

PR #3217 完整报告

radixark/miles

Reduce NVFP4 weight synchronization overhead

合并时间: 2026-09-14 15:25

原文链接: <http://prhub.com.cn/radixark/miles/pull/3217>

执行摘要

- 一句话: 消除 NVFP4 权重发布的主机同步与规则扫描开销
- 推荐动作: 值得精读, 尤其推荐给维护 NVFP4 RL、权重同步与 checkpoint 导出性能的工程师。重点关注四个设计决策: (1) 用 Python 标量 + clamp_max + torch.where 取代标量张量分配与 CUDA 标量真值判断, 这是消除隐式同步的通用手法, 前提是分子在 FP32 下精确可表示 (1536 / 2688), 换其它 e4m3_max 时需要重新确认; (2) _literal_ignore_rules 返回 None 作为“交给正则路径”的哨兵值, 虽然保留了顺序 re.match 与异常行为, 但把正确性押在调用约定上, 建议后续补一个断言或在 _is_ignored 中显式处理; (3) 用 TorchDispatchMode 把性能契约写成断言, 这是本 PR 最值得借鉴的测试范式; (4) 接受 (1,) 形状保持、并在 head 上删除策略回归测试, 属于明确声明的取舍, 建议后续补回 tests/fast/utils/test_nvfp4_ignore_policy.py 或等价用例, 并在合并 head 上重跑一次完整发布路径的墙钟对比以闭合性能证据链。

功能与动机

PR body 开篇写明目标: “Reduce NVFP4 weight-publication overhead by keeping global-scale conversion on device and avoiding repeated Python scans of checkpoint exclusion rules”。具体痛点有两个: (1) 旧的 `nvfp4_global_encode_scale_te` 用 `if global_encode_scale == torch.tensor(0.0, ...)` 在 Python 侧判断 CUDA 标量真值, PR body 记录这会 “synchronize once per expert quantization during weight publication”, 即每个专家量化都会触发一次 device→host 同步; (2) 一次 GLM-5.2 发布会对 1888 条字面量 ignored module 名字做 96000 次谓词匹配。作者同时强调前提是“Preserve the quantization math and weight-selection semantics”, 即优化不得改变位级结果。关联 issue 评论中维护者 yueming-yuan 只提出“please fix precommit”, 说明合入前唯一的硬性整改是格式问题。

实现拆解

1. 消除 CUDA 标量的同步读回 (miles/utils/nvfp4.py): `nvfp4_global_encode_scale_te` 原本先用 `torch.tensor(FP4_E2M1_MAX, device=...)`、`torch.tensor(float(nvfp4_e4m3_max), device=...)` 现造标量张量, 再用 `torch.min` 做溢出钳位, 最后在 `numel() == 1` 分支里对 CUDA 标量做 Python 真值判断来修复零尺度。改为 `torch.div(float(nvfp4_e4m3_max) * FP4_E2M1_MAX, global_amax.to(torch.float32)).clamp_max(torch.finfo(torch.float32).max)`, 零尺度修

复统一交给 `torch.where(scale == 0.0, 1.0, scale)`。收益有两层：标量张量不再被算子隐式 lift 到 device，也不再需要把 device 值读回主机；`nvfp4_global_decode_scale_te` 只是取倒数，因此自动继承该改进。作者在注释中给出可验证前提——两个受支持分子 1536（256×6）与 2688（448×6）都能被 FP32 精确表示，所以改用 Python 浮点不改变位级结果。

2. 消除重复的排除规则线性扫描（`miles/backends/megatron_utils/megatron_to_hf/process`
`ors/quantizer_nvfp4.py`）：新增 `@lru_cache(maxsize=16)` 装饰的 `_literal_ignore_rules(ignore_rules: tuple[str, ...])`，把一个转换批次内的策略快照成 `frozenset`；只要策略里含任何 `re`：规则就返回 `None`，交回原有顺序 `re.match` 路径。`_is_ignored` 增加 `literal_rules` 可选参数：非 `None` 时用 `name in literal_rules` 命中完整权重名，再用 `name.rpartition('.')` 逐级上溯到点分祖先，语义等价于旧的 `name == rule or name.startswith(f'{rule}.')`，但把 $O(\text{规则数} \times \text{权重名数})$ 降到 $O(\text{名字深度})$ 。`_quantize_moe_params` 在批次开头调用一次 `_literal_ignore_rules(tuple(ignore_rules))`，并把结果透传给两次 `should_quantize_param` 调用（`gate/up` 配对扫描与最终落盘循环），保证配对选择与输出顺序不变。
3. 把“不得读回标量”固化为可回归契约（`tests/fast-gpu/test_nvfp4_quantizer.py`）：新增 `_NoHostScalarRead(TorchDispatchMode)`，在 `__torch_dispatch__` 中拦截 `aten._local_scalar_dense.default`（`device→host` 标量读）与 `aten.lift_fresh.default`（现造主机标量张量）并对两者断言失败；新增 `test_nvfp4_global_scale_exact_without_host_scalar_read`，按 `device`（`cpu/cuda`） $\times$ `m3_max`（256/448） $\times$ `shape`（(), (1,）、(8,）、(2, 4)）展开 16 组用例，输入覆盖 0.0、3.5、`finfo.tiny`、`finfo.max`、`±inf`、`NaN`，期望值由 CPU 上冻结的旧批量算术推导，比较使用 `rtol=atol=0` 与 `equal_nan=True`。该测试同时断言 `encoded.shape == amax.shape`，把下面第 4 点的形状契约固定下来。
4. 形状契约的显式化（同一源文件 + 同一测试）：旧实现在 `numel() == 1` 且尺度为 0 时会用 `torch.tensor(1.0, device=...)` 替换，把 (1,) 塌缩成 ()；新的 `torch.where` 分支保持输入形状。PR body 说明生产量化路径传入的是标量 `amax`，因此该差异只影响批形状为单元素的情形，并用测试断言固化“形状保持”语义。
5. 集成配方调整（`tests/e2e/megatron/test_glm5_2_744b_a40b_5layer_nvfp4_w4a16.py`）：`ACTOR_NUM_GPUS / ROLLOUT_NUM_GPUS` 由 `NUM_GPUS`（8）改为 4 / 4，并移除 `--colocate`，使 W4A16 走与 W4A4 对齐的 4+4 分离式分配（TP4/EP4 训练 + 两个 TP2/DP2/EP2 推理引擎），用于在真实权重发布路径上度量优化效果；B64、rollout 次数、响应长度、fake QAT 与精度 / 后端开关均未改动。该改动同时改变了 CI 覆盖的拓扑形态。
6. 提交演进与校验记录：5 个提交依次为 `d0c6005a`（全局尺度去同步）、`d8e12d10`（尺度回归用例合并）、`a819ac5d`（字面量策略优化）、`313343d8`（删除策略回归测试）、`7f3697f4`（pre-commit 格式化）。作者记录的校验为：全局尺度相关 GPU 套件 1592 passed / 0 failed / 0 skipped in 53.45 s（单 c2 节点 8×B300，Python 3.12.3、PyTorch 2.13.0+cu130），集成运行中 W4A16 平均发布墙钟 3.9060→1.2606 s、W4A4 1.1443→0.8699 s；策略侧 11 条 CPU 回归用例（0.63 s）在 import commit 上验证后即被删除。

关键文件：

- miles/utils/nvfp4.py (模块 量化内核; 类别 source; 类型 core-logic; 符号 nvfp4_global_encode_scale_te, nvfp4_global_decode_scale_te) : 全局缩放编码 / 解码尺度是权重发布热路径的核心计算; 本 PR 在此去掉标量张量分配与 CUDA 标量真值判断, 直接消除逐专家 device→host 同步, 同时把零尺度修复统一为保持形状的 torch.where。
- miles/backends/megatron_utils/megatron_to_hf/processors/quantizer_nvfp4.py (模块 权重导出; 类别 source; 类型 core-logic; 符号 _literal_ignore_rules, _is_ignored, _should_quantize_param, _quantize_moe_params) : 引入 _literal_ignore_rules 缓存与祖先回溯匹配, 把 1888 条字面量排除规则的逐次线性扫描降为常数级哈希查询, 并显式保留 re: 正则策略的顺序 re.match 路径与异常行为。
- tests/fast-gpu/test_nvfp4_quantizer.py (模块 量化测试; 类别 test; 类型 test-coverage; 符号 _NoHostScalarRead, torch_dispatch, test_nvfp4_global_scale_exact_without_host_scalar_read) : 新增 _NoHostScalarRead (TorchDispatchMode) 守卫与 16 组零容忍用例, 把“尺度换算不得读回 device 标量、不得现造主机标量张量”从口头约定变成可回归断言, 同时固化形状保持契约。
- tests/e2e/megatron/test_glm5_2_744b_a40b_5layer_nvfp4_w4a16.py (模块 端到端配方; 类别 test; 类型 test-coverage; 符号 ACTOR_NUM_GPUS, ROLLOUT_NUM_GPUS) : W4A16 集成配方改为 4+4 分离式分配 (TP4/EP4 训练 + 两个 TP2/DP2/EP2 推理引擎) 并移除 --colocate, 用于在真实权重发布路径上度量优化效果, 同时改变了 CI 覆盖的拓扑组合。

关键符号: nvfp4_global_encode_scale_te, nvfp4_global_decode_scale_te, _literal_ignore_rules, _is_ignored, _should_quantize_param, _quantize_moe_params, _NoHostScalarRead.torch_dispatch, test_nvfp4_global_scale_exact_without_host_scalar_read

关键源码片段

miles/utils/nvfp4.py

全局缩放编码 / 解码尺度是权重发布热路径的核心计算; 本 PR 在此去掉标量张量分配与 CUDA 标量真值判断, 直接消除逐专家 device→host 同步, 同时把零尺度修复统一为保持形状的 torch.where。

```
def nvfp4_global_encode_scale_te(
    global_amax: torch.Tensor,
    nvfp4_e4m3_max: int = int(FP8_E4M3_MAX),
) -> torch.Tensor:
    # 两个受支持分子 1536 (256 * 6) 与 2688 (448 * 6) 都能被 FP32 精确表示,
    # 所以直接用 Python 标量参与运算不会改变位级结果;
    # 而 torch.tensor(..., device=...) 现造的标量张量会被算子隐式 lift / 拷贝,
    # 在高频调用 (每个专家量化一次) 下变成可观的固定开销。
    global_encode_scale = torch.div(
        float(nvfp4_e4m3_max) * FP4_E2M1_MAX,
        global_amax.to(torch.float32),
    ).clamp_max(torch.finfo(torch.float32).max) # 与旧的 torch.min(scale, FP32_MAX) 等价

    # 旧实现是 `if global_encode_scale == torch.tensor(0.0, ...)`:
```

```

# 对 CUDA 标量做 Python 真值判断会触发一次 device→host 同步,
# 这正是权重发布阶段的主要串行开销来源。
# 现在把 TE 的零尺度修复规则整体留在 device 上,
# 标量 amax 与批量 amax 走同一条分支, 不再有 numel() == 1 的特例。
# 附带修正: 旧的标量分支用 0 维张量替换, 会把形状 (1,) 塌缩成 ();
# 新实现保持输入形状, 该契约由下面测试文件中的 shape 断言固化。
return torch.where(global_encode_scale == 0.0, 1.0, global_encode_scale)

```

```

def nvfp4_global_decode_scale_te(
    global_amax: torch.Tensor,
    nvfp4_e4m3_max: int = int(FP8_E4M3_MAX),
) -> torch.Tensor:
    # 解码尺度只是编码尺度的倒数: 零尺度修复已在上游完成,
    # 这里不需要再判断零值, 因此也不会二次引入同步点;
    # 由于编码路径改为纯 device 计算, 解码路径自动继承全部收益。
    return torch.div(1.0, nvfp4_global_encode_scale_te(global_amax, nvfp4_e4m3_max))

```

miles/backends/megatron_utils/megatron_to_hf/processors/quantizer_nvfp4.py

引入 `_literal_ignore_rules` 缓存与祖先回溯匹配, 把 1888 条字面量排除规则的逐次线性扫描降为常数级哈希查询, 并显式保留 `re`: 正则策略的顺序 `re.match` 路径与异常行为。

```

@lru_cache(maxsize=16)
def _literal_ignore_rules(ignore_rules: tuple[str, ...]) -> frozenset[str] | None:
    # 每个转换批次只做一次策略快照: list 先变成可哈希的 tuple 作为缓存键,
    # 相同内容命中缓存; 配置内容变化后键不同, 策略自然失效。
    # 这里不缓存张量、不回写配置, 避免把副作用带进导出路径。
    # 只要策略里混入任何 `re:` 规则就返回 None, 交回原有的顺序 re.match 路径,
    # 从而保留正则的短路顺序、非法正则抛出的异常类型等全部既有行为。
    if any(rule.startswith("re:") for rule in ignore_rules):
        return None
    return frozenset(ignore_rules)

```

```

def _is_ignored(name: str, ignore_rules: list[str], literal_rules: frozenset[str] | None = None) -> bool:
    if literal_rules is not None:
        # 字面量规则是冻结集合, 命中查询是 O(1);
        # 沿 "." 逐级上溯祖先, 等价于旧实现里
        # `name == rule or name.startswith(f"{rule}.")` 的前缀语义,
        # 但把「1888 条规则 * 96000 次判定」压成「名字深度次哈希查询」。
        # 注意: 只要某条规则是 `re:`, _literal_ignore_rules 就会返回 None,
        # 因此这个分支不会静默跳过正则规则。
        while True:
            if name in literal_rules:
                return True
            name, separator, _ = name.rpartition(".")
            if not separator:

```

```

        return False
# 正则策略的兜底路径：顺序扫描，短路行为与改动前完全一致。
for rule in ignore_rules:
    if rule.startswith("re:"):
        if re.match(rule[3:], name):
            return True
        continue
    if name == rule or name.startswith(f"{rule}."):
        return True
return False

```

```

def _quantize_moe_params(converted_named_params, ignore_rules):
    # 每个转换批次构建并哈希一次策略，而不是每次权重判定都重建；
    # 下面两次 _should_quantize_param 调用（gate/up 配对扫描与最终落盘循环）
    # 共用同一个 literal_rules，保证配对选择、元数据与输出顺序都不变。
    literal_rules = _literal_ignore_rules(tuple(ignore_rules))
    gated_candidates = {}
    for converted_name, param in converted_named_params:
        base, role = _split_gated_pair_name(converted_name)
        if base is None or role is None:
            continue
        if _should_quantize_param(converted_name, param, ignore_rules, literal_rules):
            roles = gated_candidates.setdefault(base, {})
            if role in roles:
                raise ValueError(
                    f"NVFP4 requires a single complete gate/up pair per conversion batch; "
                    f"found duplicate {role} tensor for {base}."
                )
            roles[role] = (converted_name, param)
    # ... 后续共享 amax 的 nvfp4_quantize_1d_pair 配对量化与三件套输出保持不变
    return quantize_named_params

```

```

def _should_quantize_param(name, weight, ignore_rules, literal_rules=None):
    if ignore_rules and _is_ignored(name, ignore_rules, literal_rules):
        return False
    # 其余入口守卫（.weight 后缀、dtype、维度、NVFP4_GROUP_SIZE 整除）完全未改
    if not name.endswith(".weight"):
        return False
    if weight.dtype not in (torch.float16, torch.bfloat16, torch.float32):
        return False
    if weight.dim() < 2:
        return False
    if weight.shape[-1] % NVFP4_GROUP_SIZE != 0:
        raise ValueError(f>Last dim {weight.shape[-1]} must be divisible by {NVFP4_GROUP_SIZE}
        for NVFP4 ({name}).")
    return True

```

tests/fast-gpu/test_nvfp4_quantizer.py

新增 `_NoHostScalarRead` (`TorchDispatchMode`) 守卫与 16 组零容忍用例，把“尺度换算不得读回 device 标量、不得现造主机标量张量”从口头约定变成可回归断言，同时固化形状保持契约。

```
class _NoHostScalarRead(TorchDispatchMode):
    # 用 TorchDispatchMode 在算子层设卡：
    # 一旦尺度换算偷偷做了 device→host 标量读 (_local_scalar_dense) ,
    # 或者现造了一个主机标量张量 (lift_fresh) , 断言立刻失败。
    # 这样“消除隐式同步”这一性能目标就不再依赖人工审查,
    # 而成为 CI 可回归验证的行为契约。
    def __torch_dispatch__(self, func, types, args=(), kwargs=None):
        assert func != torch.ops.aten._local_scalar_dense.default, "Scale conversion read a device scalar"
        assert func != torch.ops.aten.lift_fresh.default, "Scale conversion created a host scalar tensor"
        return func(*args, **(kwargs or {}))

@pytest.mark.parametrize("device", ["cpu", "cuda"])
@pytest.mark.parametrize("e4m3_max", [256, 448])
@pytest.mark.parametrize("shape", [(), (1,), (8,), (2, 4)])
def test_nvfp4_global_scale_exact_without_host_scalar_read(device, e4m3_max, shape):
    if device == "cuda" and not torch.cuda.is_available():
        pytest.skip("CUDA is not available")
    # 16 组零容忍用例：0.0 触发零尺度修复分支，tiny/max 覆盖数值边界，
    # ±inf 触发溢出钳位，NaN 覆盖传播路径。
    values = torch.tensor(
        [
            0.0,
            1.0,
            3.5,
            torch.finfo(torch.float32).tiny,
            torch.finfo(torch.float32).max,
            float("inf"),
            -float("inf"),
            float("nan"),
        ],
        dtype=torch.float32,
    )
    # 期望值由 CPU 上冻结的旧批量算术推导，与被测的 device 实现解耦，
    # 保证断言的是“数学等价”，而不只是“自洽”。
    numerator = torch.tensor(float(e4m3_max), dtype=torch.float32) * torch.tensor(6.0, dtype=torch.float32)
    expected = torch.div(numerator, values)
    expected = torch.min(expected, torch.tensor(torch.finfo(torch.float32).max))
    expected = torch.where(expected == 0.0, torch.ones_like(expected), expected)

    case_size = torch.Size(shape).numel()
```

```

for cpu_amax, expected_encode in zip(values.split(case_size), expected.split(case_size),
strict=True):
    amax = cpu_amax.reshape(shape).to(device)
    expected_encode = expected_encode.reshape(shape)
    with _NoHostScalarRead(): # 区间内只允许 device 侧算子
        encoded = nvfp4_global_encode_scale_te(amax, e4m3_max)
        decoded = nvfp4_global_decode_scale_te(amax, e4m3_max)
    # 形状契约：旧的标量分支会把 (1,) 塌缩成 (), 新实现必须保持输入形状。
    assert encoded.device == amax.device and encoded.shape == amax.shape
    torch.testing.assert_close(encoded.cpu(), expected_encode, rtol=0, atol=0, equal_nan=
True)
    torch.testing.assert_close(decoded.cpu(), torch.div(1.0, expected_encode), rtol=0, atol=0,
equal_nan=True)

```

评论区精华

评审本身非常轻量：claude[bot] 说明“This pull request is from a fork — automated review is disabled”，没有机器评审；yueming-yuan 给出 APPROVED 且正文为空，Review 评论列表为空。真正有价值的信息来自作者在 PR body 中的自我披露：①11 条策略回归用例所在的 `tests/fast/utils/test_nvfp4_ignore_policy.py` 在独立提交 [313343d8](#) 中被删除，作者自己写明“Those 11 cases are historical validation, not tests present at the final head”；②证据边界被反复强调——“No new GPU run or complete combined-head weight-publication result is claimed”“the isolated policy measurements do not establish a full-publication speedup for the combined head”；③范围切分明确——#3228 的 literal-policy 优化被折进本 PR，#3227 的 scalar kernel fusion、打包广播传输、P2P 与 SGLang 接收端改动被排除在外。唯一的外部整改要求是 issue 评论“please fix precommit”，由末次提交 [7f3697f4](#) 收敛。

- 策略回归测试在最终 head 上被删除 (testing): 字面量策略优化（缓存键、祖先回溯、re: 分流）在合并 head 上没有任何自动化覆盖；作者以“历史验证 + 证据边界”的方式声明该缺口，评审未要求补回。
- 零尺度修复分支的形状契约变更 (correctness): 接受形状保持为新契约，标量路径行为不变；仅当外部代码依赖“修复后退化为 0 维”时才有影响。
- 性能收益的证据边界与跨提交拼接 (performance): 尺度去同步的收益有实测支撑，策略优化的端到端收益未在合并 head 上复测；读者应按“已验证分量 + 未验证合成”的口径理解结论。
- pre-commit 失败的整改要求 (style): 格式问题已收敛在末次提交，未改变任何语义；这是本 PR 唯一的外部整改项。
- W4A16 集成配方改为 4+4 分离式分配 (design): 配方与 #2855 的 W4A4 形态对齐，但 CI 因此不再覆盖 8 GPU colocate 下的 NVFP4 发布路径。
- fork PR 的自动化评审被禁用 (other): 该 PR 完全依赖维护者人工判断合入，没有机器人交叉验证，策略等价性等细节缺少第三方复核。

风险与影响

- 风险：

1. 策略路径在最终 head 上无回归测试：_literal_ignore_rules 的缓存键、frozenset 命中与 rpartition('.') 祖先回溯这三段逻辑，在 head 上没有任何自动化用例；等价性只能靠人工推理。若规则里出现以 . 结尾、含连续点或大小写敏感边角，旧 startswith('f{rule}.') 与新的祖先集合查询之间可能产生差异，而 CI 不会报警。
2. 隐式不变量耦合：_is_ignored 在 literal_rules is not None 时必定在 while True 内 return，永远不会落到下面的 re: 分支，其正确性完全依赖 _literal_ignore_rules 在发现任意 re: 规则时返回 None。若后续有人复用该函数或直接从别处构造 frozenset 传入，正则规则会被静默忽略（现有实现没有断言或类型约束来防止这一点）。
3. 形状契约变更：(1,) 不再塌缩为 ()。PR body 声称生产路径传标量 amax，但任何依赖“零尺度修复后退化为 0 维”的下游代码（例如对返回尺度做 reshape 或 item() 的导出 / 检查逻辑）都会受影响；测试只覆盖了函数级契约。
4. 性能结论依赖跨提交拼接：1592 用例与墙钟数据来自 global-scale 的中间提交与镜像，策略优化的收益未在合并 head 上单独端到端复测；lru_cache(maxsize=16) 若遇上超过 16 组不同规则（多角色 / 多模型连续导出）会退化为反复重建 frozenset。
5. 失败模式变化：若 quantization_config 的 ignore/exclude 中存在非字符串元素（如 dict 形式规则），frozenset(ignore_rules) 抛 TypeError，而旧实现是在 rule.startswith 处抛 AttributeError；两者都属于 fail-fast，但异常类型与信息对排障者不同。
6. CI 覆盖形态改变：W4A16 配方移除 --colocate 并改为 4+4 后，8 GPU colocate 形态下的 NVFP4 发布路径不再被该 e2e 测试覆盖。
7. 数值安全：clamp_max 与 torch.where 在 -0.0、±inf、NaN 上已由零容忍用例覆盖（NaN 做传播验证而非 payload 位一致性），未发现明显回归；但用例注册在 blackwell 硬件上，CPU 侧仅靠 torch.cuda.is_available() 跳过。- 影响：对训练任务：NVFP4 RL（尤其 GLM-5.2 的 W4A16/W4A4 配方）每个训练步的权重发布串行段变短，作者在单 c2 节点 8xB300 上记录 W4A16 平均发布墙钟 3.9060→1.2606 s、W4A4 1.1443→0.8699 s（每用例两步），大模型多专家的场景收益更明显。对系统：由于打包 FP4/FP8 字节与融合 QDQ 的 uint16 均要求精确相等、尺度用例零容忍，量化精度预期不受影响，改变的是开销与单元素形状契约。对团队：_NoHostScalarRead 这套 TorchDispatchMode 禁同步守卫、以及 lru_cache + frozenset + 祖先回溯的规则匹配模式都可复用到其它 checkpoint 导出与尺度计算路径。对发布与回滚：改动集中在两个源文件与两个测试文件，未触及传输层、P2P 与 SGLang 接收端，回滚成本低；但策略优化部分缺少 head 级测试，回滚判断只能依赖 PR body 中的历史证据。- 风险标记：核心量化路径变更，策略优化缺少最终 head 测试覆盖，零尺度分支形状契约变更，性能收益证据跨提交拼接，e2e 配方不再覆盖 colocate 形态

关联脉络

- PR #2855 Restrict NVFP4 RL to routed experts: 同一功能线：同样改动 miles/backends/megatron_utils/megatron_to_hf/processors/quantizer_nvfp4.py 与 GLM-5.2 W4A16 e2e 配方，确立了“共享专家保持高精度、仅量化路由专家”的权重选择语义，本 PR 明确要求不改变该语义。

- PR #2864 Add fused NVFP4 fake-QAT QDQ kernels: 同一 NVFP4/W4A16 技术栈的上游 PR, tests/fast-gpu/test_nvfp4_quantizer.py 中的 TE 导出与融合 QDQ 对比矩阵即由此建立; 本 PR 在保留这些既有用例的前提下追加 16 组尺度用例。
- PR #2701 feat(ci): declare supported GPU generations on every CUDA test: 本 PR 改动的两个测试文件都带有 hardware=["blackwell"] 注册声明, run-ci-blackwell-only 标签的调度能力依赖该机制, 因此本 PR 的验证可在对应硬件上被选择性触发。
- PR #3228 (材料未提供标题) NVFP4 字面量排除策略优化: PR body 说明该 PR 的 literal-policy 优化被直接折进本 PR (import commit a819ac5d, 生产代码与回归测试字节完全一致), 是本 PR 排除策略改动的来源。
- PR #3227 (材料未提供标题) NVFP4 scalar kernel fusion: PR body 明确将 scalar kernel fusion 排除在本 PR 之外, 与本 PR 的“去除 device 标量同步”目标相邻但拆分交付, 理解二者边界有助于评估后续优化空间。