PR #49347 完整报告

vllm-project/vllm

[Online quantization] Add online MXFP4 quantization support

合并时间：2026-08-08 03:39

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49347

执行摘要

- 一句话：新增在线 MXFP4 量化，支持未量化模型直接部署
- 推荐动作：值得精读。重点看三处可复用设计：一是 mxfp4_utils.py 的 Triton 内核如何按 OCP MX 规范实现 round-to-nearest-even 并与硬件 /AITER 逐位对齐（含 denormal 的 sticky bit 处理）；二是 online/mxfp4.py 的 MoE 加载链路（padding 清零 → 批量量化 → 后端格式转换 → kernel 构建）的职责划分；三是 test_online_mxfp4.py 的 Quark parity 与 TP 分片一致性测试思路（fix_negative_zeros 归一化负零差异的细节值得借鉴）。同时建议关注 review 中暴露的 dynamic/static 键语义缺陷与 CUDA 支持边界，这两点会影响后续在线量化配置解析的演进方向。

功能与动机

PR body 明确目标："This PR adds online MXFP4 linear/MOE quantization support. This lets users pass --quantization mxfp4 (or --quantization online --quantization-config '{"linear": "mxfp4", "moe": "mxfp4"}') on an unquantized (bf16/fp16) model directly." 此前 MXFP4 只能通过加载 Quark 等离线量化后的 checkpoint 使用，用户需要先完成离线转换与模型发布；本 PR 让 bf16/fp16 模型在权重加载阶段即时量化，直接获得 MXFP4 的显存 / 带宽收益。同时补齐 Triton fallback 路径（downcast_to_mxfp in mxfp4_utils.py），并把 XPU/AITER/Triton 的量化实现收敛到单一 mxfp4_quantize 入口；测试设计上以 " 在线量化结果与 AMD Quark checkpoint 权重逐字节一致 " 作为正确性锚点。

实现拆解

1. 新增在线量化方法类（核心入口，vllm/model_executor/layers/quantization/online/mxfp4.py，+317 行）：新增 Mxfp4OnlineLinearMethod（继承 _Fp8OnlineLinearBase）与 Mxfp4OnlineMoEMethod（继承 OnlineMoEMethodBase）。Linear 方法在 create_weights 中校验 input_size_per_partition 必须能被 32 整除，在 process_weights_after_loading 中调用 mxfp4_quantize 把 bf16/fp16 权重转成 packed uint8（两个 e2m1 值一个字节）+ e8m0 scale，并通过 init_mxfp4_linear_kernel 选择内核、apply 委托 kernel.apply_weights。MoE 方法通过 select_mxfp4_moe_backend 选择 AITER/emulation 等后端，maybe_roundup_sizes 复用 oracle 的 round-up 逻辑，process_weights_after_loading 先 _zero_padding 再批量量化 w13/w2，最后经 _setup_kernel 按后端格式转换并构建 MoE kernel。

2. Triton 动态量化内核与统一入口（vllm/model_executor/layers/quantization/utils/mxfp4_utils.py，+317 行）：新增 @triton.jit 内核 _compute_quant_and_scale 与 _downcast_to_mxfp，按 OCP MX 规范实现 e8m0 scale 的 round-to-nearest-even 计算与 e2m1 打包（与硬件 /AITER 行为逐位对齐），padding 区排除在 scale 计算之外并置 0；Python 包装 downcast_to_mxfp 处理任意轴量化、permute 与 shape 校验。新增 mxfp4_quantize 聚合 XPU/AITER/Triton 多条实现，作为所有调用方的统一量化入口。
3. 配置与分发接线（vllm/model_executor/layers/quantization/online/base.py、vllm/config/quantization.py）：online/base.py 的 _ONLINE_LINEAR_METHODS/_ONLINE_MOE_METHODS 分发表注册 kMxfp4Static → 新增方法类；config/quantization.py 增加简写 "mxfp4" → kMxfp4Dynamic（与已有 mxfp8 的 kMxfp8Dynamic 语义保持一致），激活量化由 online/base.py 的默认回退逻辑（L138-145）补齐。review 中曾尝试引入 mxfp4_static/mxfp4_dynamic 双简写，最终回退保持单简写并文档化。
4. MoE 后端接入与 padding 语义统一（vllm/model_executor/layers/fused_moe/oracle/mxfp4.py、ocp_mx_emulation_moe.py、quantization/online/moe_base.py、online/fp8.py）：oracle/mxfp4.py 为 AITER_MXFP4_MXFP4/AITER_MXFP4_FP8 后端增加 convert_gpt_oss_weight_to_mxfp4_moe_kernel_format 转换分支；ocp_mx_emulation_moe.py 新增 is_supported_config 静态方法，无 amd-quark 时拒绝该后端；_zero_padding 从 online/fp8.py 上移到 moe_base.py 基类，供所有在线 MoE 方法复用在量化前清零 roundup padding（在线路径用 empty_strided 分配权重、loader 只写 unpadded 切片，未初始化 NaN 会污染 e8m0 scale 计算，而 Quark 路径用 zeros 分配）。
5. 测试、评测与文档配套：新增 tests/quantization/test_online_mxfp4.py（606 行），覆盖 triton/aiter/xpu/quark 四路实现与 torch 参考的逐位对比（含 fix_negative_zeros 归一化 e2m1 负零差异）、TP 2/4/8 分片量化与全局量化的一致性、在线 MoE 与 Quark checkpoint 权重逐字节一致（含 padding 场景，测试专门用 NaN 填充复现未初始化状态）；tests/quantization/test_online.py 把 mxfp4 加入参数化并限定 gfx942/gfx950 平台运行；tests/kernels/quantization/test_mxfp4_kernel_selection.py 用 monkeypatch 绕过 has_quark 门控验证 emulation 内核；新增 Qwen3-30B-A3B-MXFP4-AITER-TP2(-online).yaml 等 GSM8K 配置与 docs/features/quantization/online.md 文档。

关键文件：

- vllm/model_executor/layers/quantization/online/mxfp4.py（模块 在线量化；类别 source；类型 core-logic；符号 _quantize_mxfp4_moe_weight, Mxfp4OnlineLinearMethod, Mxfp4OnlineMoEMethod, process_weights_after_loading）：本 PR 核心新增文件：Mxfp4OnlineLinearMethod 与 Mxfp4OnlineMoEMethod 两个在线量化方法，在权重加载阶段把 bf16/fp16 权重转为 packed FP4 + e8m0 scale，并按后端（AITER/emulation 等）构建 kernel；MoE 路径的 _zero_padding、批量量化、_setup_kernel 链路都在此定义。
- vllm/model_executor/layers/quantization/utils/mxfp4_utils.py（模块 量化工具；类别 source；类型 core-logic；符号 downcast_to_mxfp, mxfp4_quantize, _compute_quant_and_scale, _downcast_to_mxfp）：提供按 OCP MX 规范实现、与硬件 /AITER 逐位对齐的 Triton 动态量化内核（downcast_to_mxfp/_compute_quant_and_scale）与统一入口 mxfp4_quantize，所有后端（XPU/AITER/Triton）共用的量化原语，是本

特性的正确性基石。

- tests/quantization/test_online_mxfp4.py（模块 量化测试；类别 test；类型 test-coverage；符号 fix_negative_zeros, assert_quantized_weights_equal, test_mxfp4_quantization_correctness, test_online_mxfp4_tp_weight_quant_matches_unsharded）：606 行新增测试是本特性可信度的主要支撑：覆盖 triton/aiter/xpu/quark 四路量化实现与 torch 参考的逐位对比、TP 2/4/8 分片量化一致性、在线 MoE 与 Quark checkpoint 权重逐字节对齐（含 padding 场景），fix_negative_zeros 归一化 e2m1 负零差异的细节也在此体现。
- vllm/model_executor/layers/fused_moe/oracle/mxfp4.py（模块 量化内核；类别 source；类型 core-logic）：MoE 在线量化落到 AITER 内核的关键接线：convert_weight_to_mxfp4_moe_kernel_format 增加 AITER_MXFP4_MXFP4/AITER_MXFP4_FP8 的 GPT-OSS 权重格式转换分支，此前仅支持 Quark 离线路径。
- vllm/model_executor/layers/quantization/online/base.py（模块 在线量化；类别 source；类型 data-contract）：在线量化分发表注册 kMxfp4Static → 新方法类，是配置键到具体实现的接线点；同时暴露了简写键 kMxfp4Dynamic 与分发键 kMxfp4Static 不一致这一 review 核心争议。
- vllm/model_executor/layers/quantization/online/moe_base.py（模块 在线量化；类别 source；类型 core-logic；符号 _zero_padding）：_zero_padding 从 fp8.py 上移为基类方法，统一所有在线 MoE 方法量化前的 padding 清零语义，修复在线路径 empty_strided 分配与 Quark zeros 分配的差异导致的量化结果不一致。
- vllm/model_executor/layers/fused_moe/experts/ocp_mx_emulation_moe.py（模块 仿真内核；类别 source；类型 data-contract；符号 is_supported_config）：新增 is_supported_config 静态门控（依赖 amd-quark），为在线 MoE 的 emulation 后端提供显式的可支持性判定，避免缺依赖时静默误选。
- vllm/model_executor/layers/quantization/online/fp8.py（模块 在线量化；类别 source；类型 refactor；符号 _zero_padding）：原 _zero_padding 定义从此文件移除（上移至 moe_base.py 基类），在线 FP8 MoE 路径改为复用基类实现，属于本 PR 的配套重构。
- vllm/config/quantization.py（模块 配置层；类别 config；类型 configuration）：新增 "mxfp4" 简写映射到 kMxfp4Dynamic（与 mxfp8 一致），是用户入口；其值与分发表 kMxfp4Static 的错位在 review 中成为核心讨论点，并影响后续嵌套 weight 配置的解析。
- tests/quantization/test_online.py（模块 量化测试；类别 test；类型 test-coverage；符号 on_gfx950, on_gfx942）：把 mxfp4 加入在线量化参数化测试并限定 gfx942/gfx950 平台（CUDA 上 skip），同时引入 rocm_aiter_ops.refresh_env_variables 的环境刷新，直接影响 CI 中对新特性的覆盖口径。

关键符号：Mxfp4OnlineLinearMethod, Mxfp4OnlineMoEMethod, _quantize_mxfp4_moe_weight, process_weights_after_loading, _setup_kernel, get_fused_moe_quant_config, maybe_roundup_sizes, downcast_to_mxfp, mxfp4_quantize, _compute_quant_and_scale, _downcast_to_mxfp, _zero_padding, is_supported_config, has_quark

关键源码片段

vllm/model_executor/layers/quantization/online/mxfp4.py

本 PR 核心新增文件：Mxfp4OnlineLinearMethod 与 Mxfp4OnlineMoEMethod 两个在线量化方法，在权重加载阶段把 bf16/fp16 权重转为 packed FP4 + e8m0 scale，并按后端（AITER/emulation 等）构建 kernel；MoE 路径的 _zero_padding、批量量化、_setup_kernel 链路都在此定义。

```
def _quantize_mxfp4_moe_weight(
    weight: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor]:
    """批量量化：逐 expert 调用 mxfp4_quantize 后预分配输出张量。

    返回 packed FP4 权重（uint8，一个字节装两个 e2m1 值）与每组 32 元素
    共享的 e8m0 scale（uint8），每个 expert 一对。
    """
    num_experts = weight.size(0)
    first_quant, first_scale = mxfp4_quantize(weight[0])
    # 预分配而非 stack：保证内存布局一致，避免后续 kernel 格式转换出错
    w_quant = torch.empty(
        (num_experts, *first_quant.shape),
        dtype=first_quant.dtype,
        device=weight.device,
    )
    w_scales = torch.empty(
        (num_experts, *first_scale.shape),
        dtype=first_scale.dtype,
        device=weight.device,
    )
    w_quant[0] = first_quant
    w_scales[0] = first_scale
    for i in range(1, num_experts):
        w_quant[i], w_scales[i] = mxfp4_quantize(weight[i])

    return w_quant, w_scales


class Mxfp4OnlineMoEMethod(OnlineMoEMethodBase):
    """MoE 方法：加载时把 bf16/fp16 专家权重在线量化为 MXFP4。"""

    def process_weights_after_loading(self, layer: Module) -> None:
        # 防止重复处理：在线量化会被多层复用同一个 layer 对象
        if getattr(layer, "_already_called_process_weights_after_loading", False):
            return

        # roundup 出来的 padding 区由基类统一清零：在线路径用
        # empty_strided 分配权重，loader 只写 unpadded 切片，未清零的
        # NaN 会污染 e8m0 scale 计算（Quark 路径用 zeros，无此问题）
```

```
        self._zero_padding(layer)

        if self.mxfp4_backend == Mxfp4MoeBackend.NONE:
            layer._already_called_process_weights_after_loading = True
            return

        layer.w13_input_scale = None
        layer.w2_input_scale = None

        w13, w13_scale = _quantize_mxfp4_moe_weight(layer.w13_weight)
        w2, w2_scale = _quantize_mxfp4_moe_weight(layer.w2_weight)

        self._setup_kernel(
            layer,
            w13,
            w2,
            w13_scale,
            w2_scale,
            getattr(layer, "w13_bias", None),
            getattr(layer, "w2_bias", None),
        )

        layer._already_called_process_weights_after_loading = True
```

vllm/model_executor/layers/quantization/utils/mxfp4_utils.py

提供按 OCP MX 规范实现、与硬件 /AITER 逐位对齐的 Triton 动态量化内核（downcast_to_mxfp/_compute_quant_and_scale）与统一入口 mxfp4_quantize，所有后端（XPU/AITER/Triton）共用的量化原语，是本特性的正确性基石。

```
@triton.jit
def _compute_quant_and_scale(src_tensor, valid_src_mask):
    # 每个 block 共享 32 个元素一个 e8m0 scale（OCP MX 规范），
    # 这里把二维 tile 展开为 [OUT_DIM, QUANT_MX_SCALE, 32] 以便按 block 归约
    BLOCK_SIZE_OUT_DIM: tl.constexpr = src_tensor.shape[0]
    BLOCK_SIZE_QUANT_DIM: tl.constexpr = src_tensor.shape[1]
    BLOCK_SIZE_QUANT_MX_SCALE: tl.constexpr = src_tensor.shape[1] // 32

    # 显式转 fp32：bfloat16 上多数算子不受支持，也可避免反复转换
    f32_tensor = src_tensor.to(tl.float32)
    abs_tensor = tl.abs(f32_tensor)
    # padding 区用 -1.0 占位，不参与 scale 计算
    abs_tensor = tl.where(valid_src_mask, abs_tensor, -1.0)
    abs_tensor = tl.reshape(
        abs_tensor, [BLOCK_SIZE_OUT_DIM, BLOCK_SIZE_QUANT_MX_SCALE, 32]
    )
    max_val = tl.max(abs_tensor, axis=2, keep_dims=True)
    # Round-to-nearest-even 计算 e8m0 scale，与硬件 /aiter MXFP4 量化器对齐
    max_val = max_val.to(tl.int32, bitcast=True)
    max_val = (max_val + 0x200000).to(tl.uint32, bitcast=True) & 0x7F800000
```

```
max_val = max_val.to(tl.float32, bitcast=True)
# 加 epsilon 防止全零 block 出现 log2(0) = -inf
eps = tl.where(max_val == 0.0, 2 ** (-126), 0.0)
scale_e8m0_unbiased = tl.log2(max_val + eps).floor() - 2
scale_e8m0_unbiased = tl.clamp(scale_e8m0_unbiased, min=-127, max=127)
dequant_scale_rounded = tl.exp2(scale_e8m0_unbiased)
dequant_scale_exponent = dequant_scale_rounded.to(tl.uint32, bitcast=True)

dequant_scale_rounded = dequant_scale_exponent.to(tl.float32, bitcast=True)
quant_scale = tl.where(dequant_scale_rounded == 0, 0, 1.0 / dequant_scale_rounded)

f32_tensor = tl.reshape(
    f32_tensor, [BLOCK_SIZE_OUT_DIM, BLOCK_SIZE_QUANT_MX_SCALE, 32]
)
quant_tensor = f32_tensor * quant_scale

# 缩放后恢复原形状；padding 区在 MX 格式下必须为 0
quant_tensor = quant_tensor.reshape([BLOCK_SIZE_OUT_DIM, BLOCK_SIZE_QUANT_DIM])
quant_tensor = tl.where(valid_src_mask, quant_tensor, 0)
dequant_scale_exponent = dequant_scale_exponent.reshape(
    [BLOCK_SIZE_OUT_DIM, BLOCK_SIZE_QUANT_MX_SCALE]
)

# 提取 scale 的指数部分并转为 uint8
dequant_scale_exponent = (dequant_scale_exponent >> 23).to(tl.uint8)
# 手动把 fp32 值裁剪为 packed e2m1：先拆符号 / 指数 / 尾数
quant_tensor = quant_tensor.to(tl.uint32, bitcast=True)
signs = quant_tensor & 0x80000000
exponents = (quant_tensor >> 23) & 0xFF
mantissas = quant_tensor & 0x7FFFFF

# 处理 denormal：0.25 <= x < 0.75 映射到 0.5
E8_BIAS = 127
E2_BIAS = 1
# 把隐式 1 移入尾数，移出的位按 OR 折叠成 sticky bit，
# 使 round-to-even 能区分“恰好平局”与“逼近平局”
adjusted_exponents = tl.core.sub(E8_BIAS, exponents + 1, sanitize_overflow=False)
folded_mantissas = 0x400000 | (mantissas >> 1)
denormal_mantissas = folded_mantissas >> adjusted_exponents
denormal_mantissas |= (
    (denormal_mantissas << adjusted_exponents) != folded_mantissas
).to(tl.uint32)
mantissas = tl.where(exponents < E8_BIAS, denormal_mantissas, mantissas)

# 正规数：指数 bias 从 127 改成 1；次正规数指数保持 0
exponents = tl.maximum(exponents, E8_BIAS - E2_BIAS) - (E8_BIAS - E2_BIAS)

# 用“加 rounding bias 再截断”实现 RNE：bias 的 LSB 是保留的尾数位，
# 平局时进位到偶数候选，存在更低位的 sticky bit 时总是进位
```

```
    rounding_bias = ((mantissas >> 22) & 1) + 0x1FFFFF
    rounded_mantissa = (mantissas + rounding_bias) >> 22 # in {0, 1, 2}
    e2m1_tmp = tl.minimum((exponents << 1) + rounded_mantissa, 0x7)
    e2m1_value = ((signs >> 28) | e2m1_tmp).to(tl.uint8)

    # 两个 e2m1 打包进一个 uint8：低位放偶数位，高位放奇数位
    e2m1_value = tl.reshape(
        e2m1_value, [BLOCK_SIZE_OUT_DIM, BLOCK_SIZE_QUANT_DIM // 2, 2]
    )
    evens, odds = tl.split(e2m1_value)
    out_tensor = evens | (odds << 4)

    return out_tensor, dequant_scale_exponent
```

vllm/model_executor/layers/quantization/online/moe_base.py

_zero_padding 从 fp8.py 上移为基类方法，统一所有在线 MoE 方法量化前的 padding 清零语义，修复在线路径 empty_strided 分配与 Quark zeros 分配的差异导致的量化结果不一致。

```
    def _zero_padding(self, layer: torch.nn.Module) -> None:
        # 取未 padding 的真实维度，把 roundup 后多出来的区清零
        hidden_size = layer.moe_config.hidden_dim_unpadded
        intermediate_size = layer.moe_config.intermediate_size_per_partition_unpadded

        # w13 形状为 [experts, w13_num_shards * intermediate, hidden],
        # 每个 shard 的尾部都可能存在 padding，需要逐 shard 清零
        w13_shard = layer.w13_weight.shape[1] // self.moe.w13_num_shards
        if w13_shard > intermediate_size:
            for shard in range(self.moe.w13_num_shards):
                start = shard * w13_shard + intermediate_size
                layer.w13_weight[:, start : (shard + 1) * w13_shard, :] = 0
        if layer.w13_weight.shape[2] > hidden_size:
            layer.w13_weight[:, :, hidden_size:] = 0

        if layer.w2_weight.shape[1] > hidden_size:
            layer.w2_weight[:, hidden_size:, :] = 0
        if layer.w2_weight.shape[2] > intermediate_size:
            layer.w2_weight[:, :, intermediate_size:] = 0

        # 带 bias 的模型（如 GPT-OSS biased MoE）同样需要清零
        if getattr(layer, "w13_bias", None) is not None:
            w13_bias_shard = layer.w13_bias.shape[1] // self.moe.w13_num_shards
            if w13_bias_shard > intermediate_size:
                for shard in range(self.moe.w13_num_shards):
                    start = shard * w13_bias_shard + intermediate_size
                    layer.w13_bias[:, start : (shard + 1) * w13_bias_shard] = 0

        if (
            getattr(layer, "w2_bias", None) is not None
            and layer.w2_bias.shape[1] > hidden_size
```

```
    ):
        layer.w2_bias[:, hidden_size:] = 0
```

评论区精华

- 量化键（QuantKey）语义争议：BowenBao 质疑简写 mxfp4 解析到 kMxfp4Dynamic 让代码定义看起来像 w4a16（"Is activation quantized or not?"、"the code definition today looks rather like w4a16"）；fxmarty 澄清激活量化由 online/base.py L138-145 的默认回退生效；mgoin 认为 "They shouldn't NEED to specify static or dynamic for weights/activations, since there is only one valid choice for each side"。最终保留 "mxfp4": kMxfp4Dynamic 简写并更新文档；嵌套配置 {"moe":{"weight":"mxfp4"}} 会因解析到 kMxfp4Dynamic 而分发表只有 kMxfp4Static 而抛 unsupported-weight ValueError，作者确认该写法当前不支持（activation override 在 main 未实现）。已解决（按设计保留，留待后续）。
- CUDA/SM120 与精度边界（外部用户实测）：bvolpato 在 RTX 5070 Ti 实测报告三点：自动选核选中 FlashInfer 后在 capability 120 上 mm_fp4（cute-dsl）不支持导致启动崩溃；强制 marlin 后 20 题算术 smoke 3/20 vs BF16 的 19/20；Granite MoE 在 CUDA 走 emulation 后端却因缺 amd-quark 启动失败。作者回应：本 PR 范围为 AMD，欢迎单独 PR 修选核；公平对比是 offline MXFP4 + marlin 而非 BF16（"Note that marlin does NOT quantize activations"）；安全回退属于 linear kernel / MoE oracle 的职责。未解决（超出本 PR 范围）。
- is_shuffled 清理时机：fxmarty 自评指出 quark_moe.py 中后端特定的 is_shuffled=True 应移入 convert_to_fp8_moe_kernel_format，并希望 replace_parameter 能迁移属性；经讨论先在 f9ebfe5 revert，等待 PR#49601 合入后再处理。已解决（延后独立处理）。
- 评测配置未挂 CI：mgoin 指出新增 GSM8K 配置（Qwen3-30B-A3B-MXFP4-AITER-TP2-online.yaml 等）没有挂到任何 CI 配置；作者回应目前仅用于 PR 手工验证，如需要可接入。已解决（确认仅手工验证）。
  - mxfp4 简写键映射 kMxfp4Dynamic 与分发注册 kMxfp4Static 不一致 (design): 保留 "mxfp4": kMxfp4Dynamic 简写并更新文档；嵌套 weight 配置当前不受支持（activation override 在 main 未实现），mxfp8 存在同源缺陷待后续统一解决。
  - CUDA（SM120）自动选核失败与小模型精度退化 (correctness): 作者接受部分建议但判定超出本 PR 范围；建议单独开 PR/issue 修选核能力级 guard、补充小模型精度评估、处理 CUDA 回退。
  - quark_moe.py 中 is_shuffled 后端特定代码的清理时机 (refactor): 本 PR 内 revert，延后到 #49601 合并后独立清理。
  - 新增 GSM8K 评测配置未接入 CI (testing): 保留在仓库但未注册 CI，作者表示可后续接入。
  - emulation 后端 is_supported_config 引入 has_quark 门控 (design): 门控按设计保留；测试通过 monkeypatch 绕过 has_quark 验证 is_supported_config 逻辑本身。

风险与影响

- 风险：

- CUDA 平台选核崩溃：auto backend selection 在 SM120 上仍选 FlashInferMxFp4LinearKernel，而 CuTeDSL mm_fp4 不支持 capability 120（vllm/model_executor/kernels/linear/mxfp4/flashinfer.py L24-30），用户实测启动即报 BackendSupportedError；本 PR 未增加能力级 guard，CUDA 用户只能手动 --linear-backend marlin。
- 小模型精度退化：w4a4 对激活也量化，小 dense 模型算术任务退化显著（外部 smoke 3/20 vs BF16 19/20）。作者主张公平对照是 offline MXFP4，但对直接对比 BF16 的用户而言体验落差大，支持边界需在文档中明确。
- 配置键 dynamic/static 不一致：QUANT_KEY_NAMES["mxfp4"] 解析为 kMxfp4Dynamic，而分发表只注册 kMxfp4Static；裸简写经默认激活回退可用，嵌套 {"moe":{"weight":"mxfp4"}} 则抛 ValueError；mxfp8 在 main 上存在同源缺陷（weight=kMxfp8Dynamic）。
- emulation 后端依赖 amd-quark：ocp_mx_emulation_moe.py 的 is_supported_config 在无 amd-quark 时拒绝，但 CUDA 用户在选核阶段仍可能先选中 emulation 再在启动时报 ImportError 类失败，回退责任完全转嫁给 linear/MoE oracle。
- 测试盲区：test_online_mxfp4.py 的 XPU 分支被 pytest.skip 禁用（TODO）；emulation 的 is_supported_config 测试依赖 monkeypatch has_quark；新增 GSM8K 配置未注册 CI，长期回归保障不足。
- MoE padding 正确性：在线路径 empty_strided 分配的 padding 区若未清零，NaN 会污染 e8m0 scale 与量化结果（Quark 路径用 zeros 无此问题）；已由 _zero_padding 上移基类与显式 NaN 测试覆盖，但任何绕过基类的自定义在线 MoE 方法仍有同类风险。

- 影响：
  - 用户：AMD gfx942/gfx950 用户（MI350/MI355 走 AITER，其他 AMD 走 emulation/Triton）可直接对未量化 bf16/fp16 模型启用 MXFP4，获得 FP4 密度的显存 / 带宽收益而无需离线转换；CUDA 用户可强制后端使用但存在启动崩溃与精度退化风险。
  - 系统：在线量化体系从 fp8/int8/nvfp4 扩展到 mxfp4，扩展了 kernel 选择（init_mxfp4_linear_kernel）与 MoE oracle 的分发面；_zero_padding 上移基类让 fp8 等既有在线 MoE 方法共享 padding 清零语义，统一了在线量化路径的隐性约束。
  - 团队 / 协作：29 个 commit 含多次 revert（replace_parameter、static/dynamic 简写、is_shuffled）与 6 次主干合并，跨 18 个文件；review 中 dynamic/static 键语义的讨论暴露了在线量化配置解析的既有设计短板，可能推动后续简写解析重构；AMD 与社区贡献者（bvolpato）的往返也建立了 CUDA 边界的改进议题。
  - 测试与 CI：test_online.py 对 mxfp4 参数在 CUDA 上直接 skip（仅 gfx942/gfx950 运行），避免误报；新增 Quark parity 测试为后续所有 MXFP4 在线后端提供逐字节回归基线。
  - 风险标记：CUDA 能力级缺失，配置键 dynamic/static 不一致，小模型精度退化，emulation 依赖 amd-quark，评测配置未接入 CI，XPU 测试未启用

关联脉络

- PR #49348 Abstract AITER W4A4 as an MXFP4 kernel: PR body 明确列为前置依赖："Abstract AITER W4A4 as an MXFP4 kernel so as to be able to test this PR on

MI350/MI355 (done in PR#49348)"，与本 PR 的 AITER 后端路径直接配套。

- PR #49601 （标题未在材料中给出）quark_moe is_shuffled 相关处理：review 中讨论提及 "Let's get PR#49601 merged first"，commit 31e98fa 也注明 "following #49601"，本 PR 的 is_shuffled 清理依赖它。
- PR #51365 [XPU] quick fix online quantization UT break: 与本 PR 共同修改 tests/quantization/test_online.py 的 XPU 在线量化路径，修复硬编码 cuda 导致的单测失败。
- PR #51357 Fix ROCm architecture import on non-ROCm platforms: 同改 vllm/model_executor/layers/fused_moe/oracle/mxfp4.py 与 mxfp4 量化相关导入路径，修复非 ROCm 平台导入触发 torch.cuda 初始化的问题。
- PR #50833 [Bugfix][Quantization] Fix dynamic INT8 W8A8 MoE config being built as W8A16: 同属在线量化配置键（dynamic/static）语义缺陷的修复，与本 PR 中 kMxfp4Dynamic/kMxfp4Static 映射讨论相互印证。