PR #44340 完整报告

vllm-project/vllm

[Quant] Support compressed-tensors WNA8O8Int linears and WNInt embeddings

合并时间：2026-06-04 22:40

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44340

执行摘要

- 一句话：支持 compressed-tensors WNA8O8Int 线性层和 WNInt embedding
- 推荐动作：值得精读，特别是 Triton kernel 实现（_dequant_gather_kernel）和混合精度内核工厂模式（choose_mp_linear_kernel）。理解如何集成新量化方案到现有架构具有参考价值。

功能与动机

根据 PR 描述，compressed-tensors checkpoint 中经常包含 INT 权重与静态 per-tensor INT8 激活量化，但现有方案未覆盖此模式。此外，量化 embedding 层缺乏原生支持，需要通用反量化方法。通过引入这两种新方案，用户可以直接加载更多类型的 compressed-tensors 量化模型。

实现拆解

1. 新增 CompressedTensorsWNA8O8Int 方案：文件 compressed_tensors_wNa8o8.py，处理 2/4/8 bit INT 权重两种格式（pack-quantized 和 int-quantized），在输入输出应用 fake_quant_static_int8，后端通过 choose_mp_linear_kernel 选择内核（首选 Humming，回退 Marlin/Torch）。
2. 新增量化 Embedding 方案：文件 compressed_tensors_embedding.py，CompressedTensorsEmbeddingWNA16Int 类，利用 Triton kernel _dequant_gather_kernel 实现 fused gather+unpack+dequant，避免全量解包。
3. 引入 HummingLinearKernel：文件 kernels/linear/mixed_precision/humming.py，作为混合精度内核，替换原有的手工后端分发，处理权重重命名和 humming 标准格式转换。
4. 主配置类修改：compressed_tensors.py 注册新方案，扩展 get_quant_method 分发 VocabParallelEmbedding，添加 _is_wNa8o8_int 检测方法，更新输出激活量化配置解析。
5. 配套更改：humming_utils.py 的 convert_linear_layer_to_humming_standard 支持无 TP 分区层；Gemma4 系列模型更新量化键；依赖 requirements 更新 compressed-tensors 版本；增加测试覆盖。

关键文件：

- vllm/model_executor/layers/quantization/compressed_tensors/schemes/compressed_tensors_wNa8o8.py（模块 量化方案；类别 source；类型 data-contract；符号 fake_quant_static_int8, CompressedTensorsWNA8O8Int, init, get_min_capability）：

新增 WNA8O8Int 方案类，核心实现

- vllm/model_executor/layers/quantization/compressed_tensors/compressed_tensors_embedding.py（模块 量化方法；类别 source；类型 data-contract；符号 _dequant_gather_kernel, _dequant_gather_triton, CompressedTensorsEmbeddingWNA16Int, init）：新增量化 Embedding 方案类，含 Triton 内核
- vllm/model_executor/kernels/linear/mixed_precision/humming.py（模块 混合精度内核；类别 source；类型 data-contract；符号 HummingLinearKernel, get_min_capability, can_implement, process_weights_after_loading）：新增 HummingLinearKernel 作为 WNA8O8Int 后端
- vllm/model_executor/layers/quantization/compressed_tensors/compressed_tensors.py（模块 量化配置；类别 source；类型 data-contract；符号 _is_wNa8o8_int）：注册新方案、分发 embedding 方法、添加检测逻辑
- vllm/model_executor/layers/quantization/utils/humming_utils.py（模块 工具函数；类别 source；类型 data-contract；符号 convert_linear_layer_to_humming_standard）：重构 converter，支持 ReplicatedLinear 等无 TP 分区层
- tests/kernels/quantization/test_quantized_embedding.py（模块 测试；类别 test；类型 test-coverage；符号 _dequant_gather_torch, test_dequant_gather）：新增 Triton 反量化收集内核的测试

关键符号：fake_quant_static_int8, CompressedTensorsWNA8O8Int.init, CompressedTensorsWNA8O8Int.create_weights, CompressedTensorsWNA8O8Int.process_weights_after_loading, CompressedTensorsWNA8O8Int._register_weight, CompressedTensorsWNA8O8Int._pack_int_quantized_weight, CompressedTensorsEmbeddingWNA16Int.init, CompressedTensorsEmbeddingWNA16Int.create_weights, CompressedTensorsEmbeddingWNA16Int.process_weights_after_loading, CompressedTensorsEmbeddingWNA16Int.embedding, _dequant_gather_kernel, _dequant_gather_triton, HummingLinearKernel.get_min_capability, HummingLinearKernel.can_implement, HummingLinearKernel.process_weights_after_loading, HummingLinearKernel.apply_weights, convert_linear_layer_to_humming_standard, prepare_humming_layer, CompressedTensorsConfig._is_wNa8o8_int

关键源码片段

vllm/model_executor/layers/quantization/compressed_tensors/schemes/compressed_tensors_wNa8o8.py

新增 WNA8O8Int 方案类，核心实现

```
# SPDX-License-Identifier: Apache-2.0
# 实现 WNA8O8Int 方案，处理 INT 权重 + INT8 激活伪量化

import torch
```

```
from vllm.model_executor.kernels.linear import MPLinearLayerConfig, choose_mp_linear_kernel


def fake_quant_static_int8(x: torch.Tensor, scale: torch.Tensor) -> torch.Tensor:
    # 静态 per-tensor 对称 INT8 伪量化
    scale = scale.to(x.dtype)
    q = torch.clamp(torch.round(x / scale), -128.0, 127.0)
    return q * scale


class CompressedTensorsWNA8O8Int(CompressedTensorsScheme):
    def __init__(self, num_bits: int, strategy: str, group_size: int | None = None,
                 has_input_act: bool = False, has_output_act: bool = False,
                 layer_name: str | None = None, quant_format: str = 'pack-quantized'):
        self.num_bits = num_bits
        self.pack_factor = 32 // num_bits
        self.group_size = -1 if group_size is None else group_size
        self.has_input_act = has_input_act
        self.has_output_act = has_output_act
        self.quant_format = quant_format
        self.is_int_quantized = quant_format == 'int-quantized'
        if num_bits not in WNA8O8_SUPPORTED_TYPES_MAP:
            raise ValueError(f'Unsupported num_bits = {num_bits}')
        self.quant_type = WNA8O8_SUPPORTED_TYPES_MAP[num_bits]
        self._input_scale = None
        self._output_scale = None

    def create_weights(self, layer, output_size, input_size, output_partition_sizes,
                       input_size_per_partition, params_dtype, weight_loader, **kwargs):
        mp_config = MPLinearLayerConfig(
            full_weight_shape=(input_size, output_size),
            partition_weight_shape=(input_size_per_partition, sum(output_partition_sizes)),
            weight_type=self.quant_type, act_type=params_dtype,
            group_size=self.group_size, zero_points=False, has_g_idx=False,
        )
        self.kernel = choose_mp_linear_kernel(mp_config)(
            mp_config, w_q_param_name='weight_packed', w_s_param_name='weight_scale')
        self._register_weight(layer, input_size, input_size_per_partition, params_dtype, weight_
        loader)
    # ... 其他方法
```

vllm/model_executor/layers/quantization/compressed_tensors/compressed_tensors_embedding.py

新增量化 Embedding 方案类，含 Triton 内核

```
# SPDX-License-Identifier: Apache-2.0
# 量化 Embedding：按需反量化打包权重

import torch
```

```
from vllm.triton_utils import tl, triton

@triton.jit
def _dequant_gather_kernel(
    ids_ptr, packed_ptr, scale_ptr, out_ptr, hidden, packed_cols, num_groups,
    NUM_BITS: tl.constexpr, PACK_FACTOR: tl.constexpr, GROUP_SIZE: tl.constexpr, BLOCK: tl.
    constexpr,
):
    row = tl.program_id(0)
    col = tl.program_id(1) * BLOCK + tl.arange(0, BLOCK)
    col_mask = col < hidden
    tid = tl.load(ids_ptr + row).to(tl.int64)
    packed_idx = col // PACK_FACTOR
    shift = (col % PACK_FACTOR) * NUM_BITS
    packed = tl.load(packed_ptr + tid * packed_cols + packed_idx, mask=col_mask, other=0)
    q = ((packed >> shift) & ((1 << NUM_BITS) - 1)) - (1 << (NUM_BITS - 1))
    if GROUP_SIZE == 0: # per-channel
        scale = tl.load(scale_ptr + tid)
    else: # per-group
        grp = col // GROUP_SIZE
        scale = tl.load(scale_ptr + tid * num_groups + grp, mask=col_mask, other=0.0)
    out = q.to(tl.float32) * scale.to(tl.float32)
    tl.store(out_ptr + row * hidden + col, out.to(out_ptr.dtype.element_ty), mask=col_mask)


def _dequant_gather_triton(ids, weight_packed, weight_scale, hidden, num_bits):
    n = ids.numel()
    out = torch.empty(n, hidden, dtype=weight_scale.dtype, device=weight_packed.device)
    num_groups = weight_scale.shape[1]
    group_size = 0 if num_groups == 1 else hidden // num_groups
    block = min(triton.next_power_of_2(hidden), 1024)
    grid = (n, triton.cdiv(hidden, block))
    _dequant_gather_kernel[grid](ids, weight_packed, weight_scale, out, hidden,
        weight_packed.shape[1], num_groups,
        NUM_BITS=num_bits, PACK_FACTOR=32//num_bits, GROUP_SIZE=group_size, BLOCK=
        block)
    return out

class CompressedTensorsEmbeddingWNA16Int(QuantizeMethodBase):
    # ... 实现嵌入层的创建和执行
    def embedding(self, ids):
        return _dequant_gather_triton(ids, self.weight_packed, self.weight_scale,
                                      self.hidden_size, self.num_bits)
```

vllm/model_executor/kernels/linear/mixed_precision/humming.py

新增 HummingLinearKernel 作为 WNA8O8Int 后端

```
# SPDX-License-Identifier: Apache-2.0
# Humming GEMM 作为 WNA16Int 混合精度线性内核
```

```
from vllm.model_executor.layers.quantization.utils.humming_utils import (
    convert_linear_layer_to_humming_standard, prepare_humming_layer)

class HummingLinearKernel(MPLinearKernel):
    @classmethod
    def can_implement(cls, c: MPLinearLayerConfig):
        if not current_platform.is_cuda():
            return False, 'Humming is only supported on CUDA'
        if not _has_module('humming'):
            return False, 'Humming is not installed'
        if c.has_g_idx:
            return False, 'Humming does not support act-order (g_idx)'
        if c.zero_points:
            return False, 'Humming linear kernel only supports symmetric weights'
        return True, None

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        name_map = {'weight': self.w_q_name, 'weight_scale': self.w_s_name}
        group_size = self.config.group_size
        quant_config = {
            'quant_method': 'humming',
            'dtype': 'int' + str(self.config.weight_type.size_bits),
            'group_size': 0 if group_size == -1 else group_size,
        }
        convert_linear_layer_to_humming_standard(layer=layer, name_map=name_map)
        prepare_humming_layer(layer, quant_config)

    def apply_weights(self, layer, x, bias=None):
        from humming.layer import HummingMethod
        flatten_inputs = x.view(-1, x.size(-1))
        output = HummingMethod.forward_layer(layer, flatten_inputs, layer.compute_config)
        return output.view(*x.shape[:-1], output.size(-1))
```

评论区精华

1. 使用辅助函数组织方案检测（dsikka）：建议使用 _is_wNa8o8_int 等静态方法统一检测逻辑，已实现。
2. Embedding 方案校验（dsikka）：应明确拒绝非 WNA16 方案，当前代码在 get_quant_method 中检查并抛 ValueError。
3. Humming 零点和部分对齐支持（jinzhen-lin）：Humming 实际支持 zero_points，但当前 can_implement 返回 False；mgoin 表示未充分测试，计划后续跟进。

- 使用辅助函数组织方案检测 (design): 已实现 _is_wNa8o8_int 静态方法，满足要求。
- Embedding 方案校验应明确拒绝非 WNA16 (correctness): 当前代码在 get_quant_method 中对非 WNA16 方案抛出 ValueError。
- Humming 零点和部分对齐支持 (design): 已知限制，暂不启用，待后续 PR 支持。

风险与影响

- 风险：伪量化方案可能引入精度差异，需用户自行验证。 Humming 内核仅支持 CUDA 且需要 humming 库，未测试零点支持可能导致不准确。 ReplicatedLinear 修复提供了兼容性，但仍可能遗漏其他无 input_size_per_partition 属性的层。 新方案注册可能干扰现有模型加载路径，需关注错误处理是否恰当。
- 影响：用户：可直接加载更多 compressed-tensors 量化模型（WNA8O8Int 线性层和量化 embedding）。 系统：新增 Triton kernel 和 Humming 后端，运行时自动选择。 团队：需维护新增的 kernel 和方案，后续可能跟进 zero_points 支持。
- 风险标记：伪量化精度风险，Humming 零点支持未测试，ReplicatedLinear 兼容性修复，新方案注册错误处理

关联脉络

- 暂无明显关联 PR