

# PR #49764 完整报告

vllm-project/vllm

[Quantization] Share online weight scales across TP

合并时间: 2026-08-07 07:34

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/49764>

## 执行摘要

- 一句话: 在线量化 `amax` 跨 TP 归约, 保证权重量化跨拓扑一致
- 推荐动作: 值得精读。三个设计决策尤其有借鉴价值: (1) `_is_tp_sharded` 用 "amax 归约维 vs 权重切分维" 的交集判定是否需要通信, 避免对列并行 `per-channel`、EP 等本就全局的场景做无谓 `collective`; (2) MoE 归约组直接复用 EP 进程组而非新建组, 简洁且与 `DP x PCP x TP` 展平切分严格对应; (3) 测试用 `stub` 进程组 + `fake all-reduce` 在单进程中验证多卡语义, 并把断言写进 `fake` 函数内部, 方法可推广到其他分布式工具函数测试。

## 功能与动机

PR body 明确指出问题: "Online weight quantization currently derives scales from rank-local shards. Consequently, tensor-parallel packing can use a different recipe from unsharded or expert-parallel packing of the same BF16 checkpoint." 最终合入版本的重写提交 (aoshen02 的 2f89363) 进一步解释了动机: "The result is self-consistent but not reproducible across topologies, which breaks eval comparisons, bisection, and RL rollouts where the trainer holds BF16 or a different sharding." 即问题不只是精度, 而是同一 checkpoint 在不同并行拓扑下不可复现, 直接影响 RL 训练器与 BF16 权重之间的衔接。

## 实现拆解

实现按 4 步拆解:

1. 定义公共 `amax` 推导与归约工具 (`quant_utils.py`, 变更入口): 新增 `weight_amax` (用 `torch.aminmax` 一次遍历同时取 `lo / hi`, 再对二者取绝对值后 `maximum`, 避免为全量权重物化 `abs()` 中间张量)、`amax_for_tp_weight_quant` (`dense` 权重在 `get_tp_group().device_group` 上做 `ReduceOp.MAX` all-reduce, 仅在 `is_sharded` 时执行)、`amax_for_moe_weight_quant` (逐 expert `amax` 在 `get_ep_group().device_group` 上归约, `moe_tp_size == 1` 即 EP 模式时跳过)。选 EP 组的依据是: MoE 关闭 EP 时张量切分按 `DP x PCP x TP` 展平, `span` 恰等于 EP 进程组。
2. FP8 在线量化改造 (`fp8.py`, 主战场, +79/-19): 新增 `_fp8_max`、`_fp8_scale`、`_fp8_channel_scale`、`_fp8_quant_per_channel`、`_is_tp_sharded` 五个 helper。四个 `process_weights_after_loading` 入口都改为 "先算 `amax` → 按需归约 → 再用固定 `scale` 量化": (a) `per-tensor linear` 对整体 `amax` 经 `amax_for_tp_weight_quant` 归约, 行 / 列并行都需要; (b) `per-channel linear` 用 `_is_tp_sharded(layer, reduces_output_dim=False)` 判定, 列并行分片的 `per-channel` 归约维 (输入维) 未被切分

, 可免通信; (c) per-tensor MoE 对逐 expert amax 归约后再算 scale; (d) PTPC MoE 的 w2 改为 per-channel amax 归约 + `_fp8_quant_per_channel` 量化, w13 保持原动态量化路径。实现细节上 `_fp8_max` 用 0 维 tensor 作除数以保持真除法语义 (除以 Python float 会被 PyTorch 降级为乘倒数), `_fp8_quant_per_channel` 按 `_QUANT_CHUNK_ELEMS = 16M` 元素分块, 限制 fp32 中间值峰值约 64 MB。

3. NVFP4 与 INT8 MoE 同步对齐 (nvfp4.py、int8.py) :

`_quantize_moe_weight_to_nvfp4` 增加 `moe_tp_size: int = 1` 参数, per-expert global scale 先经 `amax_for_moe_weight_quant` 归约再参与 `FLOAT4_E2M1_MAX * FLOAT8_E4M3_MAX / amax` 计算; INT8 的 w2 per-row scale 改为先整体算 `w2_amax = weight_amax(layer.w2_weight, dim=-1)` 经归约后逐 expert 复用 (w2 的 amax 归约维正是 TP 切分维), w13 因归约维未被切分而保持逐 expert 原逻辑。TP=1 时两条路径数值均与原实现完全一致。

4. 测试配套 (tests/quantization/test\_online.py, +179) : 新增 `_patch_max_reduce` 桩 (stub 进程组 + 假 all\_reduce 断言 `ReduceOp.MAX` 与 group, 再把未分片权重算出的 full\_amax 复制进 tensor, 等价于真实多卡归约结果), 以及 `_quantize_linear`、`_quantize_moe` 两个参考实现工具函数。三个 parity 测试分别覆盖 dense per-tensor / per-channel (含两种切分维)、MoE per-tensor / per-channel / NVFP4 w2 切分、INT8 w2 切分, 全部断言打包权重与 scale 逐元素相等; 另有 `test_is_tp_sharded_false_when_scale_is_already_global` 固化 " 无需通信 " 的判定语义。PR body 说明无需文档改动, 无 schema / 部署配套变更。

关键文件:

- vllm/model\_executor/layers/quantization/utils/quant\_utils.py (模块 量化工具; 类别 source; 类型 data-contract; 符号 `weight_amax`, `amax_for_tp_weight_quant`, `amax_for_moe_weight_quant`) : 公共工具层, 新增 `weight_amax`、`amax_for_tp_weight_quant`、`amax_for_moe_weight_quant` 三个函数, 是全部在线量化路径的归约入口; EP 组复用与 `is_sharded` 跳过语义都在这里定义。
- vllm/model\_executor/layers/quantization/online/fp8.py (模块 在线量化; 类别 source; 类型 data-contract; 符号 `_fp8_max`, `_fp8_scale`, `_fp8_channel_scale`, `_fp8_quant_per_channel`) : 主战场 (+79/-19), 新增 `_fp8_max / _fp8_scale / _fp8_channel_scale / _fp8_quant_per_channel / _is_tp_sharded`, 四个 `process_weights_after_loading` 入口全部改造为 " 先算 amax、按需归约、再量化 "。
- tests/quantization/test\_online.py (模块 量化测试; 类别 test; 类型 test-coverage; 符号 `_patch_max_reduce`, `fake_all_reduce`, `test_is_tp_sharded_false_when_scale_is_already_global`, `_quantize_linear`) : 测试配套 (+179), 新增 fake all-reduce 桩与三个跨分片 parity 测试, 用单进程验证多卡归约语义, 是本 PR 正确性论证的核心证据。
- vllm/model\_executor/layers/quantization/online/nvfp4.py (模块 在线量化; 类别 source; 类型 data-contract; 符号 `_quantize_moe_weight_to_nvfp4`) : NVFP4 MoE 量化入口同步改造: `_quantize_moe_weight_to_nvfp4` 增加 `moe_tp_size` 参数, per-expert global scale 先经 EP 组归约再参与打包。
- vllm/model\_executor/layers/quantization/online/int8.py (模块 在线量化; 类别 source; 类型 data-contract; 符号 `Int8OnlineMoEMethod._quantize_weights`) : INT8 MoE w2

的 per-row scale 改为整体 amax + 归约后复用，修复 w2 中间维被 TP 切分时 scale 不一致的问题；w13 因归约维未切分保持原逻辑。

关键符号：weight\_amax, amax\_for\_tp\_weight\_quant, amax\_for\_moe\_weight\_quant, \_is\_tp\_sharded, \_fp8\_max, \_fp8\_scale, \_fp8\_channel\_scale, \_fp8\_quant\_per\_channel, \_quantize\_moe\_weight\_to\_nvfp4, Int8OnlineMoEMethod.\_quantize\_weights

## 关键源码片段

### vllm/model\_executor/layers/quantization/utils/quant\_utils.py

公共工具层，新增 weight\_amax、amax\_for\_tp\_weight\_quant、amax\_for\_moe\_weight\_quant 三个函数，是全部在线量化路径的归约入口；EP 组复用与 is\_sharded 跳过语义都在这里定义。

```
# quant_utils.py: 在线量化共用的 scale 推导工具
# 核心思路：同一份 BF16 checkpoint 在不同并行拓扑下必须量化出完全一致的
# 结果，因此在推导 scale 之前，把 amax（绝对值最大值）在对应归约组内做
# MAX all-reduce，且只在真正需要时发起通信。
```

```
from vllm.distributed.parallel_state import get_ep_group, get_tp_group
```

```
def weight_amax(
    weight: torch.Tensor, *, dim: int | None = None, keepdim: bool = False
) -> torch.Tensor:
    """计算 max(lweightl)，且不物化全尺寸的 abs() 中间张量。

    torch.aminmax 一次遍历同时得到最小值与最大值，再分别取绝对值后
    maximum，内存占用远低于 weight.abs().amax()，适合加载期量化超大权重。
    """
    lo, hi = weight.aminmax(dim=dim, keepdim=keepdim)
    return torch.maximum(lo.abs(), hi.abs())
```

```
def amax_for_tp_weight_quant(amax: torch.Tensor, is_sharded: bool) -> torch.Tensor:
    """当权重确实沿 amax 归约的维度被 TP 切分时，在 TP 组内做 MAX
    all-reduce，使每个分片推导出的 scale 与整权重一致。
```

```
is_sharded 由 fp8.py 的 _is_tp_sharded 判定：行并行切输入维、列并行切
输出维；若 amax 只归约输入维（如 per-channel），列并行分片本就与未分片
等价，无需通信。
"""
```

```
if is_sharded:
    torch.distributed.all_reduce(
        amax,
        op=torch.distributed.ReduceOp.MAX,
        group=get_tp_group().device_group,
    )
return amax
```

```
def amax_for_moe_weight_quant(amax: torch.Tensor, moe_tp_size: int) -> torch.Tensor:
    """把逐 expert 的 amax 在 MoE 张量切分组内做 MAX all-reduce。

    MoE 权重在 EP 关闭时按 DP x PCP x TP 展平切分，span 恰好等于 EP 进程
    组，因此复用 get_ep_group(); EP 模式下 moe_tp_size == 1，每个 rank
    持有完整 expert，跳过通信。
    """
    if moe_tp_size > 1:
        torch.distributed.all_reduce(
            amax,
            op=torch.distributed.ReduceOp.MAX,
            group=get_ep_group().device_group,
        )
    return amax
```

### vllm/model\_executor/layers/quantization/online/fp8.py

主战场 (+79/-19)，新增 `_fp8_max / _fp8_scale / _fp8_channel_scale / _fp8_quant_per_channel / _is_tp_sharded`，四个 `process_weights_after_loading` 入口全部改造为 "先算 amax、按需归约、再量化"。

- # fp8.py: 在线 FP8 量化的 scale 推导辅助函数
- # 关键点: scale 由全局 amax 推导; 量化 kernel 的除法语义 (真除法 vs 乘倒数) 必须被精确复刻, 否则会在可表示中点附近产生不一致的舍入。

```
def _fp8_max(device: torch.device) -> torch.Tensor:
    # 用 0 维 tensor 而非 Python float 作除数: float 除法会被 PyTorch
    # 降级为乘倒数, 与 kernel 内的真除法在舍入上不一致。
    return torch.tensor(get_fp8_min_max()[1], device=device, dtype=torch.float32)
```

```
def _fp8_scale(amax: torch.Tensor) -> torch.Tensor:
    """amax / fp8_max, 即 per-tensor / per-expert 的 scale."""
    amax = amax.to(torch.float32)
    return amax / _fp8_max(amax.device)
```

```
def _fp8_channel_scale(amax: torch.Tensor) -> torch.Tensor:
    """per-channel scale 增加下界保护, 避免极小 amax 产生除零 / 下溢."""
    fp8_max = _fp8_max(amax.device)
    return _fp8_scale(amax).clamp_min(1.0 / (fp8_max * 512))
```

```
def _is_tp_sharded(layer: Module, *, reduces_output_dim: bool = True) -> bool:
    """判断权重是否沿 amax 归约的维度被 TP 切分。
```

行并行切输入维, 列并行切输出维; per-output-channel 的 scale 只归约输入维, 所以列并行层本身已与未分片等价, `reduces_output_dim=False` 时返回 False, 从而跳过无谓的 collective。  
"""

```

if layer.tp_size == 1:
    return False
is_row_parallel = layer.input_size != layer.input_size_per_partition
is_column_parallel = layer.output_size != layer.output_size_per_partition
return is_row_parallel or (reduces_output_dim and is_column_parallel)

```

```

def _fp8_quant_per_channel(weight: torch.Tensor, scale: torch.Tensor) -> torch.Tensor:
    """按 channel 做真除法量化，逐块执行以控制 fp32 中间值峰值约 64 MB。"""
    fp8_min, fp8_max = get_fp8_min_max()
    out = torch.empty_like(weight, dtype=current_platform.fp8_dtype())
    chunk_rows = max(1, _QUANT_CHUNK_ELEMS // weight.shape[-1])
    for start in range(0, weight.shape[0], chunk_rows):
        rows = slice(start, start + chunk_rows)
        out[rows] = (weight[rows] / scale[rows]).clamp_(fp8_min, fp8_max).to(out.dtype)
    return out

```

## tests/quantization/test\_online.py

测试配套 (+179)，新增 fake all-reduce 桩与三个跨分片 parity 测试，用单进程验证多卡归约语义，是本 PR 正确性论证的核心证据。

```

# test_online.py: 用桩进程组 + 假 all_reduce 为单进程单元测试提供归约语义
# 注意: helper 提升为模块级导入后，测试补丁点也要跟随到名字被查找的位置，
# 即 quant_utils.get_tp_group / get_ep_group，而不是 vllm.distributed.parallel_state。

```

```

def _patch_max_reduce(monkeypatch, full_amax) -> None:
    """替身 MAX all-reduce: 断言 op 为 ReduceOp.MAX、group 为预期桩对象，
    然后把未分片权重算出的 full_amax 复制进被归约的 tensor —— 等价于多
    rank 环境中 MAX 归约后的最终结果。
    """
    expected = cast(ProcessGroup, object())
    stub = SimpleNamespace(device_group=expected)
    monkeypatch.setattr(quant_utils, "get_tp_group", lambda: stub)
    monkeypatch.setattr(quant_utils, "get_ep_group", lambda: stub)

    def fake_all_reduce(tensor, op, group):
        assert op == torch.distributed.ReduceOp.MAX
        assert group is expected
        tensor.copy_(full_amax)

    monkeypatch.setattr(torch.distributed, "all_reduce", fake_all_reduce)

```

## 评论区精华

核心 review 讨论集中在代码风格与导入时机，最终由作者在提交中完整回应：

- Isotr0py 在 quant\_utils.py 的 diff 上连续两次评论: "Can we move this to top-level import?" 与 "ditto."，针对两个 amax helper 函数体内的 from vllm.distributed.parallel\_state import get\_tp\_group 局部导入。

- aoshen02 在提交 0a0352b 中将其提升为模块级导入，并说明：vllm/distributed 下没有任何模块导入 quant\_utils，不存在循环依赖风险；同时把测试的 monkeypatch 目标从 vllm.distributed.parallel\_state.get\_tp\_group 改为 quant\_utils.get\_tp\_group——因为模块级导入后，补丁必须打在名字实际被查找的位置。
- Isotr0py 最终 APPROVED ("LGTM, leave a nit")。另一个值得注意的讨论点来自 fork 协作：aoshen02 把整 PR 重写为 "Make online weight scales TP-invariant (rewrite of #49764)" 并通过 fork PR #7 / #8 回合，最终合入版本由重写 + 导入提升两个 fork 提交构成。
- quant\_utils 内函数级导入应提升为模块级 (style): aoshen02 在提交 0a0352b 中提升为模块级导入，说明 vllm/distributed 无反向依赖 quant\_utils、不存在循环导入；同时把测试的 monkeypatch 目标从 vllm.distributed.parallel\_state.get\_tp\_group 改为 quant\_utils.get\_tp\_group，保证补丁点在名字实际被查找的位置。Isotr0py 随后 APPROVED。
- PR 由 aoshen02 重写并经 fork PR 回合 (other): 两次 fork PR (#7、#8) 均被 S1ro1 合并，head 最终态为 aoshen02 主导的重写版本；中间经历多轮 CI (Buildkite #82582 / #82661 / #82672 / #82684) 与 pre-commit 修复。

## 风险与影响

- 风险：
  1. 分布式归约挂起风险：amax\_for\_tp\_weight\_quant / amax\_for\_moe\_weight\_quant 在 process\_weights\_after\_loading 中发起 all-reduce，该路径由所有 rank 并行执行；若进程组未初始化（如非标准单进程加载路径）或实际并行拓扑与假设不符（MoE 张量切分 span 不等于 EP 组），可能出现挂起或错误归约。当前正确性依赖 "关闭 EP 时 MoE 张量切分按 DP x PCP x TP 展开，恰等于 EP 进程组" 这一前提。
  2. 量化输出数值变更：TP > 1 场景下在线量化产出的打包权重与 scale 会与合入前不同（更正确但不兼容），同一 checkpoint 的生成输出与 benchmark 数值会变化，需要用户重新评估回归结果。
  3. INT8 w2 scale 路径重构：Int8OnlineMoEMethod.\_quantize\_weights 中 w2 从逐 expert 循环内计算 scale 改为整体 amax + 归约后复用，TP=1 时数值不变，但属于行为敏感路径的代码重构，单测仅有 fake 归约覆盖，缺少真实多卡 parity 测试。
  4. 性能：仅为加载期一次性通信（dense 为单个 scalar，MoE 为 per-expert 向量），热路径无影响；\_fp8\_quant\_per\_channel 分块把 fp32 中间值峰值控制在约 64 MB，无显存风险。- 影响：用户侧：使用在线量化（如 fp8\_per\_tensor、fp8\_per\_block、nvfp4\_per\_token）且 TP > 1 的服务，加载后的权重数值会变化，但跨拓扑一致性显著提升；EP 用户数值不变且无额外通信开销。系统侧：改动全部集中在模型加载期的权重处理（process\_weights\_after\_loading），不进入 serving 热路径，推理延迟与吞吐无影响。团队侧：为 "同一 checkpoint 跨拓扑可复现量化" 建立了公共工具（quant\_utils 中的三个函数）与单进程内模拟多卡归约的测试范式，RL rollout、eval 对比、bisection 场景直接受益；后续新增在线量化路径可直接复用这套归约约定。- 风险标记：引入分布式 collective，量化权重数值行为变更，依赖 EP 组拓扑假设，单元测试用 fake 归约

## 关联脉络

- PR #49601 [Weight processing] Copy over new\_data attributes in replace\_parameter:  
同属权重加载 / 量化链路：本 PR 的 FP8 / NVFP4 / INT8 在线量化路径大量使用 replace\_parameter 替换权重参数，49601 正是为 replace\_parameter 补齐 new\_data 属性拷贝并清理量化后端重复打标，两者共同构成在线量化权重替换的基础设施。
- PR #47106 [Kernel] Support Nvfp4 Cutedsl Moe Swiglu-oai and Relu2(non-gated)  
Activation: NVFP4 MoE 量化同一功能线：47106 扩展 FlashInfer CuteDSL NVFP4 MoE 的激活与打包支持，本 PR 修改同一 MoE 权重打包入口  
\_quantize\_moe\_weight\_to\_nvfp4 的 scale 推导范围，二者都围绕 NVFP4 MoE 权重布局与量化正确性。