PR #46765 完整报告

vllm-project/vllm

[ROCm][Quantization][5/N] Refactor quark_moe w8a8-int8 w/ oracle

合并时间：2026-07-28 05:01

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46765

执行摘要

- 一句话：重构 INT8 MoE 路径，集成 oracle 后端选择
- 推荐动作：这份 PR 是 oracle 重构系列的重要一环，适合以下角色精读：
- 框架开发者：学习如何在现有 MoE 方法中集成 oracle 选择机制，以及如何优雅处理重构中的行为保持要求。
- ROCm 平台工程师：了解如何为新硬件扩展量化方案支持（_supports_quant_scheme）。
- QA & CI 负责：参考 test_int8_moe_oracle.py 的测试设计原则（平台无关、仅测试选择逻辑不启动内核）。
- 架构师：关注 static-activation INT8 路径的遗留问题及后续迁移计划。

功能与动机

继续 ROCm Quark MoE oracle 重构系列（已完成 w4a8、w4a4、fp8），将 w8a8-int8 路径也纳入统一的后端选择机制。此举使不同量化精度复用同一套 oracle 逻辑，提升代码一致性和可维护性，并为 ROCm 平台启用 Triton 原生 INT8 MoE 内核。

实现拆解

实现主要分为 5 步：

1. 核心重构：在 vllm/model_executor/layers/quantization/quark/quark_moe.py 的 QuarkW8A8Int8MoEMethod.__init__ 中，新增 oracle 后端选择。动态激活方案通过 select_int8_moe_backend 选择后端和专家类，并存储供后续使用；静态激活方案保留在 legacy fused_experts 路径。
2. 权重重排与内核构建：在 process_weights_after_loading 中，根据所选后端调用 convert_to_int8_moe_kernel_format 进行格式转换（TRITON 无操作，HUMMING/CPU 需要重排），然后通过 make_int8_moe_kernel 和 make_int8_moe_quant_config 构建模块化内核。
3. 扩展 INT8 量化方案：在 quant_utils.py 添加 kInt8StaticTensorSym 和 kInt8DynamicTensorSym 键；在 triton_moe.py 的 _supports_quant_scheme 中将 device_supports_int8 扩展为包含 ROCm CDNA GPU，并同时支持 per-channel 和 per-tensor 动态激活方案。
4. 改进错误诊断：在 oracle/int8.py 的 select_int8_moe_backend 中，当无后端匹配时，将 NotImplementedError 信息扩展为包含 weight_key 和 activation_key，并提示设置

VLLM_LOGGING_LEVEL=DEBUG 查看原因。

5. 测试与评估配套：新增 tests/quantization/test_int8_moe_oracle.py，包含三组 oracle 选择测试（动态方案、显式 triton 后端、不支持后端报错），平台独立。添加 tests/evals/gsm8k/configs/Qwen1.5-MoE-A2.7B-Chat-INT8.yaml 用于端到端评估。

关键文件：

- tests/quantization/test_int8_moe_oracle.py（模块 INT8 测试；类别 test；类型 test-coverage；符号 _make_int8_moe_config, test_int8_dynamic_schemes_dispatch_to_triton, test_int8_explicit_moe_backend_triton, test_int8_unsupported_moe_backend_raises）：新增文件，提供 INT8 MoE oracle 后端选择的三组单元测试，覆盖动态方案选择 Triton、显式指定 triton 后端和不支持后端报错，是保证重构正确性的核心测试。
- vllm/model_executor/layers/quantization/quark/quark_moe.py（模块 量化层；类别 source；类型 core-logic；符号 QuarkW8A8Int8MoEMethod.init, QuarkW8A8Int8MoEMethod.create_weights, QuarkW8A8Int8MoEMethod.process_weights_after_loading, QuarkW8A8Int8MoEMethod.apply）：核心重构文件，修改 QuarkW8A8Int8MoEMethod，集成 oracle 选择 INT8 后端，动态激活路径通过模块化内核，静态激活路径保留 legacy 路径，并完成权重重排转换。
- vllm/model_executor/layers/fused_moe/experts/triton_moe.py（模块 Triton 内核；类别 source；类型 core-logic；符号 TritonExperts._supports_quant_scheme）：扩展 TritonExperts._supports_quant_scheme 以支持 ROCm CDNA GPU 的 INT8，并添加 per-tensor 动态 INT8 方案。
- vllm/model_executor/layers/fused_moe/oracle/int8.py（模块 后端选择；类别 source；类型 core-logic；符号 select_int8_moe_backend）：改进 select_int8_moe_backend 的错误信息，当无后端匹配时输出 weight_key 和 activation_key 以方便调试。
- vllm/model_executor/layers/quantization/utils/quant_utils.py（模块 量化工具；类别 source；类型 data-contract；符号 kInt8StaticTensorSym, kInt8DynamicTensorSym）：添加 kInt8StaticTensorSym 和 kInt8DynamicTensorSym 量化键，用于 per-tensor INT8 方案。
- tests/evals/gsm8k/configs/Qwen1.5-MoE-A2.7B-Chat-INT8.yaml（模块 评估配置；类别 test；类型 test-coverage）：新增 Qwen1.5-MoE INT8 模型的 gsm8k 评估配置，准确性阈值 0.50。
- tests/quantization/test_quark.py（模块 集成测试；类别 test；类型 test-coverage）：更新测试使用的模型名为 amd/tiny-qwen3-moe-w8a8-int8。

关键符号：select_int8_moe_backend, make_int8_moe_kernel, convert_to_int8_moe_kernel_format, TritonExperts._supports_quant_scheme, QuarkW8A8Int8MoEMethod.init, QuarkW8A8Int8MoEMethod.process_weights_after_loading

关键源码片段

tests/quantization/test_int8_moe_oracle.py

新增文件，提供 INT8 MoE oracle 后端选择的三组单元测试，覆盖动态方案选择 Triton、显式指定 triton 后端和不支持后端报错，是保证重构正确性的核心测试。

```
def _make_int8_moe_config(moe_backend: str = "auto") -> FusedMoEConfig:
    """创建 INT8 MoE 配置对象用于测试。"""
    from vllm.model_executor.layers.fused_moe.activation import MoEActivation
    return FusedMoEConfig(
        num_experts=8,
        experts_per_token=2,
        hidden_dim=256,
        intermediate_size=256,
        num_local_experts=8,
        num_logical_experts=8,
        moe_parallel_config=FusedMoEParallelConfig.make_no_parallel(),
        activation=MoEActivation.SILU,
        in_dtype=torch.bfloat16,
        device="cuda",
        routing_method=RoutingMethodType.Renormalize,
        moe_backend=moe_backend,
    )


@requires_int8_moe
@pytest.mark.parametrize(
    "weight_key,activation_key",
    [
        # per-channel 权重 + 动态 per-token 激活
        (kInt8StaticChannelSym, kInt8DynamicTokenSym),
        # per-tensor 权重 + 动态 per-tensor 激活
        (kInt8StaticTensorSym, kInt8DynamicTensorSym),
    ],
)
def test_int8_dynamic_schemes_dispatch_to_triton(weight_key, activation_key):
    """两种动态 INT8 MoE 方案都应选择 Triton 后端。"""
    config = _make_int8_moe_config()
    backend, experts_cls = select_int8_moe_backend(
        config, weight_key=weight_key, activation_key=activation_key
    )
    assert backend == Int8MoeBackend.TRITON, "预期使用 Triton 后端"
    assert experts_cls is not None, "预期获得非空专家类"
```

vllm/model_executor/layers/quantization/quark/quark_moe.py

核心重构文件，修改 QuarkW8A8Int8MoEMethod，集成 oracle 选择 INT8 后端，动态激活路径通过模块化内核，静态激活路径保留 legacy 路径，并完成权重重排转换。

```
class QuarkW8A8Int8MoEMethod(QuarkMoEMethod):
    def __init__(self, weight_config, input_config, moe):
        super().__init__(moe)
        self.weight_quant = weight_config
```

```
        self.input_quant = input_config
        self.weight_qscheme = self.weight_quant.get("qscheme", "per_tensor")
        self.static_input_scales = not self.input_quant.get("is_dynamic", False)

        # --- oracle 集成开始 ---
        # 为模块化内核准备状态变量
        self.moe_quant_config: FusedMoEQuantConfig | None = None
        self.moe_kernel: mk.FusedMoEKernel | None = None
        self.int8_backend: Int8MoeBackend | None = None
        self.experts_cls: type[mk.FusedMoEExperts] | None = None

        # 动态激活 INT8 MoE 通过 oracle + 模块化内核
        # 模块化 TritonExperts 内核接收 float 激活并在内部量化为 int8
        # 因此无法应用加载的静态激活缩放
        # TODO: 静态激活 INT8 暂时保留在 legacy fused_experts 路径中
        # 需要后续迁移到 expert backend
        if not self.static_input_scales:
            # 将 Quark 权重方案映射到 oracle 量化键
            if self.weight_qscheme == "per_channel":
                weight_key = kInt8StaticChannelSym
                activation_key = kInt8DynamicTokenSym
            else:
                weight_key = kInt8StaticTensorSym
                activation_key = kInt8DynamicTensorSym

            self.int8_backend, self.experts_cls = select_int8_moe_backend(
                config=moe,
                weight_key=weight_key,
                activation_key=activation_key,
            )
```

评论区精华

Review 中几个关键讨论：

- 行为保持要求：BowenBao 强调重构不应改变行为，要求保留之前支持的 per-tensor 和 static-activation INT8 MoE。作者 amd-sourjya 通过恢复 per-tensor 动态支持，并将 static-activation 保留在 legacy 路径来达成平衡（添加 TODO 待后续迁移）。
- 测试位置合理性：fxmarty-amd 指出初始测试放在 test_gfx950_moe.py 且要求 ROCm 不合理。amd-sourjya 将其移到新建的 test_int8_moe_oracle.py，仅依赖 Triton INT8 可用性，使测试也能在 CUDA 平台运行。
- 权重重排安全性：fxmarty-amd 提醒不应假设 INT8 内核不需要权重重排，应模仿 CompressedTensors 的 convert_to_int8_moe_kernel_format。amd-sourjya 采纳建议，在 process_weights_after_loading 中加入了通用格式转换。
- CI 失败排查：AndreasKaratzas 指出 AMD CI 失败。作者追踪到模型路由层被 INT8 量化导致 AITER 线性核 w_q.shape % 16 == 0 断言失败。通过重新生成保留路由和 lm_head 为 bf16 的模型解决。

- 重构时是否应保留 per-tensor 和 static-activation INT8 MoE 支持 (design): 已解决: per-tensor 动态支持已恢复，static-activation 保留在 legacy 路径并标记 TODO 待后续迁移。
- INT8 oracle 测试是否应放在 gfx950 特定文件中 (testing): 已解决：测试移到平台无关文件 test_int8_moe_oracle.py，gate 改为 Triton INT8 可用性。
- 处理 INT8 MoE 后端特定权重重排 (correctness): 已解决: process_weights_after_loading 现在调用 convert_to_int8_moe_kernel_format 进行后端相关重排。
- AMD CI 失败：AITER INT8 线性核断言 w_q.shape % 16 == 0 (correctness): 已解决：新模型 amd/tiny-qwen3-moe-w8a8-int8 路由未量化，测试通过。

风险与影响

- 风险:
  1. 静态激活路径未迁移：static-activation INT8 MoE 仍走 legacy fused_experts 路径，未利用模块化内核的优势；后续可能回退或产生不一致。需要跟踪 TODO 完成迁移。
1. 量化方案限制：重构后的 oracle 路径只支持动态激活方案（per-channel 和 per-tensor 权重 + 动态 per-token/per-tensor 激活），如果用户使用静态激活量化，将回退到 legacy 路径，但 legacy 路径在后续可能被移除。
2. ROCm 平台验证：虽然测试通过，但 ROCm 上的 Triton INT8 MoE 内核是新增支持，实际生产环境中可能遇到性能或正确性问题。
3. 模型兼容性：AITER 断言问题说明部分 INT8 量化模型（尤其包含量化路由的）可能不兼容，需要检查模型配置。 - 影响：影响范围中等:
- ROCm 用户：享受 Triton 原生 INT8 MoE 内核支持，性能可能提升。
- CUDA 用户：无功能变化，但 TritonExperts 现在也支持 per-tensor 动态 INT8 方案（之前仅 per-channel），提供更多灵活性。
- Quark INT8 MoE 用户：自动通过 oracle 选择最优后端，无需手动指定 --moe-backend。
- 开发者：统一的 oracle 模式使得添加新量化精度后端更加规范。
- 风险标记：静态激活路径未迁移，仅动态激活 INT8 支持，ROCm INT8 新后端，需后续 PR 完成迁移

关联脉络

- PR #39136 [ROCm][Quantization][1/N] Refactor quark_moe w4a8 w/ oracle: 同一 ROCm Quark MoE oracle 重构系列的前序 PR，本 PR 是该系列的第 5 部分，延续相同模式。
- PR #41436 [ROCm][Quantization][2/N] Refactor quark_moe w4a4 w/ oracle: 同一 ROCm Quark MoE oracle 重构系列的前序 PR。
- PR #43721 [ROCm][Quantization][3/N] Refactor quark_moe fp8 w/ oracle: 同一 ROCm Quark MoE oracle 重构系列的前序 PR。