MoE.init, QuarkW4A4MXFp4MoE.process_weights_after_loading, QuarkW4A4MXFp4MoE.create_weights, QuarkW4A4MXFp4MoE.get_online_weight_loader, QuarkConfig.init, QuarkConfig.get_quant_method, QuarkConfig._create_online_mxfp4_config, load_weights_and_postprocess, ModelRunner.load_model

关键源码片段

python/sglang/srt/layers/quantization/quark/schemes/quark_w4a4_mxfp4_moe.py

核心 MOE 在线量化实现，引入了 get_online_weight_loader 和闭包 online_mxfp4_moe_weight_loader。该文件变化最大，涉及初始化、权重创建和加载流程重构。

```
# SPDX-License-Identifier: Apache-2.0
from __future__ import annotations
import logging
from typing import TYPE_CHECKING, Any
import torch
from sglang.srt.layers.quantization.quark.schemes import QuarkMoEScheme
from sglang.srt.utils.common import mxfp_supported

if TYPE_CHECKING:
    from sglang.srt.layers.moe.token_dispatcher import CombineInput, StandardDispatchOutput

logger = logging.getLogger(__name__)

# 条件导入 aiter 的量化函数，仅在 ROCm 上可用
_is_hip = is_hip()
if _is_hip:
    from aiter.ops.triton.quant import dynamic_mxfp4_quant
else:
    dynamic_mxfp4_quant = None

class QuarkW4A4MXFp4MoE(QuarkMoEScheme):
    def __init__(self, weight_config, input_config, is_checkpoint_mxfp4_serialized: bool = True):
        # ... 原有初始化 ...
        self.is_checkpoint_mxfp4_serialized = is_checkpoint_mxfp4_serialized
        # 非预量化模式下检查硬件支持并发出警告
        if not self.is_checkpoint_mxfp4_serialized:
            if not mxfp_supported():
                raise NotImplementedError("Online MXFP4 quantization for MoE layers requires
                AMD ROCm gfx95x")
            logger.info_once("Using online MXFP4 quantization... Beware of prediction quality
            degradation.")

    def create_weights(self, layer, num_experts, hidden_size, intermediate_size_per_partition,
```

```
        params_dtype, **extra_weight_attrs):
            original_weight_loader = extra_weight_attrs.get("weight_loader")
            if self.is_checkpoint_mxfp4_serialized:
                weight_loader = original_weight_loader
            else:
                # 替换为在线量化 weight loader
                weight_loader = self.get_online_weight_loader(layer, original_weight_loader)
            extra_weight_attrs["weight_loader"] = weight_loader
            # 注册参数 ... 后续代码保持

        def get_online_weight_loader(self, layer, original_weight_loader):
            """返回一个闭包，在权重加载时执行逐 grup 的 MXFP4 量化"""
            def online_mxfp4_moe_weight_loader(param, loaded_weight, shard_id=None):
                # 将加载的权重移至 GPU
                loaded_weight = loaded_weight.to(param.device)
                # 使用 aiter 的 dynamic_mxfp4_quant 进行量化，返回 (qweight, weight_scale)
                qweight, weight_scale = dynamic_mxfp4_quant(loaded_weight)
                # 直接调用原始 weight_loader 将量化数据写入 param
                original_weight_loader(param, qweight, shard_id)
                # 对于 MOE 的 scale 参数，需要根据 param 名称分别写入
                if "w13" in param_name:
                    original_weight_loader(layer.w13_weight_scale, weight_scale, shard_id)
                elif "w2" in param_name:
                    original_weight_loader(layer.w2_weight_scale, weight_scale, shard_id)
            return online_mxfp4_moe_weight_loader
```

python/sglang/srt/layers/quantization/quark/quark.py

核心配置类改造，支持在线方案配置生成、层排除、量化统计。

```
# SPDX-License-Identifier: Apache-2.0
import logging
from typing import Any, Optional
from sglang.srt.layers.quantization.base_config import QuantizationConfig
from transformers import PretrainedConfig

class QuarkConfig(QuantizationConfig):
    def __init__(self, quant_config=None, hf_config: PretrainedConfig | None = None,
                 kv_cache_group=None, kv_cache_config=None, pack_method="reorder",
                 is_prequantized: bool = False, online_scheme: Optional[str] = None):
        super().__init__()
        # 如果指定了在线方案，则动态生成 quant_config
        if online_scheme is not None:
            assert not is_prequantized
            if online_scheme == "quark_mxfp4":
                quant_config = self._create_online_mxfp4_config(model_type=hf_config.model_type)
            else:
                raise ValueError(f"Unsupported online_scheme: {online_scheme}")
        self.is_prequantized = is_prequantized
        self.quant_config = quant_config
```

```
    # ... 其他初始化
    self._quantized_layers = set()

@property
def quantized_layers(self) -> tuple[list[str], int]:
    """返回去重后的层类型列表及其总数，用于日志输出"""
    layer_types = sorted(set(name.split(".")[-1] for name in self._quantized_layers))
    return layer_types, len(self._quantized_layers)

def get_quant_method(self, layer, prefix):
    # ... 原有逻辑，但在每个量化分支中增加 self._quantized_layers.add(prefix)

@staticmethod
def _create_online_mxfp4_config(model_type: str) -> dict:
    """为在线 MXFP4 构建合成量化配置"""
    # 根据模型类型动态设置 layer_quant_config
    quant_config = {
        "quantize": "quark_mxfp4",
        "format": "mx_fp4",
        "layer_quant_config": [
            {"name": "*", "weight": {"num_bits": 4, "block_size": 32, "target_dtype": "float4_e2m1"}
            },
            # 排除常见不需要量化的层
        ],
        "exclude": ["lm_head", "router", "gate"],
        "packed_modules_mapping": {},
    }
    return quant_config
```

评论区精华

1. 准确性下降：kkHuang-amd 指出 GSM8K 准确率从 0.934 下降到 0.892。fxmarty-amd 确认这是对所有线性层应用 MXFP4 量化的结果，并说明更大模型（如 Qwen3.5-397B）影响较小（下降 0.5% 以内）。建议用户验证自身模型。
2. MOE weight loader 复杂性：kkHuang-amd 和 HaiShaw 质疑为何 MOE 需要额外 buffer 创建；fxmarty-amd 最初解释为需要按组量化，但后续在 per-group 量化下简化，移除了不必要的完整权重暂存。HaiShaw 要求简化状态管理。
3. QUANTIZATION_CHOICES 清理：fxmarty-amd 指出列表中有大量不支持在线量化的选项，提议只保留真正支持在线量化的。BowenBao 担忧已预量化模型指定 --quantization 的兼容性。最终保留现有列表但添加 TODO 注释。
4. quantized_layers 属性用途：kkHuang-amd 询问 self._quantized_layers 的作用；fxmarty-amd 解释用于在 model_runner.py 中输出日志，方便用户了解哪些层被量化。
5. 常量提取：BowenBao 指出魔法数字 1073741824，fxmarty-amd 提取为 GIB_BYTES 常量。

- 准确率下降与量化范围 (correctness): 准确率下降是 MXFP4 量化的固有特性，已在文档中说明，建议用户验证自身模型。后续 PR #18182 可能包含更精细的配置来改善。

- MOE weight loader 复杂性简化 (design): MOE 在线量化已被简化为按 shard 量化并直接写入 scale，无需完整权重暂存，状态管理更清晰。
- QUANTIZATION_CHOICES 在线量化支持梳理 (design): 保留原有列表并添加 TODO 注释，避免破坏现有工作流。后续可独立清理。
- quantized_layers 属性的用途 (design): 保留该属性，认为有调试价值。
- 魔法数字提取为常量 (style): 已修复。

风险与影响

- 风险:
  1. 准确率下降：对所有线性层进行 MXFP4 量化可能导致准确率下降 1-4 个百分点（在中等规模模型上尤其明显）。必须确保用户通过文档知晓此风险。
  2. 硬件依赖：在线量化依赖 AMD ROCm 设备和 FP4 硬件支持（gfx95x），在不兼容硬件上会抛出 NotImplementedError。
  3. 内存峰值：虽然在线量化相比预量化 checkpoint 节省了总内存，但在量化过程中每个 shard 首先加载到 device 再量化，短期内存在一次物理转换的峰值。测试已验证内存增量在合理范围。
  4. 未来兼容性：_create_online_mxfp4_config 中对 packed_modules_mapping value 的移除假设与 per-group 量化不冲突，若后续其他量化方法复用此路径，需重新评估。 - 影响：用户：AMD MI350X/MI355X 用户现在可以无需提前量化模型即可运行 MXFP4 推理，简化部署流程。但需注意准确率权衡。系统：增加了一种新的量化方法入口（quark_mxfp4），与现有预量化模型加载路径并行。内存日志和量化统计为性能调优提供帮助。团队：需要维护新的 weight_loader 分支，并与后续的 quark_int4fp8_moe 等方法同步。文档介绍了新的配置参数。
- 风险标记：精度下降，硬件依赖，内存峰值，兼容性风险

关联脉络

- 暂无明显关联 PR