

PR #53793 完整报告

vllm-project/vllm

[Perf][Kernel][Quantization] Fuse ReLU2 with static FP8 activation quantization

合并时间: 2026-09-15 01:26

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53793>

执行摘要

- 一句话: ReLU2 与静态 FP8 量化融合为单个 Triton 内核
- 推荐动作: 值得精读, 尤其是两类读者: 做量化内核的工程师应重点看 `relu2_fp8_quant.py` 中如何用「显式 bf16 往返 + 倒数乘法 + NaN 重注入」把自定义 Triton 内核与既有编译产物对齐到逐字节一致, 以及作者如何用全 BF16 编码空间扫掠来证明这一点; 做框架契约的工程师应重点看 `quant_activation.py` + `base_linear.py` 如何用 `requires_unquantized_input` 把「层上有旁路消费者」表达成能力位, 而不是在模型里特判。需要注意两点: 一是所有性能数字要区分「内核边界」(可靠, 6 倍级) 与「端到端」(欠功率, +1.34% 且解码区间跨零); 二是 LoRA 场景的融合抑制是这次契约泛化的副作用, 若有 LoRA 服务性能敏感场景, 建议后续补一组 LoRA 开启 / 关闭的对比数据。

功能与动机

Nemotron-H 共享专家 MLP 的 `down_proj` 之前走「ReLU2 → QuantFP8 → FP8 GEMM」两段式路径, 激活与量化各产生一次完整读写。PR body 明确目标是: "Fuse BF16 ReLU2 with static per-tensor FP8 activation quantization for compatible Nemotron-H MLP down-projections", 并要求 Triton 生产者 "preserves the observable BF16 rounding boundary"、通过通用 `maybe_fused_act_quant` 接口返回 `QuantizedActivation`, 使消费者直接复用自己的 checkpoint input scale 而不二次量化。该工作挂在激活量化总线上 (#43501), 复用 #51415 的手动融合接口与 #44260 的消费者契约; 被取代的 #53791 是环境变量门控、模型内实现的原型, 作者判定 "No other open PR found in the duplicate-work search implements this ReLU2-to-static-FP8 producer/consumer path"。

实现拆解

1. 新增融合生产者内核 (`vllm/model_executor/layers/fusion/relu2_fp8_quant.py`, 新增 76 行): `_relu_squared_static_fp8_quant_kernel` 在寄存器里完成 `load → ReLU → 平方 → 量化 → store`, 三处细节刻意对齐既有实现: 平方后 `.to(tl.bfloat16).to(tl.float32)` 显式复刻独立 ReLU2 内核的 BF16 舍入边界; 量化写成 `activated * (1.0 / scale)` 而非除法, 以匹配 O2 Inductor 编译出的 QuantFP8; `tl.where(input_is_nan, x, quantized)` 把 NaN 透传到 FP8 转换之后。宿主函数 `relu_squared_static_fp8_quant(x, linear)` 校验 bf16、连续布局、FP32 标量 `input_scale` 同设备, 按 `min(next_power_of_2(x.shape[-1]), 2048)` 定 block、`num_warps` 上限 4, 空张量直接跳过 `launch`, 最后组装 `QuantizedActivation(data, scale, orig_dtype, orig_shape, kFp8StaticTensorSym)`。

2. 注册表与派发改造 (vllm/model_executor/layers/fusion/fused_act_quant.py) :
 _FUSED_ACT_QUANT 新增 (ReLUSquaredActivation, kFp8StaticTensorSym) 条目, 且只在 current_platform.is_cuda() and has_device_capability(90) 时注册, import 期即完成平台裁剪。新增可选的 _FUSED_ACT_QUANT_SUPPORT 谓词表与 _relu_squared_static_fp8_quant_supported, 把「注册键粒度」与「实际可消费输入粒度」分开: maybe_fused_act_quant 现在先取 get_input_quant_key(linear), 再校验谓词, 任一不满足即回退 act_fn(x), 模型中可无条件调用。
3. 契约收口 (维护者1fc034a) (vllm/model_executor/layers/fusion/quant_activation.py)
 : expose_input_quant_key 由写公开属性 layer.input_quant_key 改为写私有的 _input_quant_key, 并新增 get_input_quant_key(layer)——当层声明 requires_unquantized_input 时返回 None。这一步把「另一条消费分支需要原始激活」从模型内的特判变成通用能力位, 也避免生产者绕过契约直接读属性。
4. 跨模块能力位 (vllm/model_executor/layers/linear.py、vllm/lora/layers/base_linear.py、vllm/model_executor/models/nemotron_h.py) : LinearBase 新增类属性 requires_unquantized_input = False, BaseLinearLayerWithLoRA 覆盖为 True (LoRA adapter 分支始终消费原始激活, 与 base layer 是否能吃预量化输入无关); NemotronHMLP.forward 把 x = self.act_fn(x) 换成 x = maybe_fused_act_quant(self.act_fn, x, self.down_proj), 模型侧只有一行改动。
5. 测试与行为锁定: 新增 tests/kernels/test_relu2_fp8_quant.py (229 行), 覆盖 (1,1)/(17,5120)/(128,10240) 等形状、65536 个 BF16 编码 × 7 个 scale 的比特级一致、relu2/quant_fp8 custom op 四种开关组合、O0 无编译、空张量、三类不支持输入的 fallback、以及 torch.compile(fullgraph=True) 下 17/31/63 token 复用同一编译帧 (compile_count == 1); tests/kernels/test_fused_quant_activation.py 用 MockLinearRequiresUnquantized 锁定契约行为, tests/model_executor/test_nemotron_h_quantization.py 用 test_relu2_fp8_fusion_uses_registry 锁定 MLP 调用点, tests/lora/test_layers.py、tests/compile/passes/test_silu_mul_quant_manual_fusion.py、tests/fusion/test_quant_activation_contract.py 同步适配私有属性改名与 LoRA 语义。

层面	文件	关键符号	变更性质
内核	layers/fusion/relu2_fp8_quant.py	_relu_squared_static_fp8_quant_kernel、relu_squared_static_fp8_quant	新增
派发	layers/fusion/fused_act_quant.py	_FUSED_ACT_QUANT_SUPPORT、_relu_squared_static_fp8_quant_supported、maybe_fused_act_quant	语义扩展
契约	layers/fusion/quant_activation.py	get_input_quant_key、expose_input_quant_key	私有化 + 能力隐藏

层面	文件	关键符号	变更性质
消费者	layers/linear.py、 lora/layers/base_linear.py	requires_unquantized_input	跨模块能力位

关键文件：

- vllm/model_executor/layers/fusion/relu2_fp8_quant.py（模块 激活融合；类别 source；类型 core-logic；符号 _relu_squared_static_fp8_quant_kernel, relu_squared_static_fp8_quant）：本 PR 的核心：新增 ReLU2 + 静态 FP8 量化的 Triton 融合生产者，三处数值细节（BF16 舍入边界、倒数乘法、NaN 透传）决定了与既有算子链的逐字节一致性。
- vllm/model_executor/layers/fusion/fused_act_quant.py（模块 激活融合；类别 source；类型 data-contract；符号 _relu_squared_static_fp8_quant_supported, _FUSED_ACT_QUANT_SUPPORT, maybe_fused_act_quant）：手动融合的生产者注册表与派发入口；本 PR 在此加入 ReLU2 条目、引入可选的 per-entry support 谓词，并把派发改走 get_input_quant_key。
- vllm/model_executor/layers/fusion/quant_activation.py（模块 激活融合；类别 source；类型 data-contract；符号 get_input_quant_key, expose_input_quant_key）：消费者契约文件；新增 get_input_quant_key 并把 expose_input_quant_key 改为写私有属性，是维护者收口 LoRA 通用处理与后续所有 producer 的读取入口。
- vllm/model_executor/models/nemotron_h.py（模块 模型接入；类别 source；类型 core-logic；符号 NemotronHMLP.forward）：模型侧的唯一接入点：NemotronHMLP.forward 用 maybe_fused_act_quant 替换直接调用激活函数，是触发融合的一行改动。
- vllm/lora/layers/base_linear.py（模块 低秩适配；类别 source；类型 core-logic；符号 BaseLinearLayerWithLoRA.requires_unquantized_input）：跨模块能力位的落点：LoRA 包装层声明 requires_unquantized_input = True，使所有附带 adapter 分支的线性层退回未量化激活路径。
- vllm/model_executor/layers/linear.py（模块 线性层；类别 source；类型 data-contract；符号 LinearBase.requires_unquantized_input）：为 requires_unquantized_input 提供默认值 False，定义所有线性层参与预量化输入融合的基线语义。
- tests/kernels/test_relu2_fp8_quant.py（模块 内核测试；类别 test；类型 test-coverage；符号 test_relu2_static_fp8_quant, test_relu2_static_fp8_quant_all_bf16_values, test_relu2_static_fp8_quant_falls_back, test_relu2_static_fp8_quant_torch_compile_dynamic）：229 行新增测试构成本 PR 的正确性护栏：全 BF16 编码空间比特级对比、custom op 派发独立、无编译路径、fallback 与动态 shape 单帧编译复用。
- tests/kernels/test_fused_quant_activation.py（模块 内核测试；类别 test；类型 test-coverage；符号 MockLinearRequiresUnquantized, test_maybe_fused_act_quant_preserves_required_unquantized_input）：通用融合接口的契约测试，新增 MockLinearRequiresUnquantized 与对应用例，锁定「旁路消费者需要

原始激活时必须回退」的行为。

关键符号: `relu_squared_static_fp8_quant`, `_relu_squared_static_fp8_quant_kernel`, `_relu_squared_static_fp8_quant_supported`, `maybe_fused_act_quant`, `get_input_quant_key`, `expose_input_quant_key`, `NemotronHMLP.forward`, `test_relu2_static_fp8_quant_all_bf16_values`

关键源码片段

`vllm/model_executor/layers/fusion/relu2_fp8_quant.py`

本 PR 的核心: 新增 ReLU2 + 静态 FP8 量化的 Triton 融合生产者, 三处数值细节 (BF16 舍入边界、倒数乘法、NaN 透传) 决定了与既有算子链的逐字节一致性。

```
# vllm/model_executor/layers/fusion/relu2_fp8_quant.py
@triton.jit
def _relu_squared_static_fp8_quant_kernel(
    x_ptr, scale_ptr, output_ptr, num_elements,
    FP8_MIN: tl.constexpr, FP8_MAX: tl.constexpr, BLOCK_SIZE: tl.constexpr,
):
    offsets = tl.program_id(0) * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)
    mask = offsets < num_elements
    x = tl.load(x_ptr + offsets, mask=mask, other=0.0).to(tl.float32)
    input_is_nan = x != x
    relu = tl.maximum(x, 0.0)
    # 保留独立 ReLU2 内核的 BF16 输出边界: 先平方再显式往返 bf16,
    # 使融合内核与「ReLU2 → QuantFP8」两段式路径逐字节一致。
    activated = (relu * relu).to(tl.bfloat16).to(tl.float32)
    # 对齐 O2 Inductor 编译出的 QuantFP8: 先取倒数再相乘,
    # 而不是直接除以 scale, 否则低位舍入结果会不同。
    quantized = activated * (1.0 / tl.load(scale_ptr))
    quantized = tl.maximum(tl.minimum(quantized, FP8_MAX), FP8_MIN)
    # NaN 需要穿透到 FP8 转换之后, 才能匹配原生算子链的传播行为。
    quantized = tl.where(input_is_nan, x, quantized)
    tl.store(output_ptr + offsets, quantized, mask=mask)

def relu_squared_static_fp8_quant(
    x: torch.Tensor, linear: LinearBase
) -> QuantizedActivation:
    """BF16 ReLU2 followed by static per-tensor FP8 quantization."""
    # 契约前置条件: bf16 + 连续布局 + 同设备 FP32 标量 input_scale,
    # 同一条判定也在 fused_act_quant 的 support 谓词里提前执行。
    assert x.dtype == torch.bfloat16
    assert x.is_contiguous()
    scale = linear.input_scale
    assert scale.dtype == torch.float32
    assert x.device == scale.device
    assert scale.numel() == 1
```

```

output = torch.empty_like(x, dtype=current_platform.fp8_dtype())
if x.numel() != 0:
    # 按最后一维定 block 并封顶，避免 hidden size 很大（如 10240）时
    # 网格与线程配置失控——对应 mgoin 的“cap it a little bit”意见。
    block_size = min(triton.next_power_of_2(x.shape[-1]), 2048)
    num_warps = min(max(block_size // 256, 1), 4)
    grid = lambda meta: (triton.cdiv(x.numel(), meta["BLOCK_SIZE"]),)
    _relu_squared_static_fp8_quant_kernel[grid](
        x, scale, output, x.numel(),
        FP8_MIN=_FP8_MIN, FP8_MAX=_FP8_MAX,
        BLOCK_SIZE=block_size, num_warps=num_warps,
    )

# 直接返回 QuantizedActivation: 下游 FP8 线性层据 quant_key 跳过自己的
# 输入量化，并复用 checkpoint 自带的 input_scale，避免二次量化。
return QuantizedActivation(
    data=output,
    scale=scale,
    orig_dtype=x.dtype,
    orig_shape=x.shape,
    quant_key=kFp8StaticTensorSym,
)

```

vllm/model_executor/layers/fusion/fused_act_quant.py

手动融合的生产者注册表与派发入口；本 PR 在此加入 ReLU2 条目、引入可选的 per-entry support 谓词，并把派发改走 get_input_quant_key。

```

# vllm/model_executor/layers/fusion/fused_act_quant.py
# ( 激活模块类型，消费者 input_quant_key) -> 融合生产者
_FUSED_ACT_QUANT: dict[tuple[type, QuantKey], Callable] = {
    (SiluAndMul, kFp8StaticTensorSym): _silu_and_mul_fp8_static,
}

def _relu_squared_static_fp8_quant_supported(
    act_fn: torch.nn.Module, x: torch.Tensor, linear: LinearBase,
) -> bool:
    """Return whether the ReLU2 static-FP8 producer can consume this input."""
    # 注册表以 ( 激活类型，量化 key) 为粒度，但真正可消费的输入更窄：
    # bf16、连续布局、同设备 FP32 标量 input_scale 缺一不可。
    scale = getattr(linear, "input_scale", None)
    return (
        isinstance(act_fn, ReLUSquaredActivation)
        and x.is_cuda
        and x.dtype == torch.bfloat16
        and x.is_contiguous()
        and isinstance(scale, torch.Tensor)
        and scale.dtype == torch.float32
        and scale.device == x.device
    )

```

```

        and scale.numel() == 1
    )

# 可选谓词表：未登记的条目保持“命中 key 即融合”的旧语义。
_FUSED_ACT_QUANT_SUPPORT: dict[
    tuple[type, QuantKey], Callable[[torch.nn.Module, torch.Tensor, LinearBase], bool]
] = {}

# 该路径只在 SM90+ CUDA 上验证过，因此在 import 期按平台能力注册，
# 不支持的平台连谓词都不会被查询到。
if current_platform.is_cuda() and current_platform.has_device_capability(90):
    _RELU2_STATIC_FP8_KEY = (ReLUSquaredActivation, kFp8StaticTensorSym)
    _FUSED_ACT_QUANT[_RELU2_STATIC_FP8_KEY] = relu_squared_static_fp8_quant
    _FUSED_ACT_QUANT_SUPPORT[_RELU2_STATIC_FP8_KEY] = (
        _relu_squared_static_fp8_quant_supported
    )

def maybe_fused_act_quant(
    act_fn: torch.nn.Module, x: torch.Tensor, linear: LinearBase,
) -> "torch.Tensor | QuantizedActivation":
    """Apply act_fn, fusing the downstream linear's input quant when possible."""
    # 走 get_input_quant_key 而非直接读属性：LoRA 这类需要原始激活的
    # 消费者会把有效 key 隐藏为 None，从而稳定回退到纯激活路径。
    key = get_input_quant_key(linear)
    if key is not None:
        registry_key = (type(act_fn), key)
        producer = _FUSED_ACT_QUANT.get(registry_key)
        support = _FUSED_ACT_QUANT_SUPPORT.get(registry_key)
        # 谓词缺失时默认放行，保证既有生产者行为不变。
        if producer is not None and (support is None or support(act_fn, x, linear)):
            return producer(x, linear)
    return act_fn(x)

```

vllm/model_executor/layers/fusion/quant_activation.py

消费者契约文件；新增 `get_input_quant_key` 并把 `expose_input_quant_key` 改为写私有属性，是维护者收口 LoRA 通用处理与后续所有 `producer` 的读取入口。

```

# vllm/model_executor/layers/fusion/quant_activation.py
def expose_input_quant_key(layer: torch.nn.Module, kernel) -> None:
    """Store the kernel's pre-quantized input key on the layer, if any."""
    # 内核只声明自己能力的“原始”key，写进私有属性；
    # 对外统一经 get_input_quant_key 读取，避免绕过旁路消费方。
    key = kernel.input_quant_key()
    if key is not None:
        layer._input_quant_key = key

def get_input_quant_key(layer: torch.nn.Module) -> QuantKey | None:

```

```

"""Return the pre-quantized input key the complete layer can consume."""
# 只要层上存在任何需要未量化激活的分支（例如 LoRA adapter），
# 就整体关闭预量化输入：生产者回退为 plain activation，两边都安全。
if getattr(layer, "requires_unquantized_input", False):
    return None
return getattr(layer, "_input_quant_key", None)

# vllm/model_executor/layers/linear.py
class LinearBase(PluggableLayer):
    # 默认不阻断预量化输入；需要原始激活的子类自行覆盖为 True。
    requires_unquantized_input = False

# vllm/lora/layers/base_linear.py
class BaseLinearLayerWithLoRA(BaseLayerWithLoRA):
    # LoRA adapter 分支始终消费原始激活，与 base layer 是否能消费
    # 预量化激活无关，因此在这里显式声明为能力位。
    requires_unquantized_input = True

```

评论区精华

1. 方向性改造要求（已解决）：mgoin 在模型文件上直接给出接入路径——"We are just about to land the manual version of ActivationQuantFusionPass using this here #51415, so if you wouldn't mind integrating with that I think it'd be best"。作者接受了这个方向，合并 #51415 并回复 "moved the producer to maybe_fused_act_quant registry keyed by the downstream input_quant_key"，删除了原先带 VLLM_EXPERIMENTAL_NEMOTRON_SHARED_RELU2_FP8_QUANT 环境开关、模型内判断 TP 度与 custom op 开关的 custom op 方案。
2. 内核配置调优（已解决）：mgoin 评论 "Probably worth tuning/capping this at least a little bit for different sizes"，作者把 BLOCK_SIZE 改为 `min(triton.next_power_of_2(x.shape[-1]), 2048)`，num_warps 随 block 放大但封顶 4。
3. 命名与测试门槛（已解决）：mgoin 提 "nit: would just remove the Bf16 for the name overall" 与 "Also requires >= sm90"，最终符号简化为 `relu_squared_static_fp8_quant / _relu_squared_static_fp8_quant_kernel`，测试加上 `requires_sm90`（`current_platform.is_cuda()` and `has_device_capability(90)`）`skipif`。
4. 资格判定收口与 LoRA 通用化（已解决，本 PR 最有价值的设计收敛）：mgoin 追问 "I think we could put most this logic into _relu_squared_static_fp8_quant_supported, any reason not to? Do we really need to care about the custom op choices?"，随后以提交 c1fc034a "Simplify. Fix lora generally and move checks out of model def" 落地：模型里不再有任何 gating，LoRA 的原始激活需求变成 `requires_unquantized_input` 能力位由 `get_input_quant_key` 统一处理，且派发结果不再依赖 `relu2 / quant_fp8` 的 custom op 选择。
5. 位精确性的边界声明（作者自述，未被反驳）：PR 描述明确 "The exhaustive bit-exact claim is specifically against the default, separately compiled O2 native ReLU2 and

QuantFP8 boundary ... It is not an all-custom-op-dispatch or end-to-end model output claim", 并在性能段落承认长输出 campaign "is underpowered and does not establish a decode improvement or rule out a regression".

6. 流程信息: claude[bot] 因 PR 来自 fork 而无法自动评审 ("automated review is disabled"), CodeRabbit 对最近一轮 review 给出 "No actionable comments were generated"; mgoin 两次 APPROVED, 最终由 vllm-bot 合并。
- 接入通用手动融合接口 #51415 而非自建 custom op (design): 废弃环境开关与模型内 gating, 改为在 _FUSED_ACT_QUANT 注册表中以 (激活类型, input_quant_key) 为键注册生产者, 模型侧只保留一行调用。
- 把资格判定上收为 support 谓词并通用化 LoRA 处理 (design): 提交 c1fc034a "Simplify. Fix lora generally and move checks out of model def" 落地: 新增 _FUSED_ACT_QUANT_SUPPORT 谓词表, LoRA 通过 requires_unquantized_input + get_input_quant_key 在通用契约层解决, 派发不再依赖编译模式与 custom op 选择。
- Triton 内核 block 与 warp 配置需要设上限 (performance): 作者回复并实现: "BLOCK_SIZE now uses min(next_power_of_2(x.shape[-1]), 2048), and num_warps scales with it up to 4", 同时保留 x.numel() == 0 时跳过 launch。
- 内核与测试的命名简化 (style): 符号重命名为 relu_squared_static_fp8_quant 与 _relu_squared_static_fp8_quant_kernel, dtype 前提由函数内 assert 与 support 谓词保证。
- 测试需要 SM90+ 门槛 (testing): 新增 requires_sm90 skipif, 覆盖该文件全部用例, 与生产端只在 SM90+ 注册的条件保持一致。
- 位精确断言的适用范围与端到端性能结论的边界 (correctness): 测试侧以 _assert_fp8_bitwise_equal 零容差断言固化该边界, 并在 test_relu2_static_fp8_quant_ignores_custom_op_dispatch 中验证结果与 custom op 开关无关; 端到端性能结论保留为方向性证据。

风险与影响

- 风险:
 1. 位精确性与 Inductor 生成代码强耦合 (正确性): relu2_fp8_quant.py 的数值语义是对着 O2 Inductor 为 QuantFP8 生成的代码写死的——倒数相乘而非除法、BF16 中间舍入、NaN 透传位置。作者用对照实验证明偏离代价真实: 把 BF16 中间舍入去掉后, 两个 shape 分别有 1,116,386 与 2,232,217 个输出字节发生变化。一旦 PyTorch/Inductor 改变代码生成策略 (例如把 1/scale 常量折叠或改回除法), 逐字节一致性会失效。好在测试用 _assert_fp8_bitwise_equal 做了零容差断言, 失败形式是显式报错而不是静默精度漂移——但这意味着该测试对 torch 版本升级敏感。
 2. LoRA 场景的融合被整体关闭 (性能回退面): BaseLinearLayerWithLoRA.requires_unquantized_input = True 是被 get_input_quant_key 全局读取的能力位, 因此不只是本 PR 的 ReLU2 生产者在 LoRA 包装层上被抑制, (SiluAndMul, kFp8StaticTensorSym) 等既有手动融合生产者同样会退回纯激活路径。这是正确性优先的正确取舍, 但会让 LoRA 服务场景失去手动融合收益, PR 描述未给出 LoRA 场景的性能数据。

3. 能力位是隐性契约（可维护性）：requires_unquantized_input 默认为 False，未来任何新增「另一条分支需要原始激活」的消费者如果忘记声明，就会静默收到预量化激活并可能算错。目前只有 tests/lora/test_layers.py::test_lora_linear_requires_unquantized_input 与 MockLinearRequiresUnquantized 两个用例守护这条语义。
4. 谓词表默认放行（设计一致性）：_FUSED_ACT_QUANT_SUPPORT 在 support is None 时无条件放行，即没有谓词的注册条目保持旧语义；新增 producer 若忘记注册谓词，边界检查会缺失。作者以注释 "Optional per-entry predicates preserve the unconditional fallback contract" 显式承认这是一个刻意的宽松默认。
5. 端到端收益证据不足（性能结论风险）：内核边界 6.09x/6.31x 是可靠的（七次反序采样取中位数），但全模型数据来自前驱源码 6a9bb4080b：共享 MLP 内核 -18.79%、prefill-heavy 请求吞吐量 +1.34%（配对 95% 区间 +0.05% ~ +2.64%）、长输出 TPOT 均值 -0.01%（区间 -5.68% ~ +5.99%）。作者自己声明该 campaign "remains directional rather than a current-head claim"，不能据此宣称解码无回退。
6. 平台与回归范围：融合只在 CUDA SM90+ 注册，SM80/A100/ROCm 完全走原路径，回归风险低；但 fused_act_quant.py 在 rebase 期间多次冲突（d40d226 记录 "# Conflicts: vllm/model_executor/layers/fusion/fused_act_quant.py"），说明该文件是同一时间线上的热点，后续 PR 需注意合并顺序。- 影响：用户侧：Nemotron-H 系列（含 Nemotron Ultra NVFP4、550B 级 checkpoint）在 Hopper/Blackwell 上服务时，共享专家 MLP 的 down_proj 前填路径获得约 6 倍的内核边界加速与约 19% 的共享 MLP 内核时间下降，端到端体现为小幅 prefill 吞吐量改善；其他模型、其他量化方案、SM90 以下设备行为完全不变。系统侧：QuantizedActivation 的消费者契约被扩展出一条通用能力位 requires_unquantized_input，expose_input_quant_key 的属性私有化改变了所有读取 input_quant_key 的调用方（maybe_fused_act_quant、测试 mock 同步改为 _input_quant_key），未来新增手动融合 producer 必须同时考虑能力键、support 谓词和旁路消费者三者。团队侧：本 PR 与 #51415（手动融合接口）、#44260（消费者契约）共同确立了「注册表 + 可选谓词 + 能力位隐藏」的手动融合范式，后续每迁移一种 (activation, quant key) 组合都可复用同一骨架；维护者 c1fc034a 的收口也把「不要往模型代码里塞 gating」变成了显式评审标准。- 风险标记：位精确性与 Inductor 生成代码强耦合，LoRA 场景手动融合被整体关闭，requires_unquantized_input 属隐性契约易漏声明，support 谓词缺失时默认放行，端到端收益证据欠功率，融合文件多次 rebase 冲突

关联脉络

- PR #55309 [Qwen3.8-Flash-Next] Fuse PLE residual and QSA output gate: 同为「用 Triton 内核算术融合换取性能、同时保持精确舍入边界」的同类工作，可对比两处融合如何定义各自的一致性 oracle（该 PR 对齐 BF16 舍入边界，本 PR 对齐 O2 Inductor 的倒数乘法与 NaN 传播）。
- PR #55071 [LoRA][Refactor] Unify multimodal LoRA token count hooks: 同样在 LoRA 与模型接口之间建立通用钩子 / 契约；本 PR 向 BaseLinearLayerWithLoRA 注入 requires_unquantized_input 能力位属于同一类「把模型特殊需求收敛到统一接口」的演进。
- PR #54965 [ROCm][Perf] W4A16: keep skinny GEMM zero-points packed 4-bit: 同为量化内核的性能与精度权衡类变更（不同后端），都强调在改变内部数据布局 / 计算方式后仍

保持数值语义，可作为本 PR 位精确设计思路的横向参照。

- PR #56773 [Bugfix][CPU] Fix DeepSeek-R1 (FP8 MLA + MoE) correctness on CPU backend: 同属 FP8 量化在不同后端 / 路径上的一致性议题；该 PR 修复 CPU 上 FP8 加载与 MLA 权重错乱，与本 PR 关注「同一数值语义在多条执行路径上必须一致」的问题域相邻。