

PR #50029 完整报告

vllm-project/vllm

[Quantization] Preserve precision in online NVFP4 expert packing

合并时间: 2026-08-06 17:23

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/50029>

执行摘要

- 一句话: 在线 NVFP4 打包改为逐专家直接量化, 去除中间舍入损耗
- 推荐动作: 值得精读。这是一个很小的修复, 但演示了两个关键工程判断: 第一, 量化链路上每个中间 cast 都是精度损失点, 'fold 后先 cast 回原 dtype 再量化' 与 '直接把 FP32 scale 传给量化内核' 在数学上等价、位模式上不等价; 第二, 用 bit-for-bit 一致性测试把这类隐性精度问题锁死。建议同时参考 aoshen02 的基准方法 (f32 复现量化器、302M 元素 dequant 误差比较) 和 #51125 的 w13 shard 分配分析。

功能与动机

在线 NVFP4 打包旧实现把权重当作整体: 先按 expert 算出 FP32 全局 encode scale, 折叠进权重并 cast 回原 dtype, 再用 neutral scale 调用 scaled_fp4_quant。PR body 指出 'That intermediate cast adds a rounding step before the group-16 scales and E2M1 values are selected', 即这次中间 cast 会提前损失精度, 影响 group-16 scale 与 E2M1 值的选择。aoshen02 的独立基准进一步给出结论: 'it isn't a trade — the fix is better on precision,latency,andpeakmemoryatonce.', 并用 Inkling-NVFP4checkpoint (E=256、64 层、experts.w13_weight.scale2 为 F32 [256]) 确认 per-expert scale 契约真实存在。PR 同时明确缩小范围, 通用 MoE 分配问题交由 #51125 处理, 与 #49764 (per-expert scale 跨 TP 全局一致) 互补。

实现拆解

1. 变更入口与核心函数: 改动集中在 vllm/model_executor/layers/quantization/online/nvfp4.py 的 _quantize_moe_weight_to_nvfp4。旧路径将 (E, N, K) 权重视为整体, 先按 expert 维度算 amax 与 FP32 global_scale, 再把 weight.float() * global_scale 折叠并用 .to(weight.dtype) 转回 BF16/FP16, 随后 reshape(-1, k) 以 neutral scale (torch.ones) 调用一次 scaled_fp4_quant。
2. 新实现路径: 先对权重调用 weight.contiguous() 保证内存连续; 随后用列表推导对每个 expert 分别调用 scaled_fp4_quant(expert_weight, expert_scale, is_sf_swizzled_layout=False), zip(weight, global_scale, strict=True) 保证权重与 scale 严格对齐; 最后用 torch.stack 将逐 expert 的 packed weight 与 block scale 组装回 (E, N, K//2)、(E, N, K//16), 输出契约不变。
3. 为什么这样改: 利用 scaled_fp4_quant(w, g) == scaled_fp4_quant(w * g, 1) 的数学恒等关系, 把 scale 折叠延后到量化内核内部处理, 避免在 BF16/FP16 域进行一次有损中间

cast; 同时消除了整张 FP32 scaled 临时张量, 降低峰值内存。逐专家调用后续也便于适配 EP 场景。

4. 测试配套: tests/quantization/test_online.py 新增

test_online_nvfp4_quantizes_original_expert_weights (SM100 skipif 门控), 用随机 bf16 权重验证生产路径与逐 expert 直接调用 scaled_fp4_quant 的结果 torch.equal 逐位一致; 同时新增对 scaled_fp4_quant 与 _quantize_moe_weight_to_nvfp4 的导入。

5. 范围与协作: 该 PR 最初包含更广的 MoE 修复, 按 aoshen02 的 review 意见收缩为仅 NVFP4 打包修复; 通用 non-gated MoE w13 分配问题拆到 #51125 (作者被列为 first author), per-expert scale 跨 TP 共享由 #49764 跟进。

关键文件:

- vllm/model_executor/layers/quantization/online/nvfp4.py (模块 量化层; 类别 source; 类型 core-logic; 符号 _quantize_moe_weight_to_nvfp4): 核心量化路径, _quantize_moe_weight_to_nvfp4 从整体 fold-scale 量化改为逐专家直接量化, 删除中间 cast 与 FP32 临时张量, 是本次精度修复的主体。
- tests/quantization/test_online.py (模块 量化测试; 类别 test; 类型 test-coverage; 符号 test_online_nvfp4_quantizes_original_expert_weights): 新增 SM100 门控的 bit-for-bit 一致性测试, 验证新打包路径与逐专家直接量化等价, 防止回归。

关键符号: _quantize_moe_weight_to_nvfp4, test_online_nvfp4_quantizes_original_expert_weights

关键源码片段

vllm/model_executor/layers/quantization/online/nvfp4.py

核心量化路径, _quantize_moe_weight_to_nvfp4 从整体 fold-scale 量化改为逐专家直接量化, 删除中间 cast 与 FP32 临时张量, 是本次精度修复的主体。

```
def _quantize_moe_weight_to_nvfp4(
    weight: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
    """Quantize stacked MoE expert weights `` (E, N, K) `` to NVFP4.

    One FP32 global scale per expert plus per-block (group-16) FP8 scales,
    matching the ModelOpt NVFP4 checkpoint layout. Returns packed FP4 weights
    `` (E, N, K // 2) `` , block scales `` (E, N, K // 16) `` , and the per-expert
    global scale `` (E,) `` stored as ``amax / (fp4_max * fp8_max)``.
    """
    assert weight.dim() == 3, f"expected 3D expert weights, got {weight.shape}"
    k = weight.shape[-1]
    assert k % 16 == 0, f"last dim must be a multiple of 16, got {k}"

    # 先对整个 E×N×K 张量求出每个 expert 的 amax, 再换算成 FP32 全局 encode scale。
    # 该 scale 与 ModelOpt NVFP4 checkpoint 中的 per-expert scale 契约保持一致。
    amax = weight.abs().amax(dim=(1, 2)).to(torch.float32).clamp_min(1e-8)
    global_scale = (FLOAT4_E2M1_MAX * FLOAT8_E4M3_MAX) / amax
    weight_scale_2 = (1.0 / global_scale).to(torch.float32)
```

```

# 保持原始 BF16/FP16 权重作为量化器输入：旧实现先把 FP32 全局 scale 折叠
# 进位宽并 cast 回原 dtype，这会在 group-16 scale 与 E2M1 值选取前引入一次
# 额外舍入。这里改为逐 expert 直接把权重与 FP32 scale 传给 scaled_fp4_quant。
weight = weight.contiguous()
quantized_experts = [
    scaled_fp4_quant(
        expert_weight,
        expert_scale,
        is_sf_swizzled_layout=False,
    )
    for expert_weight, expert_scale in zip(
        weight,
        global_scale,
        strict=True, # strict=True 保证权重与 scale 数量严格对齐
    )
]
# 逐 expert 结果重新 stack 回 (E, N, K//2) 与 (E, N, K//16), shape 契约不变。
qweight = torch.stack([quantized for quantized, _ in quantized_experts])
block_scale = torch.stack([block_scale for _, block_scale in quantized_experts])
return (
    qweight,
    block_scale,
    weight_scale_2,
)

```

tests/quantization/test_online.py

新增 SM100 门控的 bit-for-bit 一致性测试，验证新打包路径与逐专家直接量化等价，防止回归。

```

@pytest.mark.skipif(
    not (
        current_platform.is_cuda() and current_platform.is_device_capability_family(100)
    ),
    reason="NVFP4 weight quantization needs a Blackwell (SM100) GPU.",
)
def test_online_nvfp4_quantizes_original_expert_weights() -> None:
    torch.manual_seed(0)
    # 随机 bf16 权重形状 (E=2, N=32, K=32), K 是 16 的倍数以满足打包约束。
    weight = torch.randn(2, 32, 32, device="cuda", dtype=torch.bfloat16)

    # 生产打包路径，与加载时量化逻辑一致。
    quantized, block_scale, global_decode_scale = _quantize_moe_weight_to_nvfp4(weight)
    # global_decode_scale 是 1 / encode_scale, 先换算回 encode scale 再逐专家量化。
    global_encode_scale = 1.0 / global_decode_scale
    expected = [
        scaled_fp4_quant(
            expert_weight,
            expert_scale,
            is_sf_swizzled_layout=False,

```

```

    )
    for expert_weight, expert_scale in zip(
        weight,
        global_encode_scale,
        strict=True,
    )
]

# 逐位断言：新路径必须与逐 expert 直接调用 scaled_fp4_quant 完全一致。
assert torch.equal(
    quantized,
    torch.stack([expert_weight for expert_weight, _ in expected]),
)
assert torch.equal(
    block_scale,
    torch.stack([expert_scale for _, expert_scale in expected]),
)

```

评论区精华

讨论主要在 issue/PR 评论中进行（review_comments 为 0，claude[bot] 因 fork 自动 review 被禁用）。核心交锋如下：

- aoshen02 对 online/nvfp4.py 的一半做了独立基准：'it isn't a trade — the fix is better on precision, latency, and peak memory at once.'。它用 thinkingmachines/Inkling-NVFP4 checkpoint (E=256、hidden=6144、intermediate=3072、gated w13、64 MoE 层) 确认 experts.w13_weight.scale2 是 F32 [256] 的真 per-expert scale，并在 fp32 下按 nvfp4_utils.cuh::cvt_warp_fp16_to_fp4 复现量化器，对 (8, 6144, 6144) 共 302M 元素比较 dequant 误差，结论是该修复在精度、延迟、峰值内存三方面同时占优。
- aoshen02 要求收窄范围：'could you shrink this pr to keep the nvfp4 fix only, then this PR LGTM, I've open another one to fix all related issues and list you as first author.' 作者随后将 PR 从 draft 转 ready，通用 MoE 分配问题转入 #51125。
- CI 权限：aoshen02 两次 /ci run 因无 write access 被拒，改由作者 S1ro1 触发 Buildkite #82504、#82581。
- per-expert loop 是否是性能 trade-off (performance)：逐 expert 循环不会造成性能回退，反而在精度、延迟、峰值内存上同时更优，PR 继续。
- PR 范围收缩，拆分到 #51125 (design)：本 PR 缩小为仅 NVFP4 打包修复，通用 non-gated MoE w13 分配问题移交 #51125。
- CI 触发权限 (other)：由作者触发 CI 成功完成。

风险与影响

- 风险：
 - 精度行为变化：量化输出 (packed weight、block scale) 因去除中间舍入而逐位不同于旧版本，凡是离线缓存、权重哈希或对比测试依赖旧位型的场景需要重新生成基线；这是

预期修正而非回归。

- 平台覆盖：新测试仅在 CUDA + SM100 上运行，其他平台无覆盖；
_quantize_moe_weight_to_nvfp4 本身依赖自定义算子 scaled_fp4_quant 的可用性。
- 性能不确定性：逐 expert 循环在 E 很大（256 专家 × 64 层）时会增加 kernel 启动次数；aoshen02 的独立基准说明实际持平或更优，但该数据未完整进入 PR，建议在大模型上复测 load 时间。
- 依赖变化：实现从单一整体调用改为逐专家调用后再 stack，若未来 scaled_fp4_quant 的 per-call 语义（如 swizzle 布局）变化，需要同步维护本路径。
- 影响：
 - 用户影响：Blackwell（SM100）上使用 nvfp4_per_token 在线量化的 MoE 模型加载时量化精度更高，峰值内存略降；运行时推理 kernel 与 checkpoint 均不变化，无需迁移。
 - 系统影响：load-time 路径从一次大调用变为 E 次小调用加 stack，shape 契约保持 (E, N, K//2) 与 (E, N, K//16)，对下游 FlashInfer TRTLLM 内核透明。
 - 团队影响：与 #51125、#49764 组成在线量化打包正确性系列修复，为后续 per-expert scale 跨 TP 全局一致（#49764）铺路。
 - 风险标记：精度行为逐位变化，Blackwell 专属测试覆盖，逐专家循环启动开销，依赖 scaled_fp4_quant 语义

关联脉络

- PR #49764 [Quantization] Share online weight scales across TP: PR body 明确说明本修复与 #49764 互补：#49764 使 per-expert scale 跨 TP ranks 全局一致，本修复在此前提下才真正成立。
- PR #51125 [Bugfix] Size and iterate w13 by shard count for non-gated MoE: 讨论中 aoshen02 要求把通用 MoE 分配问题从本 PR 拆出，作者在 #51125 中配合修复 non-gated MoE w13 shard 数量问题并被列为 first author。