

# PR #2864 完整报告

radixark/miles

Add fused NVFP4 fake-QAT QDQ kernels

合并时间: 2026-09-11 12:03

原文链接: <http://prhub.com.cn/radixark/miles/pull/2864>

## 执行摘要

- 一句话: 新增融合 NVFP4 fake-QAT QDQ 内核、Megatron 适配层与 W4A16 RL 端到端配方
- 推荐动作: 值得精读, 但建议按“契约 → 内核 → 测试”的顺序读: (1) 先读 NVFP4QDQConfig.\_\_post\_init\_\_ 与 current\_nvfp4\_qdq\_config(), 这是理解整个内核能力边界 (哪些开关合法、哪些组合被拒绝) 的最短路径; (2) 再看 miles/utils/nvfp4\_fake\_qat.py 的 env 门控 + 延迟 import 设计, 它决定了这项功能是否会影响未启用它的用户; (3) 最后读 tests/fast-gpu/test\_nvfp4\_quantizer.py 的 \_te\_qdq\_reference 与 bit-exact 断言, 理解“逐比特一致”具体对齐了什么 (含 +0/-0 的整数视图比较)。特别值得学习的设计决策有: 用编译期特化代替运行时分支、保留 TE 变量名但在文档串 / 注释 / 测试三处澄清语义、以及用 --no-gradient-accumulation-fusion 保证 STE 梯度回落到原始专家参数。同时请带着两个疑问阅读: TE 版本升级后 bit-exact 契约如何维护, 以及 Megatron 侧 hook 缺失时的静默失配如何被发现。

## 功能与动机

PR body 说明这是把 Megatron-LM #88 的单仓实现重新表达为「更干净的成对切分」: Miles 负责内核、FP32-amax/STE 适配、严格 Blackwell 测试与 E2E 配方, Megatron-LM #91 只加一个 env 门控的懒调用 (TEGroupedLinear.\_get\_weight\_tensors())。要解决的核心问题是旧 fake-QAT 走 quantize(...).dequantize(...) 两趟路径, 量化数据与 block scale 需要一次全局内存往返; 融合内核把这部分留在寄存器, 并把逐张量算术顺序对齐 TE 2.17 的 1D/1x16 契约 (参考 TE #2972、#3068, 向量化加载 /FP4 转换 /4over6 结构改编自 FlashInfer #3448)。严格 oracle 来自 TE 的 test\_nvfp4\_quantize\_exact.py, 四类数据族镜像 FlashInfer 的 test\_fp4\_quantize.py。

## 实现拆解

1. 内核与数值契约 (miles/utils/fused\_nvfp4\_qdq.py, 新增): 实现 rank-2 BF16/FP16 权重的融合 QDQ, 寄存器内保留 E4M3 block scale 与打包 E2M1 值, 只写回反量化结果。NVFP4QDQConfig 是 frozen dataclass, \_\_post\_init\_\_ 提前拒绝非法组合 (e4m3 max 非 256/448、非 4over6 却用 256、非 4over6 却开 error\_use\_fast\_math)。dtype、standard/4over6、E4M3 max、误差度量、exact/FP16 候选误差都做成编译期 CuTe 特化并缓存, 网格步长内核在运行时不分支。

2. 环境变量到配置的映射（同文件 `_env_flag / _env_applies_to_weights / current_nvfp4_qdq_config`）：把 TE 的 `NVTE_NVFP4_4OVER6`（`none/weights/activations/all`）、`NVTE_NVFP4_4OVER6_E4M3_USE_256`、`NVTE_NVFP4_4OVER6_ERR_MODE`、`NVTE_NVFP4_4OVER6_ERR_USE_FAST_MATH` 翻译成 `NVFP4QDQConfig`；`NVTE_USE_FAST_MATH=1`、布尔值非 0/1、以及 `inputs/gradients` 这类旧作用域都直接抛 `ValueError`，即 fail-fast 而不是静默降级。
3. Megatron 适配层（`miles/utils/nvfp4_fake_qat.py`，新增 26 行）：  
`maybe_fake_quantize_nvfp4_weight_tensors` 先用 `OPEN_TRAINING_NVFP4_FAKE_QAT_FLAG` 门控，未开启时原样返回并完全不 import CuTe DSL；开启时对每个离散权重套 `fake_nvfp4_quantization_ste`，由 STE 把梯度透传给原始专家参数。这样未启用路径不引入 cutlass 依赖，也不改变现有训练数值。
4. 严格数值验证（`tests/fast-gpu/test_nvfp4_quantizer.py`，+380/-2）：新增 `_make_te_qdq_quantizer / _te_qdq_reference` 构造 TE 原生 `NVFP4Quantizer` 参考（关掉 RHT、2D 量化、随机舍入、random sign mask），`test_fused_nvfp4_qdq_is_bit_exact_with_te` 用 uint16 整数视图做逐比特比较（区分 +0/-0），覆盖 bf16/fp16 × 12 种 shape × random/boundary/zeros/maxes 四类数据 × 标准 NVFP4 与 8 组 4over6 配置；另有 `STE/main_grad` 保留测试，以及 env 契约矩阵与非法值拒绝测试（`test_current_nvfp4_qdq_config_rejects_stale_te_scopes` 等）。
5. 端到端配方与 CI 注册（`tests/e2e/megatron/test_glm5_2_744b_a40b_5layer_nvfp4_w4a16.py`，新增 287 行）：复制原 NVFP4 配方为 W4A16 版本，`NVFP4_ENV` 固定 `NVTE_USE_FAST_MATH=0`、`NVTE_NVFP4_4OVER6=all`、E4M3 max 448、MSE + FP16 候选误差；`execute()` 里 actor 用 TF8/EP8 colocate 加四个 2 卡 SGLang 引擎（`--sglang-moe-runner-backend flashinfer_cutedsl`、`--sglang-moe-a2a-backend none`），并用 BF16 的 TE recipe 只让路由专家走 fake QDQ，共享专家经 `--extra-high-precision-layers-{hf,megatron}` 保持 BF16；`--no-gradient-accumulation-fusion` 让 STE 梯度落到原始专家参数。诊断上启用 `--sglang-enable-fp32-lm-head`、`--moe-router-use-torch-mm`，关闭 `--ci-disable-kl-checker / logprob / weight-update checker`，改用显式 rollout/train/trajectory dump，避开 SGLang 不支持的 NVFP4 权重校验和。

测试与 CI 配套：E2E 以 `register_cuda_ci(est_time=3600, suite="stage-c-8-gpu-b200", labels=["megatron","model-scripts"], hardware=["blackwell"])` 注册，PR 打上 `run-ci-blackwell-only`；维护者在合并前补了 `fix(ci): declare Blackwell for W4A16 test` 与 `fix(ci): include W4A16 in Megatron runs` 两个提交；原先的手工基准 `tests/manual/benchmark_fused_nvfp4_qdq.py` 在 E2E 通过后由单独提交（142e904）删除，只在 PR body 里保留历史 revision 链接与原始结果。

关键文件：

- `miles/utils/fused_nvfp4_qdq.py`（模块 量化内核；类别 source；类型 core-logic；符号 `NVFP4QDQErrorMode`，`NVFP4QDQConfig`，`current_nvfp4_qdq_config`，`fused_nvfp4_qdq`）：本 PR 的核心：新增 1266 行 CuTe DSL 融合 NVFP4 QDQ 内核，外加编译期配置、TE 环境变量解析、amax 计算与 STE 包装，决定整套 fake-QAT 的数值契约。

- `miles/utils/nvfp4_fake_qat.py` (模块 假量化适配; 类别 `source`; 类型 `core-logic`; 符号 `NVFP4_FAKE_QAT_FLAG`, `maybe_fake_quantize_nvfp4_weight_tensors`) : Megatron 侧唯一入口: `env` 门控 + 函数内延迟 `import`, 保证未启用该路径的进程既不改变数值也不依赖 CuTe DSL, 是“默认零影响”的关键。
- `tests/fast-gpu/test_nvfp4_quantizer.py` (模块 精度测试; 类别 `test`; 类型 `test-coverage`; 符号 `_select_local_cuda_device`, `_make_qdq_input`, `_make_te_qdq_quantizer`, `_te_qdq_reference`) : 定义数值正确性的硬门槛: 与 TE 原生 `NVFP4Quantizer` 的逐比特一致、STE 梯度与 `main_grad` 保留、以及完整的 TE 环境变量契约矩阵, 是本 PR 唯一可自动化的正确性证据。
- `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 `prepare`, `execute`, `_validate_glm_checkpoint`, `_extra_high_precision_layers_hf_args`) : 端到端配方: `colocate` GLM-5.2 5 层 NVFP4 W4A16 GRPO, 把 fake QDQ 训练与 CuTe DSL NVFP4 推理串起来, 并按 Blackwell 注册进 CI。

关键符号: `fused_nvfp4_qdq`, `fake_nvfp4_quantization_ste`, `compute_nvfp4_amax`, `current_nvfp4_qdq_config`, `maybe_fake_quantize_nvfp4_weight_tensors`, `_env_applies_to_weights`, `_te_qdq_reference`, `test_fused_nvfp4_qdq_is_bit_exact_with_te`, `execute`

## 关键源码片段

### `miles/utils/fused_nvfp4_qdq.py`

本 PR 的核心: 新增 1266 行 CuTe DSL 融合 NVFP4 QDQ 内核, 外加编译期配置、TE 环境变量解析、`amax` 计算与 STE 包装, 决定整套 fake-QAT 的数值契约。

# `miles/utils/fused_nvfp4_qdq.py` (节选: 数值契约与环境变量映射, 均为完整实现单元)

```
class NVFP4QDQErrorMode(IntEnum):
    """Four Over Six candidate error metric."""

    MAE = 0
    MSE = 1

@dataclass(frozen=True)
class NVFP4QDQConfig:
    # 所有字段都是编译期开关: CuTe 特化按它们分桶缓存, 热路径不做运行时分叉。
    use_4over6: bool = False
    e4m3_max: int = 448
    error_mode: NVFP4QDQErrorMode = NVFP4QDQErrorMode.MAE
    # 沿用 TE 的公开变量名; 在 TE 2.17 契约下它表示 FP16 舍入的候选误差计算,
    # 而不是普通的指令级 fast math。
    error_use_fast_math: bool = False

    def __post_init__(self) -> None:
        if self.e4m3_max not in (256, 448):
```

```

        raise ValueError(f"NVFP4 E4M3 max must be 256 or 448, got {self.e4m3_max}.")
# E4M3 max 256 只在 4over6 路径上有定义, 非法组合在这里就被挡住。
if not self.use_4over6 and self.e4m3_max != 448:
    raise ValueError("E4M3 max 256 is only supported by Four Over Six.")
if not self.use_4over6 and self.error_use_fast_math:
    raise ValueError("The FP16 error contract only applies to Four Over Six.")

def _env_flag(name: str, default: str = "0") -> bool:
    value = os.getenv(name, default).strip()
    if value == "1":
        return True
    # 空串按 0 处理, 但 "true"/"yes" 这类拼写直接报错, 避免被静默当成关闭。
    if value in ("0", ""):
        return False
    raise ValueError(f"{name} must be 0 or 1, got {value!r}.")

def _env_applies_to_weights(name: str, default: str) -> bool:
    # 作用域四态枚举, 只有 weights / all 才算命中权重侧。
    value = os.getenv(name, default).strip().lower()
    if value not in ("none", "weights", "activations", "all"):
        raise ValueError(f"{name} must be one of none, weights, activations, or all; got {value!r}.")
    return value in ("weights", "all")

def current_nvfp4_qdq_config() -> NVFP4QDQConfig:
    """Resolve the weight QDQ contract from Transformer Engine environment variables."""
    # 普通量化 fast math 超出本内核的数值契约, 直接 fail-fast。
    if _env_flag("NVTE_USE_FAST_MATH"):
        raise ValueError(
            "Fused NVFP4 QDQ requires NVTE_USE_FAST_MATH=0; ordinary quant fast math "
            "is outside its numerical contract."
        )

    use_4over6 = _env_applies_to_weights("NVTE_NVFP4_4OVER6", "none")
    use_e4m3_256 = _env_applies_to_weights("NVTE_NVFP4_4OVER6_E4M3_USE_256", "all")
    error_mode_name = os.getenv("NVTE_NVFP4_4OVER6_ERR_MODE", "MAE").strip().upper()
    try:
        error_mode = NVFP4QDQErrorMode[error_mode_name]
    except KeyError as exc:
        raise ValueError(f"NVTE_NVFP4_4OVER6_ERR_MODE must be MAE or MSE, got {error_mode_name!r}.") from exc

    error_use_fast_math = _env_flag("NVTE_NVFP4_4OVER6_ERR_USE_FAST_MATH")
    return NVFP4QDQConfig(
        use_4over6=use_4over6,
        # 只有权重侧启用 4over6 且 E4M3 256 作用域命中时, max 才收紧到 256。
        e4m3_max=256 if use_4over6 and use_e4m3_256 else 448,

```

```

        error_mode=error_mode,
        # 尽管变量名带 legacy 色彩，这里选择的是 TE 的 FP16 候选误差契约。
        error_use_fast_math=(use_4over6 and error_use_fast_math),
    )

```

```

@dsl_user_op
def _load_v4_u32(base_ptr: Int64, *, loc=None, ip=None) -> tuple[UInt32, UInt32, UInt32,
    UInt32]:
    """一次 16B 向量化载入：量化与反量化全程留在寄存器，不走全局内存往返。"""
    result = llvm.inline_asm(
        llvm.StructType.get_literal([T.i32(), T.i32(), T.i32(), T.i32()]),
        [Int64(base_ptr).ir_value(loc=loc, ip=ip)],
        "ld.global.v4.u32 {$0, $1, $2, $3}, [$4];",
        "=r,=r,=r,=r,l",
        has_side_effects=False,
        is_align_stack=False,
        asm_dialect=llvm.AsmDialect.AD_ATT,
        loc=loc,
        ip=ip,
    )
    return (
        UInt32(llvm.extractvalue(T.i32(), result, [0], loc=loc, ip=ip)),
        UInt32(llvm.extractvalue(T.i32(), result, [1], loc=loc, ip=ip)),
        UInt32(llvm.extractvalue(T.i32(), result, [2], loc=loc, ip=ip)),
        UInt32(llvm.extractvalue(T.i32(), result, [3], loc=loc, ip=ip)),
    )

```

### miles/utils/nvfp4\_fake\_qat.py

Megatron 侧唯一入口：env 门控 + 函数内延迟 import，保证未启用该路径的进程既不改变数值也不依赖 CuTe DSL，是“默认零影响”的关键。

```

# miles/utils/nvfp4_fake_qat.py (完整实现)
"""Megatron-facing adapter for fused NVFP4 fake QAT."""

from __future__ import annotations

import os

import torch

# 单一开关：默认关闭时整条新增路径完全不生效，也不 import CuTe DSL。
NVFP4_FAKE_QAT_FLAG = "OPEN_TRAINING_NVFP4_FAKE_QAT_FLAG"

def maybe_fake_quantize_nvfp4_weight_tensors(
    weight_tensors: list[torch.Tensor],
) -> list[torch.Tensor]:
    """Apply env-gated fused NVFP4 fake QAT to discrete TE grouped-linear weights."""

```

```

if os.getenv(NVFP4_FAKE_QAT_FLAG, "0") != "1":
    # 未开启时原样返回：不引入 cutlass 依赖，也不改变现有训练数值。
    return weight_tensors

# 延迟 import，保证不使用该路径的进程不需要 CuTe DSL。
from miles.utils.fused_nvfp4_qdq import current_nvfp4_qdq_config, fake_nvfp4_quantization_
ste

# 每次调用都从 TE 环境变量重新解析契约：非法配置在这里 fail-fast。
qdq_config = current_nvfp4_qdq_config()
# STE：前向走 QDQ，反向直接把梯度透传给原始专家参数。
return [fake_nvfp4_quantization_ste(weight, qdq_config) for weight in weight_tensors]

```

```
__all__ = ["NVFP4_FAKE_QAT_FLAG", "maybe_fake_quantize_nvfp4_weight_tensors"]
```

### tests/fast-gpu/test\_nvfp4\_quantizer.py

定义数值正确性的硬门槛：与 TE 原生 NVFP4Quantizer 的逐比特一致、STE 梯度与 main\_grad 保留、以及完整的 TE 环境变量契约矩阵，是本 PR 唯一可自动化的正确性证据。

# tests/fast-gpu/test\_nvfp4\_quantizer.py（节选：TE 参考路径与逐比特断言）

```

def _make_te_qdq_quantizer(config: NVFP4QDQConfig):
    # 逐项关掉 RHT / 2D 量化 / 随机舍入 / random sign mask，只保留本 PR 支持的最小契约，
    # 让参考实现与 fused kernel 处在同一数值前提下，否则 bit-exact 比较没有意义。
    return te.NVFP4Quantizer(
        rowwise=True,
        columnwise=False,
        with_amax_reduction=False,
        with_rht=False,
        with_post_rht_amax=False,
        with_2d_quantization=False,
        stochastic_rounding=False,
        row_scaled_nvfp4=False,
        nvfp4_use_4over6=config.use_4over6,
        nvfp4_e4m3_max=config.e4m3_max,
        nvfp4_4over6_err_mode=config.error_mode.name,
        with_random_sign_mask=False,
    )

```

```

def _te_qdq_reference(x: torch.Tensor, config: NVFP4QDQConfig) -> tuple[torch.Tensor, torch.
Tensor]:
    m, n = x.shape
    # TE 的 BF16 分派按 M 维 16 行对齐，这里补零对齐后只比对有效区域。
    padded_m = ((m + 15) // 16) * 16
    if padded_m == m:
        x_padded = x.contiguous()
    else:

```

```

padding = torch.zeros((padded_m - m, n), dtype=x.dtype, device=x.device)
x_padded = torch.cat((x.contiguous(), padding), dim=0)

quantized = _make_te_qdq_quantizer(config).quantize(x_padded)
reference = quantized.dequantize(dtype=x.dtype)[:m, :n].contiguous()
assert quantized._amax_rowwise is not None
return reference, quantized._amax_rowwise.reshape(1)

@torch.inference_mode()
def test_fused_nvfp4_qdq_is_bit_exact_with_te(monkeypatch, dtype, shape, init_data, config) ->
None:
    """Cover BF16/FP16 x shapes x data patterns x the full supported feature matrix."""
    monkeypatch.setenv("NVTE_USE_FAST_MATH", "0")
    monkeypatch.setenv("NVTE_NVFP4_4OVER6_ERR_USE_FAST_MATH", "1" if config.error_use_
fast_math else "0")
    x = _make_qdq_input(shape, dtype, init_data)
    amax = compute_nvfp4_amax(x)
    expected, te_amax = _te_qdq_reference(x, config)
    actual = fused_nvfp4_qdq(x, amax, config)

    # amax 决定全局 scale, 它必须先逐比特一致, 否则整张权重的误差都会被放大。
    assert torch.equal(amax.reshape(1).view(torch.int32), te_amax.view(torch.int32))
    # 整数视图区分 +0 / -0: rtol=atol=0 的浮点比较区分不出符号位。
    actual_bits = actual.view(torch.uint16)
    expected_bits = expected.view(torch.uint16)
    assert torch.equal(
        actual_bits, expected_bits
    ), f"bit mismatch count: {torch.count_nonzero(actual_bits != expected_bits).item()}"
    torch.testing.assert_close(actual, expected, rtol=0.0, atol=0.0)

```

## 评论区精华

- 本 PR 没有任何行内 review 评论 (review\_comments\_count=0)。正式审核只有 yueming-yuan 的一条 APPROVED, 且未附正文; claude[bot] 明确说明「This pull request is from a fork — automated review is disabled」, 因此 1266 行 CUDA 内核没有逐行审查记录, 数值正确性实际由 PR body 里的 C1 复现证据 (严格测试在 2xB200 上通过、pytest -q -o addopts= tests/fast-gpu/test\_nvfp4\_quantizer.py 全绿) 和测试矩阵承担。
- 讨论重心因此落在 PR body 的设计声明与 commit 上: \*\*Ownership split:\*\*Miles owns the kernel, FP32-amax/STE adapter, strict Blackwell tests, and E2E recipe, \*\*Deliberate non-goals:\*\*stochastic rounding, RHT, 2D quantization, transpose/columnwise output, row scaling, ordinary quantization fast math, pre-TE-2.17 compatibility, 即主动声明能力边界而不是留待 review 猜测。
- 合并前维护者的三个提交 (fix(ci): declare Blackwell for W4A16 test、fix(ci): include W4A16 in Megatron runs、style: format NVFP4 adapter imports) 显示 CI 元数据是在合并阶段才补齐的, 与 #3184 那类“遗漏 hardware 声明阻塞 PR CI”的问题同源。

- `error_use_fast_math` 这个字段名与 TE 变量名一一对应，但其 TE 2.17 语义是“FP16 舍入的候选误差计算”而非普通指令级 fast math；代码在 `NVFP4QDQConfig` 文档串和 `current_nvfp4_qdq_config()` 返回处都写了澄清注释，避免读者误解。
- Fork PR 跳过自动审查，最终只有一次无正文的 APPROVED (other): 以人工审批合并；1266 行 CUDA 内核没有任何逐行审查记录，数值正确性完全依赖 PR body 中的 C1 复现证据与测试矩阵，读者需要自行复核关键算术。
- 合并前补齐 CI 元数据：Blackwell 硬件声明与 Megatron 运行集 (testing): 最终以 `register_cuda_ci(est_time=3600, suite="stage-c-8-gpu-b200", labels=["megatron", "model-scripts"], hardware=["blackwell"])` 注册，与 #2701 引入的硬件感知调度机制对齐。
- 手工 benchmark 的删除与性能证据的去留 (performance): 仓库内不再保留性能基线与复现脚本，融合内核的实际收益只能通过历史 revision 链接与原始结果回溯，后续做性能回归时要重新搭基准。
- `error_use_fast_math` 命名与 TE 2.17 语义的刻意对齐 (design): 为保持与 TE 环境变量一一对应而保留易误解的名字，并在文档串、源码注释、`test_current_nvfp4_qdq_config_maps_full_latest_te_env_contract` 三处显式固定语义。

## 风险与影响

- 风险：
  1. 无独立审查的高复杂度内核：miles/utils/fused\_nvfp4\_qdq.py 新增 1266 行内联 PTX / CuTe DSL，包含手写 `ld.global.v4.u32`、`mul.f32x2`、`add.rn.f32` 等算术顺序敏感的指令。fork PR 跳过自动审查且无行内评论，一旦某条指令的舍入模式与 TE 漂移，只会表现为训练数值缓慢偏移，回归测试之外缺少人工防线。
  2. 与 TE 版本的强耦合：bit-exact 结论只在 TE 2.17 + CuTe DSL 4.6.2 + SM100 上验证 (PR body 明确硬件 / 镜像 / 依赖版本，并声明 no pre-2.17 version fallback or runtime support validator)。TE 后续改动候选误差或 4over6 实现时，`test_fused_nvfp4_qdq_is_bit_exact_with_te` 会直接失败，而生产路径的数值则会静默改变。
  3. 环境契约解析是硬失败路径：`current_nvfp4_qdq_config()` 对 `NVTE_USE_FAST_MATH=1`、`NVTE_NVFP4_4OVER6=inputs|gradients`、`"true"` 这类布尔拼写都抛 `ValueError`；任何沿用旧 TE 环境变量脚本的用户会在训练启动期直接报错。它是 fail-fast (比静默错误好)，但缺迁移说明。另外该函数在每次权重 QDQ 调用都会重新读环境变量并构造 config (`maybe_fake_quantize_nvfp4_weight_tensors` 内调用)，存在可忽略但可优化的 per-call 开销。
  4. 跨仓库静默失配：Miles 侧只提供适配层，真正调用点在 Megatron-LM #91 的 `TEGroupedLinear._get_weight_tensors()`。如果 Megatron 侧没打上该 hook，设置 `OPEN_TRAINING_NVFP4_FAKE_QAT_FLAG=1` 不会有任何效果也不会告警，容易让使用者误以为 QAT 已生效。
  5. E2E 检出力被主动削弱：`--ci-disable-logprobs-checker`  
`--ci-disable-weight-update-checker` `--ci-disable-kl-checker` 加上关闭 MLA signed-KL 断言，PR body 自己也写明“This is integration coverage, not actor/rollout

numerical parity”，因此这条 E2E 只能证明链路可跑通，不能证明 actor/rollout 数值一致。

6. CI 成本：est\_time=3600 且限定 Blackwell (8xB200 节点)，是 nightly/ 定期回归级别的资源占用；同时 tests/manual/benchmark\_fused\_nvfp4\_qdq.py 被删除后，仓库内不再有可复现的性能基线，只有历史链接。

- 影响：

- 对用户：默认零影响——未设置 OPEN\_TRAINING\_NVFP4\_FAKE\_QAT\_FLAG=1 时 maybe\_fake\_quantize\_nvfp4\_weight\_tensors 直接早退，且通过函数内延迟 import 保证不引入 CuTe DSL/cutlass 依赖。启用后仅作用于 TE 2.17+ 的离散 rank-2 专家权重，共享专家保持 BF16 (EXTRA\_HIGH\_PRECISION\_LAYERS\_HF/MEGATRON)，首末层不做排除。
- 对系统：训练侧新增一条“假量化”数值路径（前向 QDQ、反向 STE），与 rollout 侧的 CuTe DSL NVFP4 W4A16 推理形成“训练 BF16 + 权重假量化 / 服务 NVFP4”组合；融合内核减少显存流量，可能改善大模型 QAT 的内存峰值，但首次运行需承担 CuTe DSL 编译开销与缓存管理。
- 对团队 / 流程：确立了一种跨仓库协作范式——内核与测试留在 Miles，Megatron 只保留最小 hook，并靠外部 PR 编号 (Megatron-LM #91/#88/#75) 串联。后续任何 TE 升级或 Megatron hook 变动都需要两边同时核对。
- 对 CI：新增一条 3600s、仅 Blackwell 的 E2E，进入 stage-c-8-gpu-b200，直接抬高 nightly 的机器占用；同时它沿用 #2701 的 hardware=["blackwell"] 声明机制，属于该硬件感知调度脉络的延续。
- 风险标记：Fork PR 无自动化审查，1266 行 CUDA 内核缺独立复核，依赖 TE 2.17 环境契约（无 fallback），跨仓库静默失配 (Megatron-LM #91 hook)，E2E 关闭 KL/logprob/ 权重更新校验，CI 成本高 (8xB200 / 3600s)

## 关联脉络

- PR #2855 Restrict NVFP4 RL to routed experts: 同一 NVFP4 RL 配方线的直接前置：本 PR 的 E2E 沿用“共享专家保持 BF16”的结论，并在 PR body 中直接引用 #2855 作为 --ci-disable-kl-checker 跳过单边 MLA signed-KL 断言的先例。
- PR #2701 feat(ci): declare supported GPU generations on every CUDA test: 本 PR 的 E2E 使用的 register\_cuda\_ci(..., hardware=["blackwell"]) 正是 #2701 引入的硬件感知调度机制；合并前维护者专门补了 Blackwell 声明提交。
- PR #3184 ci: declare hardware on the two CUDA tests that landed after #2701: 与本次合并前 fix(ci): declare Blackwell for W4A16 test 属于同一类问题：新增 CUDA 测试漏声明 hardware 会阻塞 PR CI，说明该机制仍在被逐步补齐。