PR #42027 完整报告

vllm-project/vllm

[Kernel][MoE] Add GELU_TANH to CPU, CUTLASS, and WNA16 MoE backends

合并时间：2026-06-03 05:12

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42027

执行摘要

- 一句话：为 CPU/CUTLASS/WNA16 MoE 后端添加 GELU_TANH 激活支持
- 推荐动作：值得精读，尤其是 WNA16 量化层从硬编码断言到透传 activation 的设计改进，展示了如何将限制性设计改为参数化，以支持更多激活函数。另外，C++ 后端实现 gelu_tanh_and_mul 时采用了与 PyTorch 相同的近似公式，可作为参考。测试方法使用了 monkeypatch 拦截 fused_experts 来验证参数传递，值得学习。

功能与动机

PR body 明确指出：'Core GELU_TANH MoE activation is already on main, but three backend paths still reject it: CPU fused MoE (KeyError), CUTLASS FP8/FP4 (not listed, unnecessary fallback), WNA16 MoE (hard-asserts SiLU, crash)'。作者在评论中补充说明 Gemma4 的 GELU_TANH 主路径已支持，但剩余后端仍需补齐。

实现拆解

1. CPU C++ 核心(csrc/cpu/cpu_fused_moe.cpp)：在 FusedMOEAct 枚举中新增 GeluTanhAndMul，get_act_type 中添加 'gelu_tanh' 映射；实现 gelu_tanh_and_mul（基于 tanh 近似），并注册到 apply_gated_act 的 switch-case。
2. CPU Python 封装(vllm/model_executor/layers/fused_moe/cpu_fused_moe.py)：在 _CPU_MOE_ACT_FN 字典中添加 MoEActivation.GELU_TANH 条目，使用 F.gelu(approximate='tanh')。
3. CUTLASS 专家层(vllm/model_executor/layers/fused_moe/experts/cutlass_moe.py)：在 CutlassExpertsFp8._supports_activation 和 CutlassExpertsFp4._supports_activation 中添加 GELU_TANH，FP4 还添加 GELU_TANH_NO_MUL。
4. WNA16 量化(vllm/model_executor/layers/quantization/moe_wna16.py)：移除 assert layer.activation == MoEActivation.SILU 和对 MoEActivation 的导入，在 apply 调用 fused_experts 时传递 activation=layer.activation。
5. 测试配套：新增 tests/quantization/test_moe_wna16.py（使用 monkeypatch 验证 activation 参数透传）；扩展现有 tests/kernels/moe/test_cutlass_moe.py（断言 CUTLASS 支持 GELU_TANH 及 NO_MUL）；增强 tests/kernels/moe/test_cpu_fused_moe.py 的 CPU 激活测试覆盖。

关键文件：

- csrc/cpu/cpu_fused_moe.cpp（模块 CPU 内核；类别 source；类型 core-logic；符号 FusedMOEAct, get_act_type, gelu_tanh_and_mul, apply_gated_act）：核心 C++ 实现，新增 GELU_TANH 枚举、映射、计算函数 gelu_tanh_and_mul 和调度分支，是 CPU 后端支持的基础。
- vllm/model_executor/layers/quantization/moe_wna16.py（模块 WNA16 量化；类别 source；类型 data-contract；符号 MoeWNA16Method.apply）：WNA16 量化层是主要变更之一：移除 SiLU-only 硬断言，改为透传 activation 参数，是支持任意激活的关键。
- vllm/model_executor/layers/fused_moe/cpu_fused_moe.py（模块 CPU MoE 封装；类别 source；类型 data-contract；符号 _CPU_MOE_ACT_FN）：CPU 端的 Python 封装层需要添加 GELU_TANH 到激活映射字典，才可调用对应 C++ 核函数。
- vllm/model_executor/layers/fused_moe/experts/cutlass_moe.py（模块 CUTLASS MoE；类别 source；类型 data-contract；符号 _supports_activation）：CUTLASS 专家层的激活支持列表需要加入 GELU_TANH，否则即使主逻辑支持也会因元数据拒绝导致回退。
- tests/quantization/test_moe_wna16.py（模块 测试；类别 test；类型 test-coverage；符号 test_moe_wna16_apply_passes_layer_activation, fake_fused_experts）：新增测试确保 WNA16 在修改后能正确传递 GELU_TANH 给 fused_experts，是验证行为正确性的关键测试。

关键符号：gelu_tanh_and_mul, MoeWNA16Method.apply, CutlassExpertsFp8._supports_activation, CutlassExpertsFp4._supports_activation, get_act_type, test_moe_wna16_apply_passes_layer_activation

关键源码片段

csrc/cpu/cpu_fused_moe.cpp

核心 C++ 实现，新增 GELU_TANH 枚举、映射、计算函数 gelu_tanh_and_mul 和调度分支，是 CPU 后端支持的基础。

```
namespace {
// 新增枚举值 GeluTanhAndMul
enum class FusedMOEAct {
  SiluAndMul,
  SwigluOAIAndMul,
  GeluAndMul,
  GeluTanhAndMul,
};

// 映射字符串到枚举，新增 'gelu_tanh'
FusedMOEAct get_act_type(const std::string& act) {
  if (act == "silu") return FusedMOEAct::SiluAndMul;
  if (act == "swigluoai") return FusedMOEAct::SwigluOAIAndMul;
  if (act == "gelu") return FusedMOEAct::GeluAndMul;
  if (act == "gelu_tanh") return FusedMOEAct::GeluTanhAndMul;
  TORCH_CHECK(false, "Invalid act type: " + act);
}
```

```
// 新增 GELU_TANH 激活函数实现：gate * 0.5 * (1 + tanh(0.79788456 * (gate + 0.044715 *
gate^3))) * up
template <typename scalar_t>
void gelu_tanh_and_mul(float* __restrict__ input, scalar_t* __restrict__ output,
                       const int32_t m_size, const int32_t n_size,
                       const int32_t input_stride,
                       const int32_t output_stride) {
  using scalar_vec_t = typename cpu_utils::VecTypeTrait<scalar_t>::vec_t;
  const int32_t dim = n_size / 2;
  float* __restrict__ gate = input;
  float* __restrict__ up = input + dim;
  vec_op::FP32Vec16 one_vec(1.0);
  // 常量：0.79788456 = sqrt(2/pi), 0.5, 0.044715
  vec_op::FP32Vec16 w1_vec(0.7978845608028654);
  vec_op::FP32Vec16 w2_vec(0.5);
  vec_op::FP32Vec16 w3_vec(0.044715);
  alignas(64) float temp[16];
  for (int32_t m = 0; m < m_size; ++m) {
    for (int32_t n = 0; n < dim; n += 16) {
      vec_op::FP32Vec16 gate_vec(gate + n);
      vec_op::FP32Vec16 up_vec(up + n);
      auto gate_pow3_vec = gate_vec * gate_vec * gate_vec;
      auto inner_vec = w1_vec * (gate_vec + w3_vec * gate_pow3_vec);
      inner_vec.save(temp);
      for (int32_t i = 0; i < 16; ++i) temp[i] = std::tanh(temp[i]);
      vec_op::FP32Vec16 tanh_vec(temp);
      auto gelu_tanh = gate_vec * w2_vec * (one_vec + tanh_vec);
      auto gated_output_fp32 = up_vec * gelu_tanh;
      scalar_vec_t gated_output = scalar_vec_t(gated_output_fp32);
      gated_output.save(output + n);
    }
    gate += input_stride; up += input_stride; output += output_stride;
  }
}

// 在调度函数中注册新分支
template <typename scalar_t>
FORCE_INLINE void apply_gated_act(const FusedMOEAct act, ...) {
  switch (act) {
    case FusedMOEAct::GeluTanhAndMul:
      gelu_tanh_and_mul(input, output, m, n, input_stride, output_stride);
      return;
    // 已有分支 ...
  }
}
```

vllm/model_executor/layers/quantization/moe_wna16.py

WNA16 量化层是主要变更之一：移除 SiLU-only 硬断言，改为透传 activation 参数，是支持任意激活的关键。

```
# 移除了硬编码断言，透传 layer.activation
def apply(self, layer, x, topk_weights, topk_ids,
          shared_experts, shared_experts_input):
    from vllm.model_executor.layers.fused_moe import fused_experts
    # 不再 assert layer.activation == MoEActivation.SILU
    return fused_experts(
        x,
        layer.w13_qweight,
        layer.w2_qweight,
        topk_weights=topk_weights,
        topk_ids=topk_ids,
        activation=layer.activation, # 新增行：透传激活类型
        apply_router_weight_on_input=layer.apply_router_weight_on_input,
        global_num_experts=layer.global_num_experts,
        expert_map=layer.expert_map,
        quant_config=self.moe_quant_config,
    )
```

vllm/model_executor/layers/fused_moe/cpu_fused_moe.py

CPU 端的 Python 封装层需要添加 GELU_TANH 到激活映射字典，才可调用对应 C++ 核函数。

```
# 在激活函数映射字典中添加 GELU_TANH 条目
_CPU_MOE_ACT_FN: dict[MoEActivation, Callable[[torch.Tensor], torch.Tensor]] = {
    MoEActivation.SILU: lambda x: SiluAndMul(compile_native=False).forward_native(x),
    MoEActivation.SWIGLUOAI: _swigluoai_forward_native,
    MoEActivation.GELU: _gelu_and_mul,
    MoEActivation.GELU_TANH: (
        lambda x: F.gelu(x[..., : x.shape[-1] // 2], approximate="tanh")
        * x[..., x.shape[-1] // 2 :]
    ),
}
```

评论区精华

核心讨论：Reviewer AndreasKaratzas 指出新增的 WNA16 测试应限制在 CUDA 平台，因为 WNA16 是 CUDA 量化路径。作者响应并添加了 @pytest.mark.skipif(not current_platform.is_cuda()) 标记。 另外，gemini-code-assist bot 确认无其他反馈。TGMerritt 在 issue 评论中提供了 SM121 硬件的验证数据，确认 CUTLASS FP4 后端选择正确并工作正常。整体讨论较少，主要已在前置 PR #41050 中详述。

- 为 WNA16 测试添加 CUDA 平台跳过标记 (testing): 作者在测试中添加了 @pytest.mark.skipif(not current_platform.is_cuda())，限制测试仅在 CUDA 上运行。

风险与影响

- 风险：数值一致性与性能：CPU C++ 的 gelu_tanh_and_mul 使用 tanh(0.79788456 * x * (1 + 0.044715 * x^2)) 近似，与 PyTorch F.gelu(approximate='tanh') 一致，但需注意精度和边界值（如 tanh 范围为 [-1,1]）。 兼容性：WNA16 移除了 SiLU-only 断言，使得任意 activation 都可通过 fused_experts 处理，但 fused_experts 本身在内部会检查激活支持，故风险较小。 回归潜力：每个后端改动独立，不改变现有 SILU 或 GELU 行为，回归概率低。
- 影响：用户影响：使用 GELU_TANH MoE 激活的模型（如 Gemma4）现在可以在 CPU、CUTLASS FP8/FP4、WNA16 量化后端上正确运行，不再崩溃或回退到其他结构。 系统影响：无性能退化，新增的激活分支只会在请求时触发。 团队维护：统一了各后端的激活支持方式，降低后续添加新激活的阻力（尤以WNA16从硬编码到参数化的转变促进可扩展性）。
- 风险标记：新增激活函数分支（潜在数值差异），移除量化层硬断言（兼容性依赖 fused_experts 内部校验）

关联脉络

- PR #41050 [Kernel][MoE] Add GELU_TANH support to TRT-LLM NvFP4 MoE backend: 本 PR 的作者在评论中提到 #41050 是前置 PR，覆盖了 TRT-LLM NvFP4 路径的相同激活支持，而本 PR 补充剩余后端。