

PR #35605 完整报告

sgl-project/sglang

[XPU] Use torch scaled_mm for XPU block FP8 linear

合并时间: 2026-09-16 10:10

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/35605>

执行摘要

- 一句话: XPU 块状 FP8 线性改用 torch scaled_mm, 并预排布 scale 存储
- 推荐动作: 值得精读, 但重点不在算子本身, 而在三处工程决策: 一是派发层级的选择 (把平台分支放进 _dispatch_auto_backend 而非顶层, 保证显式 backend 仍可覆盖), 这是容易被写错且直接破坏用户配置的细节; 二是用公开 API torch.nn.functional.scaled_mm + ScalingType 显式表达 activation 与 weight 两套 block scale 配方, 替换私有 torch._scaled_mm 关键字参数写法, 可读性与前向兼容性都更好; 三是把 scale 的 transpose-contiguous 物化提前到 process_weights_after_loading_block_quant, 属于典型的「热路径代价前移」优化。建议后续补充: XPU 上 Triton vs scaled_mm 的延迟对比数据、在派发处增加 torch 版本 / 能力兜底, 以及确认除 w8a8_block_fp8_linear 之外没有其他读取 weight_scale_inv 的消费方依赖行连续布局。

功能与动机

PR body 明确写出动机: "Enable the native Torch scaled_mm path for block-wise FP8 linear on Intel XPU. The existing XPU path relies on the Triton block-FP8 implementation which is slower than scaled_mm." 即在 Intel XPU 上块状 FP8 (DeepSeek 类 [128, 128] 权重配方) 线性层目前走 Triton 实现, 性能不如 torch 原生 scaled_mm。除替换 kernel 外, body 还补充了第二条动机: 先在 process_weights_after_loading_block_quant 阶段一次性物化 transpose-contiguous 的 block scale 存储, "avoids repeated forward-time contiguous/copy_ operations inside torch scaled_mm", 把布局代价从热路径挪到加载期。

实现拆解

1. 新增核心算子实现 (python/sglang/srt/layers/quantization/fp8_utils.py, 新增 torch_w8a8_block_fp8_linear, +56 行)。函数先做两层参数校验: block_size 必须是长度为 2 的 list/tuple, 且 block_k 必须为 128、block_n 只能取 1 或 128 (等价于显式拒绝 MXFP8 的 [1, 32]、[64, 64] 等组合)。随后根据 block_n 选择权重侧配方 (BlockWise1x128 / BlockWise128x128), 把 input reshape 成 2D, input_scale 为 None 时调用 per_token_group_quant_fp8 现场量化、否则直接复用预量化结果; 对 q_input 与 weight 只在 stride(-1) != 1 时才 contiguous, 最后调用 torch.nn.functional.scaled_mm (activation 恒为 BlockWise1x128), 输出按原始 batch 维度 view 回去。这一步决定了 XPU 上块状 FP8 的实际数值行为与性能。

2. 接入派发链（同文件 `dispatch_w8a8_block_fp8_linear / _dispatch_auto_backend`）。关键改动是插入位置：最初版本把 `if _is_xpu: return torch_w8a8_block_fp8_linear` 放在 `dispatch_w8a8_block_fp8_linear` 函数体顶端，会让 `--fp8-gemm-backend=triton` 失效；最终改为在 `_dispatch_auto_backend()` 的 `npu_w8a8_mxfp8_linear` 分支之后、`triton_w8a8_block_fp8_linear` 兜底之前插入 `elif _is_xpu`，并同步更新函数内注释的优先级列表（新增 "6. XPU (Intel GPU, PyTorch torch._scaled_mm)"，Triton 降为 7）。这样显式 backend 选择依旧优先，自动模式下 XPU 才落到新后端，其他平台分支顺序完全不变。
3. 权重加载期布局改造（`python/sglang/srt/layers/quantization/fp8.py`, `process_weights_after_loading_block_quant`, +24 行）。新增守卫条件 `is_xpu()` 且 `self.w8a8_block_fp8_linear is torch_w8a8_block_fp8_linear` 且 `self.weight_block_size in ([1, 128], [128, 128])` 且 `layer.weight_scale_inv.ndim == 2`；命中后用 `torch.empty_strided(scale.shape, (1, scale.shape[0]))` 构造转置连续 buffer、`copy_` 内容后通过 `layer.weight_scale_inv.set_` 替换底层 storage。因为 `[128, 128]` 路径 `wrapper` 会传 `weight_scale.t()`、`[1, 128]` 路径 `scaled_mm` 内部会转置 `scale_b`，两种配方都因此免掉前向中的连续化拷贝。守卫里用 `dispatch` 结果做判断意味着用户显式选 `triton` 时不会误改布局。
4. 测试配套（`test/registered/e2e/xpu/test_xpu_fp8_linear.py`, 新增 336 行）。文件用 `register_xpu_ci(est_time=20, suite="stage-b-test-1-gpu-xpu")` 注册；提供 `reference_block_fp8_matmul` 反量化参考实现，并覆盖 `test_w8a8_block_fp8_dispatch_on_xpu`（断言自动模式确实返回 `torch_w8a8_block_fp8_linear` 而非 Triton）、形状 / 维度（`M=1/8/17`、`2D/3D`、非连续 `3D`）、预量化输入、非连续 strided 视图、非法 `block_size` 拒绝、非方形 `weight_scale`（`N != K`）、数值精度、`[1, 128]` 配方与 load 后布局等价（`rtol=0, atol=0`）。
5. 范围收敛与配置配套。commit "just keep block fp8 changes" 把早期一并改动的 `use_rowwise_torch_scaled_mm` XPU 分支（`return torch_release >= (2, 13)`）从 PR 中剥离，使改动只聚焦 block FP8；`fp8.py` 的 `import` 列表相应新增 `torch_w8a8_block_fp8_linear`。改动只影响 XPU，未触及任何配置项 schema 或部署脚本。

关键文件：

- `python/sglang/srt/layers/quantization/fp8_utils.py`（模块 线性层调度；类别 source；类型 core-logic；符号 `torch_w8a8_block_fp8_linear`, `dispatch_w8a8_block_fp8_linear`, `_dispatch_auto_backend`）：新增 `torch_w8a8_block_fp8_linear` 算子实现，并在 `_dispatch_auto_backend` 中为 XPU 插入派发分支；是本次性能改造的核心与全平台共用热文件。
- `python/sglang/srt/layers/quantization/fp8.py`（模块 权重量化；类别 source；类型 core-logic；符号 `process_weights_after_loading_block_quant`）：在 `process_weights_after_loading_block_quant` 中新增 XPU 专属的 `weight_scale_inv` 转置连续存储物化，是消除前向 contiguous/copy 的关键配套改动。
- `test/registered/e2e/xpu/test_xpu_fp8_linear.py`（模块 线性层测试；类别 test；类型 test-coverage；符号 `reference_block_fp8_matmul`, `TestXPUFP8Linear`, `setUp`, `test_w8a8_block_fp8_dispatch_on_xpu`）：新增 336 行 XPU 专项测试，覆盖派发路由、形状 / 维度、预量化、非连续视图、非法 `block_size` 与 load 后布局等价，是本 PR 的验证

基线。

关键符号: torch_w8a8_block_fp8_linear, dispatch_w8a8_block_fp8_linear, _dispatch_auto_backend, process_weights_after_loading_block_quant, reference_block_fp8_matmul, test_w8a8_block_fp8_dispatch_on_xpu, test_torch_w8a8_block_fp8_linear_numerical_accuracy

关键源码片段

python/sglang/srt/layers/quantization/fp8_utils.py

新增 torch_w8a8_block_fp8_linear 算子实现, 并在 _dispatch_auto_backend 中为 XPU 插入派发分支; 是本次性能改造的核心与全平台共用热文件。

```
# python/sglang/srt/layers/quantization/fp8_utils.py
# XPU 上块状 FP8 线性的新实现: 用 torch 原生 scaled_mm 取代 Triton block-FP8 kernel。
def torch_w8a8_block_fp8_linear(
    input: torch.Tensor,
    weight: torch.Tensor,
    block_size: List[int],
    weight_scale: torch.Tensor,
    input_scale: Optional[torch.Tensor] = None,
    bias: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    """Run block-FP8 linear with Torch's scaled_mm implementation."""
    # block_size 必须是二元定义, 否则后面解包 block_n, block_k 会静默出错
    if not isinstance(block_size, (list, tuple)) or len(block_size) != 2:
        raise ValueError(
            f"XPU block-FP8 scaled_mm expects a two-dimensional weight_block_size, "
            f"but got {block_size}"
        )
    block_n, block_k = block_size
    # block_k 必须为 128 (MXFP8 的 [1, 32] 在此被显式拒绝), block_n 只放开 1 与 128
    if block_k != 128 or block_n not in (1, 128):
        raise ValueError(
            "XPU block-FP8 scaled_mm supports weight_block_size [1, 128] or "
            f"[128, 128], but got {block_size}"
        )

    # 权重侧配方随 block_n 切换: [1, 128] 用 BlockWise1x128, [128, 128] 用 BlockWise128x128
    scale_b_recipe = (
        torch.nn.functional.ScalingType.BlockWise1x128
        if block_n == 1
        else torch.nn.functional.ScalingType.BlockWise128x128
    )

    # scaled_mm 只处理 2D 乘积, 先把任意前缀维度展平, 最后再 view 回去
    input_2d = input.reshape(-1, input.shape[-1])
    if input_scale is None:
        # 未预量化: 现场做 per-token-group 量化; activation 恒为 1 x block_k 配方
        q_input, activation_scale = per_token_group_quant_fp8(input_2d, block_k)
```

```

else:
    q_input = input_2d
    activation_scale = input_scale.reshape(-1, input_scale.shape[-1])

# 用 stride(-1) 判断而不是 is_contiguous(), 已连续的张量不会被多拷一次
if q_input.stride(-1) != 1:
    q_input = q_input.contiguous()
if weight.stride(-1) != 1:
    weight = weight.contiguous()
weight_t = weight.t() # 转置视图, 零拷贝
# [128, 128] 时这里显式转置 scale_b; [1, 128] 时由 scaled_mm 内部完成转置,
# 配合加载期预排布的转置连续存储, 两条路径都不再产生前向拷贝
scale_b = weight_scale if block_n == 1 else weight_scale.t()
output = torch.nn.functional.scaled_mm(
    q_input,
    weight_t,
    activation_scale,
    torch.nn.functional.ScalingType.BlockWise1x128,
    scale_b,
    scale_b_recipe,
    bias=bias,
    # 预量化输入的 input 已是 fp8, 必须显式指定输出 dtype
    output_dtype=torch.bfloat16 if input_scale is not None else input.dtype,
)
# 还原调用方的原始 batch 维度
return output.view(*input.shape[:-1], weight.shape[0])

# python/sglang/srt/layers/quantization/fp8_utils.py
# 自动派发: XPU 分支被刻意放在 NPU 之后、Triton 兜底之前。
# 函数开头先处理 --fp8-gemm-backend 的显式选择, 因此用户显式指定 triton 时
# 根本不会走到这里, XPU 默认值与用户覆盖意图不会互相打架。
def _dispatch_auto_backend() -> Callable:
    """Auto-select the best backend based on hardware capabilities."""
    # Priority order for auto selection:
    # 1. DeepGEMM (if enabled and available)
    # 2. FlashInfer TRTLLM (if Blackwell GPU and FlashInfer available)
    # 3. CUTLASS (if SM120 GPU and CUDA 12.8+)
    # 4. AITER (if AMD GPU with AITER enabled)
    # 5. NPU (Ascend)
    # 6. XPU (Intel GPU, PyTorch torch._scaled_mm)
    # 7. Triton (fallback)

    if deep_gemm_wrapper.ENABLE_JIT_DEEPGEMM:
        return deepgemm_w8a8_block_fp8_linear_with_fallback
    elif get_platform().is_blackwell and is_flashinfer_available():
        return flashinfer_gemm_w8a8_block_fp8_linear_with_fallback
    elif get_platform().is_sm120:
        return cutlass_w8a8_block_fp8_linear_with_fallback
    elif _use_aiter:

```

```

        return aiter_w8a8_block_fp8_linear
    elif is_npu_arch35():
        from sglang.srt.hardware_backend.npu.quantization.linear_method_npu import (
            npu_w8a8_mxfp8_linear,
        )

        return npu_w8a8_mxfp8_linear
    elif _is_xpu:
        # 新增分支: Intel GPU 默认走 torch scaled_mm, 不再落到 Triton
        return torch_w8a8_block_fp8_linear
    else:
        return triton_w8a8_block_fp8_linear

```

python/sglang/srt/layers/quantization/fp8.py

在 `process_weights_after_loading_block_quant` 中新增 XPU 专属的 `weight_scale_inv` 转置连续存储物化，是消除前向 `contiguous/copy` 的关键配套改动。

```

# python/sglang/srt/layers/quantization/fp8.py
# 位于 Fp8LinearMethod.process_weights_after_loading_block_quant 的尾部,
# 紧跟 AITER preshuffle 分支之后, 只对 XPU + torch scaled_mm 后端生效。
if (
    is_xpu()
    and self.w8a8_block_fp8_linear is torch_w8a8_block_fp8_linear
    and self.weight_block_size in ([1, 128], [128, 128])
    and layer.weight_scale_inv.ndim == 2
):
    # 保持 checkpoint 的逻辑形状 [N-blocks, K-blocks] 不变, 只把存储改成转置连续
    # (stride = (1, N)) : [1, 128] 下 scaled_mm 会内部转置 scale_b,
    # [128, 128] 下 wrapper 传入的是 scale_b.t(), 两种配方都因此省掉每次前向的
    # contiguous/copy_。因为 w8a8_block_fp8_linear 由 dispatch 决定, 用户显式指定
    # --fp8-gemm-backend=triton 时此分支不会命中, 权重语义保持不变。
    scale = layer.weight_scale_inv.data
    scale_b_is_contiguous = scale.t().is_contiguous()
    if not scale_b_is_contiguous:
        scale_reordered = torch.empty_strided(
            scale.shape,
            (1, scale.shape[0]), # 列优先存储, 等价于 transpose-contiguous
            dtype=scale.dtype,
            device=scale.device,
        )
        scale_reordered.copy_(scale)
        # set_ 直接替换底层 storage: 形状、dtype、数值都不变, 只有 stride 变化
        with torch.no_grad():
            layer.weight_scale_inv.set_(scale_reordered)

```

test/registered/e2e/xpu/test_xpu_fp8_linear.py

新增 336 行 XPU 专项测试，覆盖派发路由、形状 / 维度、预量化、非连续视图、非法 `block_size` 与 `load` 后布局等价，是本 PR 的验证基线。

```

# test/registered/e2e/xpu/test_xpu_fp8_linear.py
register_xpu_ci(est_time=20, suite="stage-b-test-1-gpu-xpu")

class TestXPUFP8Linear(CustomTestCase):
    def setUp(self):
        # 没有 XPU 的环境直接 skip, 避免 CPU-only CI 上误报失败
        if not torch.xpu.is_available():
            self.skipTest("XPU is not available")
        self.device = "xpu"

    def test_w8a8_block_fp8_dispatch_on_xpu(self):
        """Verify dispatch_w8a8_block_fp8_linear routes to torch_w8a8_block_fp8_linear on XPU.
        """
        dispatched_fn = dispatch_w8a8_block_fp8_linear()
        # 断言自动模式确实落到新的 scaled_mm 后端, 而不是 Triton 兜底;
        # 这条断言同时锁住了 _dispatch_auto_backend 中 XPU 分支的位置
        self.assertIs(dispatched_fn, torch_w8a8_block_fp8_linear)

    def test_torch_w8a8_block_fp8_linear_numerical_accuracy(self):
        """Compare torch_w8a8_block_fp8_linear with dequantized baseline on XPU."""
        M, K, N = 16, 256, 256
        block_size = [128, 128]

        x = torch.randn(M, K, dtype=torch.bfloat16, device=self.device)
        weight = torch.randn(N, K, dtype=torch.bfloat16, device=self.device).to(
            torch.float8_e4m3fn
        )
        weight_scale = (
            torch.rand(N // 128, K // 128, dtype=torch.float32, device=self.device)
            + 0.1
        )
        bias = torch.randn(N, dtype=torch.bfloat16, device=self.device)

        out_torch = torch_w8a8_block_fp8_linear(
            x, weight, block_size, weight_scale, bias=bias
        )
        # 手工构造 load 之后的转置连续 layout, 模拟 fp8.py 里 set_ 的存储形态
        weight_scale_post_load = torch.empty_strided(
            weight_scale.shape,
            (1, weight_scale.shape[0]),
            dtype=weight_scale.dtype,
            device=weight_scale.device,
        )
        weight_scale_post_load.copy_(weight_scale)
        out_post_load = torch_w8a8_block_fp8_linear(
            x, weight, block_size, weight_scale_post_load, bias=bias
        )
        # rtol=0/atol=0: 布局改写必须做到逐元素完全一致, 不允许任何数值漂移

```

```

torch.testing.assert_close(out_post_load, out_torch, rtol=0, atol=0)

# 再与反量化参考实现比对，确认算子本身的数值正确性
q_input, input_scale = per_token_group_quant_fp8(x, block_size[1])
out_ref = reference_block_fp8_matmul(
    q_input, input_scale, weight, block_size, weight_scale, bias=bias
).to(torch.bfloat16)

torch.testing.assert_close(out_torch, out_ref, rtol=0.05, atol=0.1)

```

评论区精华

评审共 5 条实质评论，主线是「派发顺序」「用什么 API」「block scale 语义」三件事。① rahulvijayaraghavan 在 CHANGES_REQUESTED 中指出 "To support --fp8-gemm-backend=triton, check need to be below line 573", 作者随后把 XPU 判断从 `dispatch_w8a8_block_fp8_linear` 顶部移入 `_dispatch_auto_backend()`，并回复 "dispatch for torch_w8a8_block_fp8_linear is moved to _dispatch_auto_backend()". ② 同一 reviewer 建议改用 PyTorch 公开文档 API，作者回复 "Updated."，最终实现由最初的 `torch._scaled_mm(scale_a=..., scale_b=..., out_dtype=...)` 改为 `torch.nn.functional.scaled_mm` 位置参数 + `ScalingType` 配方，并顺手删掉了 `if isinstance(output, tuple)` 分支（对应评论 "Check may not be required, since out is always Tensor"）。③ polisettyvarma 追问 "can we use 128x128 directly?", 作者解释 activation 侧始终沿用 SGLang 的 $1 \times \text{block_k}$ per-token-group 约定（scale 形状 $[M, K/128]$ ），`BlockWise128x128` 只作用于转置后的 weight operand，MXFP8 另走 1×32 且已被 `block_k` 校验挡掉。④ polisettyvarma 两次问及 BMG 支持性 ("what happens on unsupported devices like bmg?", "will this work on bmg?")，作者确认支持并以 BMG e2e 结果作答。⑤ mingfeima 在合并前补充判断：`block_n == 1` 意味着 `weight_block_size=[1, 128]`，"In SGLang that is not what current checkpoints use"——常规 /DeepSeek 类为 $[128, 128]$ ，MXFP8 为 $[1, 32]$ 已被拒绝，`channel-wise FP8` 走 `apply_fp8_linear`，因此该分支只是 torch 侧的防御性 guard，可以保留不改。

- 派发位置：XPU 分支不能挡在显式 backend 之前 (design): 作者将 XPU 判断移入 `_dispatch_auto_backend()`，位于 npu 分支之后、triton 兜底之前，并回复 "dispatch for torch_w8a8_block_fp8_linear is moved to _dispatch_auto_backend()"; 测试用 `assertIs` 锁死自动模式的返回值。
- 使用私有 `torch._scaled_mm` 还是公开 API `torch.nn.functional.scaled_mm` (design): 改为公开 API `torch.nn.functional.scaled_mm` + `ScalingType` 位置参数（activation 用 `BlockWise1x128`、weight 用 `BlockWise1x128/128x128`），并删除 `tuple` 分支；作者两次回复 "Updated."。
- activation 侧 block scale 能否直接用 128×128 (performance): 保持 activation 固定 `BlockWise1x128`、weight 侧随 `block_n` 切换的双配方结构，未做改动。
- `block_n == 1` 分支在实际 checkpoint 中是否可达 (design): "So leaving `block_n = 1` is a guard from torch, but won't be called from sglang. you can keep this PR unchanged."——保留该分支作为 torch 侧防御性校验，不额外删除，也不影响主路径。

- BMG 等设备对新后端的支持性 (question): 作者回复 "It is supported on BMG." 并以 Intel BMG 上的 e2e 结果作为依据, 未增加设备白名单或能力探测。
- torch 版本门槛与 use_rowwise_torch_scaled_mm 的 XPU 分支 (question): 作者确认是 13, 同时表示 torch 升级后该判断可移除; 随后 commit "just keep block fp8 changes" 把 rowwise 改动整体剥离出本 PR, 最终改动只保留 block FP8, 但也因此没有为新后端留下任何 torch 版本兜底。

风险与影响

- 风险: 1) 缺少 torch 版本兜底: `_dispatch_auto_backend()` 在 XPU 上无条件返回 `torch_w8a8_block_fp8_linear`, 而原 `use_rowwise_torch_scaled_mm` 中的 `torch_release >= (2, 13)` XPU 门控已随 "just keep block fp8 changes" 一并移除。若运行环境 torch 过旧、不支持带 `ScalingType` 的 `scaled_mm`, 错误会发生在模型前向阶段而不是优雅回退到 Triton, 建议在派发处补版本 / 能力检查 (材料未给出 SGLang XPU 侧 torch 版本下限, 存在不确定性)。2) 加载期改写参数存储: `fp8.py` 用 `layer.weight_scale_inv.set_(scale_reordered)` 把 stride 从行连续改成 (1, N), 打破了「`weight_scale_inv` 连续」的隐含假设; 虽然守卫把范围限制在 XPU + torch 后端 + 两种 block 尺寸 + 2D scale, 但任何绕过 `self.w8a8_block_fp8_linear` 直接读取该参数的消费方 (自定义 kernel、权重回写 / 再量化) 都需重新确认。3) 输出 dtype 语义被硬编码: `output_dtype=torch.bfloat16 if input_scale is not None else input.dtype`, 预量化分支会强制 bf16 输出, 未来若有 fp16 调用方会静默改 dtype。4) 收益缺少数据支撑: PR body 的 "Speed Tests and Profiling" 一栏为空, "Triton 更慢" 这一前提没有被 benchmark 量化, 合并后无法直接判断加速幅度与适用 shape。5) 数值容差偏宽: 多处测试使用 `rtol=0.05/atol=0.1` (对 FP8 常见), 仅 load 后布局等价用 `rtol=0/atol=0` 做严格断言。6) CI 状态: body 快照中 PR Test (Base)、PR Test (Extra)、AMD ROCm 10 三个 workflow 均为失败, 期间两次 /rerun-failed-ci; 最终由 polisettyvarma、rahulvijayaraghavan、mingfeima 三人 approve 后合并, 失败项应属无关 / 不稳定作业。7) 回归面: `dispatch` 的新增分支位于 NPU 之后, 理论上不影响 CUDA/AMD/NPU 路径, 但 `fp8_utils.py` 是被全平台共用的热文件, 任何后续编辑都需保持该分支顺序。
- 影响: 对用户: Intel XPU (BMG/Arc Pro 等) 上运行块状 FP8 量化模型 (DeepSeek 类 [128, 128] 权重配方) 时, 默认块状 FP8 线性从前向中少掉一次 scale 连续化拷贝, 预期降低 decode 阶段小 M 场景的线性层开销, 且无需改任何启动参数; 显式指定 `--fp8-gemm-backend=triton` 的用户行为不变。对系统: 影响面被 `_is_xpu` 与 `is_xpu()` 双重隔离, CUDA/ROCm/NPU 的派发结果、权重处理与数值行为完全不变; 影响集中在 XPU 的 FP8 量化线性子系统。对团队: XPU 侧 block FP8 GEMM 从自研 Triton kernel 转向 torch 原生 kernel, 减少对 Triton 调优与 tuning config 的依赖; 新增的 336 行测试为后续 XPU 量化改动提供基线保护, 并把「load 后布局改写不得引入数值漂移」这一契约固化成断言。
- 风险标记: 核心量化路径变更, 新后端缺少 torch 版本兜底, 权重加载期改写 stride 布局, 收益缺少 benchmark 数据, 输出 dtype 被硬编码为 bf16, CI 合并前为红色

关联脉络

- PR #32618 [CPU] Add fp8_per_tensor_scaled_mm_cpu kernel: 同属「非 CUDA 平台补齐 scaled_mm 类 FP8 GEMM 能力」的功能线：在 fp8.py 量化路径中接入 CPU 专用 FP8 kernel 并新增注册测试，与本 PR 在 fp8_utils.py / fp8.py 接入 XPU scaled_mm 的模式高度一致，可作为横向对照。
- PR #28723 [Intel XPU] Enable fused_moe_triton tuning on XPU and add tuned DeepSeek-OCR-2 configs: 同一 Intel XPU 平台方向的性能工作，且同样涉及 XPU 上 Triton GEMM/MoE 调优与 fp8/DeepSeek 类工作负载配置，本 PR 正是把 XPU 的 block FP8 线性从 Triton 切到 torch 原生 kernel，两者构成 XPU GEMM 后端演进的连续脉络。
- PR #39613 Accept MXFP8 dispatch in FlashInfer A2A TRT-LLM MoE: 都在处理 FP8 量化派发的入口守卫：该 PR 放宽 MoE 入口以接受 MXFP8 dispatch，本 PR 则在 torch_w8a8_block_fp8_linear 中显式拒绝 MXFP8 的 [1, 32]，并借 block_k 校验区分两种量化配方，可对照阅读量化格式与后端的匹配逻辑。