

PR #56985 完整报告

vllm-project/vllm

[Bugfix][CPU] Add per-tensor FP8 W8A16 kernel to fix Ministral crash on CPU

合并时间: 2026-09-16 15:49

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56985>

执行摘要

- 一句话: 新增 CPU 逐张量 FP8 W8A16 内核, 修复 Ministral 崩溃
- 推荐动作: 值得精读, 尤其是从事 CPU 推理与量化内核的工程师。重点关注三处设计决策: (1) 用 `expand(1, num_k_blocks)` 合成单块 `scale` 来复用分块 AMX 内核的「零算子改动」思路, 这是把 `block-scaled` 算子泛化到 `per-tensor` 场景的可借鉴模式; (2) `process_weights_after_loading` 中 $[K, N] \rightarrow [N, K]$ 的权重朝向修复, 它揭示了 `Fp8LinearMethod` 与 CPU AMX 算子之间的隐形数据契约, 任何新增 CPU FP8 内核都必须处理; (3) `can_implement` 主动拒绝 $N \% 32 \neq 0$ 的防御式设计, 把内核崩溃转化为一次干净的内核回退。阅读时建议同时确认: 新增内核在 CPU 分派表中的优先级是否会影响既有非 AMX 路径、以及测试文件模块级 AMX 跳过是否与 CI 的 CPU 机型矩阵相容。

功能与动机

PR body 明确指出: `PerTensorTorchFP8ScaledMMLinearKernel` 是当时唯一可用的 `per-tensor` FP8 CPU 内核, 且没有为支持 AMX 的 CPU 正确接线, 导致加载 fp8 量化的 `PixtralForConditionalGeneration` (例如 `Ministral-3-8B-Instruct-2512`) 时直接崩溃。作者在验证内核修复期间排查 CPU CI 反复 flaky (Buildkite #89028、#89045), 发现多模态 fixture 每次都要重新联网拉取、以及部分权重缓存的 `offline`→`online` 重试从未触发, 于是把这两个 CI 问题一并修掉。改 MTP smoke 模型则是因为 `Qwen/Qwen3.5-0.8B-Base` 的 `video_preprocessor_config.json` 会把 `size.longest_edge` 放大到 chat checkpoint 的约 9.3 倍, 使 warmup 走 114,688 token 的视频编码预算而非 16,384 token 的图像预算, 在 CPU 上同步跑视觉编码器, 造成约 800s 的初始化时间。

实现拆解

1. 新增 CPU `per-tensor` FP8 W8A16 内核: 在 `vllm/model_executor/kernels/linear/scaled_mm/cpu.py` 中引入 `_BLOCK_SIZE_K = 128` 常量 (对应 `csrc/cpu/sgl-kernels/gemm.h` 中 AMX 内核固定 K 分块) 和 `CPUFp8PerTensorScaledMMLinearKernel(FP8ScaledMMLinearKernel)`, 文件顶部新增对 `FP8ScaledMMLinearKernel` 的导入。`is_supported` 设三道门槛: CPU 平台、`torch.cpu._is_amx_tile_supported()` (Sapphire Rapids 及更新)、`ops._supports_cpu_fp8_w8a16`。`can_implement` 只接受 `per-tensor` 的权重与激活 `scale`、`bf16/fp32` 输出, 并显式拒绝 $N \% 32 \neq 0$, 避免 AMX `tinygemm` 的 N 向 32 分块出现碎片块后崩掉进程。

2. 权重与 scale 的契约转换: `process_weights_after_loading` 把 `Fp8LinearMethod` 为 `torch._scaled_mm` 约定准备好的 $[K, N]$ 权重转置回 $[N, K]$ 并 `contiguous()`, 再走 `torch.ops._C.convert_weight_packed` 做 VNNI 预打包; 随后把标量 per-tensor scale 通过 `reshape().expand(1, num_k_blocks)` 合成为形状 $(1, \text{ceil}(K/128))$ 的假 block scale, 使 `fp8_scaled_mm_cpu` 可以原样复用, 激活侧不做 FP8 量化 (W8A16)。
`apply_weights` 以 `[self._block_size_n, _BLOCK_SIZE_K]` 作为 block shape 调算子, `apply_scaled_mm` 与既有 `CPUFp8BlockScaledMMKernel` 保持一致直接抛 `NotImplementedError`, 形成统一的内核实现约定。
3. 分派表注册与导出: `vllm/model_executor/kernels/linear/__init__.py` 把新内核插到 `_POSSIBLE_FP8_KERNELS[PlatformEnum.CPU]` 列表首位, 优先于 `PerTensorTorchFP8ScaledMMLinearKernel` 与 `ChannelWiseTorchFP8ScaledMMLinearKernel`, 从而改变 AMX CPU 上所有 per-tensor FP8 层的选择结果; `vllm/model_executor/kernels/linear/scaled_mm/__init__.py` 同步在导入与 `__all__` 中导出该类, 保持包级 API 一致。
4. 测试配套: `tests/kernels/quantization/test_cpu_fp8_scaled_mm.py` 新增 `quantize_weight_per_tensor_fp8`、`ref_fp8_per_tensor_scaled_mm` 两个参考实现, `test_cpu_fp8_per_tensor_scaled_mm_kernel` 走完整 kernel 类路径 (参数化 $M \in \{1, 64\}$ 、 $N, K \in \{(32, 64), (5120, 5120)\}$ 、有无 bias), 并用 `layer.weight` 预置为 $[K, N]$ 的方式专门覆盖权重转置修复; `test_cpu_fp8_per_tensor_kernel_rejects_non_multiple_of_32_n` 验证 $N \in \{16, 48\}$ 被 `can_implement` 拒绝。文件顶部新增模块级 `torch.cpu._is_amx_tile_supported()` 跳过守卫。PR body 记录该文件 80 passed。
5. CI 修复: `.buildkite/scripts/hardware_ci/run-cpu-test.sh` 将 `OFFLINE_RETRY_PATTERN` 扩展出 `Cannot find any model weights with`, 以匹配 `default_loader.py` 在快照目录只有 config、没有权重文件时抛出的包裹型 `RuntimeError`; 同时给 `docker run` 增加 `-v ~/.cache/vllm:/root/.cache/vllm`, 让落在 `VLLM_ASSETS_CACHE` (`~/.cache/vllm/assets`) 的 `VideoAsset/ImageAsset/AudioAsset` fixture 在容器生命周期结束后仍然保留。
6. e2e 测试模型替换: `tests/v1/e2e/test_cpu_spec_decode.py` 把 MTP smoke 的 `Qwen/Qwen3.5-0.8B-Base` 改为 `Qwen/Qwen3.5-0.8B`, 绕开 -Base 的 video preprocessor 配置触发的超大视频 warmup, 属于测试可运行性调整, 不改变断言门槛 `MIN_ACCEPTANCE_LEN = 1.5`。

关键文件:

- `vllm/model_executor/kernels/linear/scaled_mm/cpu.py` (模块 量化内核; 类别 source; 类型 core-logic; 符号 `CPUFp8PerTensorScaledMMLinearKernel`, `is_supported`, `can_implement`, `process_weights_after_loading`): 新增 `CPUFp8PerTensorScaledMMLinearKernel` 是本次修复的核心: 把 per-tensor FP8 权重映射到既有 AMX 分块量化算子, 并顺带修正 $[K, N] \rightarrow [N, K]$ 的权重朝向契约与 N 非 32 倍数的拒绝逻辑。
- `tests/kernels/quantization/test_cpu_fp8_scaled_mm.py` (模块 量化测试; 类别 test; 类型 test-coverage; 符号 `quantize_weight_per_tensor_fp8`, `ref_fp8_per_tensor_scaled_mm`, `test_cpu_fp8_per_tensor_scaled_mm_kernel`,

test_cpu_fp8_per_tensor_kernel_rejects_non_multiple_of_32_n) : 新增 per-tensorFP8 参考实现与端到端内核测试，专门覆盖权重朝向修复路径和 N 非 32 倍数的拒绝路径；同时新增模块级 AMX 跳过守卫。

- vllm/model_executor/kernels/linear/__init__.py (模块 内核注册；类别 source；类型 data-contract；符号 _POSSIBLE_FP8_KERNELS) : 把新内核注册进 _POSSIBLE_FP8_KERNELS[PlatformEnum.CPU] 的首位，决定 AMX CPU 上 per-tensor FP8 的分派结果，是本次行为变更真正生效的位置。
- .buildkite/scripts/hardware_ci/run-cpu-test.sh (模块 构建脚本；类别 infra；类型 infrastructure；符号 OFFLINE_RETRY_PATTERN, run_test) : CPU CI 的两处修复：扩充 OFFLINE_RETRY_PATTERN 以匹配 default_loader.py 的包裹型 RuntimeError，并挂载 ~/.cache/vllm 以持久化多模态 fixture。
- vllm/model_executor/kernels/linear/scaled_mm/__init__.py (模块 内核导出；类别 source；类型 data-contract) : 在包级导入与 __all__ 中导出 CPUFp8PerTensorScaledMMLinearKernel，保证分派表和外部引用可解析。
- tests/v1/e2e/test_cpu_spec_decode.py (模块 推测解码；类别 test；类型 test-coverage；符号 SMOKE_CONFIGS) : 把 MTP smoke 测试的 checkpoint 从 Qwen3.5-0.8B-Base 换为非 Base 版本，规避 -Base 的 video preprocessor 配置导致的超大视频 warmup。

关键符号：CPUFp8PerTensorScaledMMLinearKernel.is_supported,
CPUFp8PerTensorScaledMMLinearKernel.can_implement,
CPUFp8PerTensorScaledMMLinearKernel.process_weights_after_loading,
CPUFp8PerTensorScaledMMLinearKernel.apply_weights,
CPUFp8PerTensorScaledMMLinearKernel.apply_scaled_mm,
quantize_weight_per_tensor_fp8, ref_fp8_per_tensor_scaled_mm,
test_cpu_fp8_per_tensor_scaled_mm_kernel, test_cpu_fp8_per_tensor_kernel_rejects_non_multiple_of_32_n, run_test

关键源码片段

vllm/model_executor/kernels/linear/scaled_mm/cpu.py

新增 CPUFp8PerTensorScaledMMLinearKernel 是本次修复的核心：把 per-tensor FP8 权重映射到既有 AMX 分块量化算子，并顺带修正 $[K, N] \rightarrow [N, K]$ 的权重朝向契约与 N 非 32 倍数的拒绝逻辑。

```
# csrc/cpu/sgl-kernels/gemm.h 中 BLOCK_K 的取值：AMX 内核固定的 K 向分块大小
_BLOCK_SIZE_K = 128
```

```
class CPUFp8PerTensorScaledMMLinearKernel(FP8ScaledMMLinearKernel):
    """基于 AMX BRGEMM 的 CPU FP8 W8A16 per-tensor GEMM。
```

```
    实现思路：复用分块量化的 AMX 内核 fp8_scaled_mm_cpu，把标量 per-tensor
    scale 伪装成“横跨整个权重”的单个块，因此激活值始终是 BF16/FP32，
    不做 FP8 激活量化——这正是它与 PerTensorTorchFP8ScaledMMLinearKernel 的关键差异。
    """
```

```

@classmethod
def is_supported(cls, compute_capability: int | None = None) -> tuple[bool, str | None]:
    # 三道门槛: CPU 平台、AMX tile 指令集 (Sapphire Rapids 及更新)、
    # 以及 libtorch stable 里是否编入了 fp8_scaled_mm_cpu 扩展算子
    if not current_platform.is_cpu():
        return False, "requires CPU platform."
    if not torch.cpu._is_amx_tile_supported():
        return False, "requires AMX tile support (Sapphire Rapids or newer)."
    if not ops._supports_cpu_fp8_w8a16:
        return False, "fp8_scaled_mm_cpu op not available."
    return True, None

@classmethod
def can_implement(cls, c: FP8ScaledMMLinearLayerConfig) -> tuple[bool, str | None]:
    # 只认 per-tensor 的权重 scale 与激活 scale, 输出仅支持 bf16/fp32
    if not c.weight_quant_key.scale.group_shape.is_per_tensor():
        return False, "requires a per-tensor weight scale."
    if not c.activation_quant_key.scale.group_shape.is_per_tensor():
        return False, "requires a per-tensor activation scale."
    if c.out_dtype not in (torch.bfloat16, torch.float32):
        return False, "Only bfloat16/float32 output dtype supported."
    n = c.weight_shape[0]
    if n % 32 != 0:
        # AMX tinygemm 以 32 为单位切 N 维, 尾部不足 32 的碎片块无法处理;
        # 在这里提前拒绝, 避免把一次崩进程的事故留到运行时才爆发
        return False, f"requires weight output dim (N={n}) to be a multiple of 32."
    return True, None

def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
    w_name, w_s_name, _, _ = self.layer_param_names
    # Fp8LinearMethod 会把权重转置成 (K, N) (torch._scaled_mm 的约定),
    # 而 convert_weight_packed 与 fp8_scaled_mm_cpu 期望 (N, K),
    # 也就是 CUDA nn.Linear 风格 checkpoint 的原始布局, 所以先转回来。
    weight = getattr(layer, w_name).t().contiguous()
    n, k = weight.shape

    # VNNI 预打包, 后续 apply_weights 以 is_vnni=True 调用算子
    packed_weight = torch.ops._C.convert_weight_packed(weight)
    replace_parameter(layer, w_name, torch.nn.Parameter(packed_weight, requires_grad=False))

    # 把标量 per-tensor scale 合成“单块”block scale, 形状 (1, num_k_blocks),
    # 这样 fp8_scaled_mm_cpu 无需任何改动即可作为 W8A16 路径复用
    weight_scale = getattr(layer, w_s_name)
    self._block_size_n = -(n // 32) * 32 # 向上取整到 32 的倍数
    num_k_blocks = -(k // _BLOCK_SIZE_K)
    block_scale = weight_scale.reshape().expand(1, num_k_blocks).contiguous()
    replace_parameter(layer, w_s_name, torch.nn.Parameter(block_scale, requires_grad=False))

```

```

def apply_weights(self, layer, x: torch.Tensor, bias: torch.Tensor | None = None) -> torch.
Tensor:
    w_name, w_s_name, _, _ = self.layer_param_names
    weight = getattr(layer, w_name)
    weight_scale = getattr(layer, w_s_name)

    # 3D 及以上输入先展平成 2D 做 GEMM, 算完再恢复原始 batch 维度
    x_2d = x.reshape(-1, x.shape[-1]) if x.dim() > 2 else x
    out = torch.ops._C.fp8_scaled_mm_cpu(
        x_2d,
        weight,
        weight_scale,
        [self._block_size_n, _BLOCK_SIZE_K], # 合成块形状: N 全覆盖, K 仍按 128 分块
        bias,
        x.dtype,
        True, # is_vnni: 权重已在 process_weights_after_loading 中预打包
    )
    return out.reshape(x.shape[:-1] + (out.size(-1),)) if x.dim() > 2 else out

def apply_scaled_mm(self, *, A, B, out_dtype, As, Bs, bias, output_shape) -> torch.Tensor:
    # 基类模板方法不会被走到, 显式抛错, 与 CPUFP8BlockScaledMMKernel 的既有约定一致
    raise NotImplementedError(
        "CPUFP8PerTensorScaledMMLinearKernel overrides apply_weights directly."
    )

```

评论区精华

本 PR 几乎没有技术层面的评审交锋: [claude\[bot\]](#) 因 PR 来自 fork 直接声明自动评审被禁用 (“This pull request is from a fork — automated review is disabled”), 唯一的人工评审是 [jikunshang](#) 的 APPROVED, 正文为空、无逐行评论; 8 条 comment 全部是 `/ci run` 与 [github-actions\[bot\]](#) 的状态回报。真正的设计权衡只体现在代码注释和 commit message 里: 例如为何要用「合成单块 scale」而不是改算子、为何必须在 `process_weights_after_loading` 里把 `[K, N]` 转回 `[N, K]`、以及为何要在 `can_implement` 提前拒绝非 32 倍数的 N。CI 侧的两处修复也给出了明确的失败证据链 (#89028 视频资产重复下载、#89045 部分权重缓存不触发重试), 属于有据可查的排查结论而非猜测。

- fork PR 的自动评审被禁用, 人工评审无正文 (other): 内核实现、权重朝向修复与 CI 改动均未经过逐行技术复核即被作者本人合并, 评审强度明显低于改动量所对应的水平。
- CI 触发要求分支与 upstream main 完全同步 (other): 经两次 merge main 后, Buildkite CI #89100 在 commit 81162a7 上成功触发, CI 门禁流程走通。
- CPU CI 反复 flaky 的根因定位: 离线重试未触发与多模态资产未持久化 (testing): 以挂载 `~/.cache/vllm` 与扩展 `OFFLINE_RETRY_PATTERN` 两个最小改动收口, 并用 `bash -n` 与真实失败日志做验证。

风险与影响

- 风险:

1. 分派优先级变更影响面被放大：新内核被放在 `_POSSIBLE_FP8_KERNELS[PlatformEnum.CPU]` 的首位，意味着所有 AMX CPU 上的 per-tensor FP8 层都会改走这条新路径。虽然 `can_implement` 会做回退，但该路径此前的行为（是否曾被 `PerTensorTorchFP8ScaledMMLinearKernel` 正确覆盖）并不明确，属于 CPU 推理主路径的行为变更；PR 未附任何性能或吞吐对比数据。
2. AMX 硬性约束下的静默回退：`is_supported` 依赖 `torch.cpu._is_amx_tile_supported()` 与 `ops._supports_cpu_fp8_w8a16`，在非 AMX 或未编译该算子的 CPU 上仍会退回旧的 Torch per-tensor 内核；文档未说明两条路径的数值差异，用户可能观察到不同硬件上精度 / 性能不一致。
3. 测试文件新增模块级 AMX 跳过：`tests/kernels/quantization/test_cpu_fp8_scaled_mm.py` 顶部新增的 `torch.cpu._is_amx_tile_supported()` 守卫会让原有的 block-scaled 内核测试（`test_cpu_fp8_scaled_mm`）在非 AMX 机器上整体不被收集，实际上缩减了原本可运行的覆盖范围，存在覆盖回退风险。
4. 实例状态隐式依赖：`apply_weights` 依赖 `process_weights_after_loading` 中动态设置的 `self._block_size_n`，若某些加载路径跳过或未成功执行该方法，会以 `AttributeError` 形式在首次前向时才暴露，缺少显式初始化或断言。
5. CI 匹配串放宽的副作用：`OFFLINE_RETRY_PATTERN` 新增 `Cannot find any model weights with` 扩大了 online 重试触发面，真实缓存损坏（而非下载不完整）的场景也会被自动重试掩盖；此外挂载 `~/.cache/vllm` 会让多模态 fixture 在 CI 上长期驻留，若断言依赖 fixture 内容更新则需要手动清理缓存。
6. 评审强度偏低：核心内核类在只有一次无正文 `APPROVED` 的情况下合并，且作者本人是 `merged_by`，缺少独立的技术复核。
 - 影响：对用户：直接在 AMX 服务器 CPU（Sapphire Rapids 及更新）上解除了 fp8 量化 Ministral/Pixtral 系列模型的加载崩溃，恢复该模型族在 CPU 侧的可用性；同时因为走 AMX BRGEMM 且不做 FP8 激活量化，W8A16 路径预期比纯 Torch 实现更快，但本 PR 未提供基准数据，收益无法量化。对系统：改变的是 CPU 后端 FP8 内核选择顺序，影响所有 per-tensor FP8 的 CPU 推理流量，非 AMX 及未启用 `fp8_scaled_mm_cpu` 的环境保持旧行为。对团队 / CI：
 - `buildkite/scripts/hardware_ci/run-cpu-test.sh` 的改动会让 CPU CI 的 offline 首跑更容易命中缓存、减少无谓的联网重试与多模态资产重复下载，降低 CPU CI 的 flaky 率与耗时；`tests/v1/e2e/test_cpu_spec_decode.py` 的 checkpoint 替换直接消除 MTP smoke 约 800s 的额外初始化开销。
 - 风险标记：CPU FP8 分派优先级变更，AMX 专属内核路径，测试文件新增 AMX 全局跳过，缺少逐行技术评审，CI 离线重试匹配放宽，无性能基准数据

关联脉络

- PR #53674 [Bugfix][ROCm] Fix MiniMax-M3 fused MXFP8 block scale: 同属“非 CUDA 后端量化内核补洞”的一条线：ROCm 侧修复 fused MXFP8 block scale 下溢并补回归测试，与本案为 CPU 侧补 per-tensor FP8 W8A16 内核并补内核测试的思路一致。
- PR #53940 [ROCm][Kimi-K3] Enable a4w4 flydsl kernels for KimiK3: 同样是通过 `envs/` 分派逻辑把量化模型路由到更合适的后端专用内核，可作为对照理解本案为何要在 `_POSSIBLE_FP8_KERNELS` 中调整优先级。

- PR #56096 [XPU] Route to fused_qk_rmsnorm_rope_gate triton kernel: 同为“平台内核路由”类改动：XPU 通过切换到融合 triton 内核获得性能收益且只改一个模型文件；本案则是在 CPU 分派表首位插入新内核，两者都说明分派表是平台优化的关键杠杆点。
- PR #56351 [CI][ROCm] Add opt-in TheRock builds for AMD CI: 同属 .buildkite 脚本族的 CI 基础设施维护，与本案对 run-cpu-test.sh 的离线重试模式与缓存挂载修复形成 CPU/AMD 两条并行的 CI 稳定性改进。