

PR #55355 完整报告

vllm-project/vllm

[Model] Add DeepSeek-V4 CPU backend

合并时间: 2026-09-10 14:14

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55355>

执行摘要

- 一句话: 为 DeepSeek-V4 添加完整的 CPU 推理后端, 涵盖 AVX512/AMX 内核移植与模块化重构。
- 推荐动作: 该 PR 是一个重大且有价值的特性添加, 实现了 DeepSeek-V4 在 CPU 上的高效推理。建议技术管理者和相关工程师精读以下内容: 1) `vllm/models/deepseek_v4/cpu/model.py` 和 `cpu_sparse.py` 中的 CPU 注意力实现, 特别是类交换和自定义操作注册模式; 2) `vllm/model_executor/layers/fused_moe/router/cpu_router.py` 中的 CPURouter 设计, 它展示了如何将 CPU 特定的路由逻辑优雅地集成到统一的 MoE 框架中; 3) `tests/kernels/test_deepseek_v4_cpu_kernels.py` 中的测试策略, 它为内核正确性验证提供了良好范例。总体而言, 这是一个设计良好、实现完整的重大特性, 值得关注其架构决策和技术实现细节。

功能与动机

动机是为 DeepSeek-V4 模型提供高性能的 CPU 推理后端, 以支持在 CPU 硬件上运行该模型。PR body 中明确指出, 目标是将依赖 PyTorch eager 操作的 CPU 路径替换为移植自 SGLang 的、针对 AVX512/AMX 指令集优化的融合内核, 从而显著提升性能。此外, 将 CPU MoE 专家迁移到统一的模块化框架, 是为了与其他后端保持架构一致性, 并为未来的优化奠定基础。代码解耦工作是为了避免在 GPU 代码路径中引入 CPU 特定的分支, 提高代码可维护性。

实现拆解

1. 内核移植与封装: 在 `csrc/cpu/sgl-kernels/` 下新增了 7 个 C++ 内核文件 (如 `flash_mla.cpp`, `store_cache.cpp`, `compressor.cpp`), 实现了 MLA 注意力、KV 缓存存储、压缩器等核心操作的 AVX512/AMX 优化版本。在 `vllm/_custom_ops.py` 中为这些内核添加了 Python 绑定和 `torch.compile` 的 `fake_impl` 注册。
2. CPU 专属模型模块: 在 `vllm/models/deepseek_v4/cpu/` 目录下新建了 `model.py`, `cpu_sparse.py`, `cpu_mla.py`, `cpu_compressor.py` 等文件, 实现了完整的 DeepSeek-V4 CPU 模型结构。关键类包括 `DeepseekV4CPUAttention`、`DeepseekV4CPUIndexer`、`DeepseekV4CPUCompressor`, 它们通过类交换 (class-swapping) 和自定义操作 (`direct_register_custom_op`) 将 CPU 内核集成到模型前向传播中。
3. CPU MoE 模块化: 重构了 `vllm/model_executor/layers/fused_moe/experts/cpu_moe.py`, 将 `CPUUnquantizedExperts` 和 `CPUExpertsFp8` 等类从 `FusedMoEExpertsMonolithic` 迁移到 `FusedMoEExpertsModular` 基类。路由逻辑被提取到新增的

vllm/model_executor/layers/fused_moe/router/cpu_router.py 中，实现了 CPURouter 类，支持 softmax、grouped top-k 和 DeepSeek-V4 特有的 sqrtsoftplus 路由方案。

4. 测试与 CI：新增了 tests/kernels/test_deepseek_v4_cpu_kernels.py，包含了 58 个内核正确性测试，验证 CPU 内核与 eager PyTorch 参考实现的一致性。测试被集成到 buildkite/hardware_tests/cpu.yaml 的 CPU-Kernel Tests 步骤中，并进行了分片（sharded x2）以加速 CI。此外，还添加了 GSM8K 模型评估配置以验证端到端推理质量。

关键文件：

- vllm/models/deepseek_v4/cpu/model.py（模块 CPU 模型；类别 source；类型 data-contract；符号 DeepseekV4MLP, DeepseekV4MoE, DeepseekV4DecoderLayer, hc_pre）：CPU 后端的模型定义入口，包含完整的 DeepSeek-V4 CPU 模型结构（MLP、MoE、DecoderLayer），是理解 CPU 后端如何集成到 vLLM 模型框架的关键。
- vllm/models/deepseek_v4/cpu/cpu_sparse.py（模块 CPU 注意力；类别 source；类型 data-contract；符号 DeepseekV4CPUAttention, DeepseekV4CPUIndexer, deepseek_v4_cpu_prepare_and_attn）：实现了 CPU 版本的稀疏 MLA 注意力，包含关键的自定义操作注册和类交换逻辑，是理解 CPU 注意力如何绕过 GPU 特定代码路径的关键。
- vllm/model_executor/layers/fused_moe/router/cpu_router.py（模块 CPU 路由器；类别 source；类型 data-contract；符号 CPURouter, select_experts, _sqrtsoftplus_bias_topk）：实现了 CPU 版本的 MoE 路由器，支持多种路由方案（包括 DeepSeek-V4 特有的 sqrtsoftplus），是理解 CPU 如何适配统一 MoE 框架的关键。
- tests/kernels/test_deepseek_v4_cpu_kernels.py（模块 内核测试；类别 test；类型 test-coverage；符号 _op_available, _make_cache, _assert_cache_parity）：提供了 CPU 内核的全面正确性测试，验证了移植内核与 eager PyTorch 参考实现的一致性，是评估代码质量的关键。

关键符号：DeepseekV4CPUAttention.forward_mqa, DeepseekV4CPUIndexer.forward, DeepseekV4CPUCompressor.forward, CPURouter._compute_routing, _sqrtsoftplus_bias_topk, DeepseekV4MoE.forward, _deepseek_v4_cpu_prepare_and_attn, forward_cpu

关键源码片段

vllm/models/deepseek_v4/cpu/model.py

CPU 后端的模型定义入口，包含完整的 DeepSeek-V4 CPU 模型结构（MLP、MoE、DecoderLayer），是理解 CPU 后端如何集成到 vLLM 模型框架的关键。

```
class DeepseekV4MoE(nn.Module):
    def __init__(self, vllm_config: VllmConfig, prefix: str = ""):
        super().__init__()
        # ... (省略部分配置读取) ...
        # DeepSeek-V4 特有的 hash MoE 路由配置
        is_hash_moe = extract_layer_index(prefix) < config.num_hash_layers
        self.hash_indices_dtype = torch.int32
        if is_hash_moe:
            # hash MoE 层使用 token-id 到 expert-id 的静态映射表,
```

```

# 而不是基于学习得分的动态路由。
self.gate.tid2eid = nn.Parameter(
    torch.randint(
        0, config.n_routed_experts,
        (config.vocab_size, config.num_experts_per_tok),
        dtype=self.hash_indices_dtype,
    ),
    requires_grad=False,
)
# ... ( 省略 expert 初始化 ) ...
# 使用 FusedMoEFactory 统一管理路由和专家计算
self.experts = FusedMoEFactory(
    # ... ( 其他参数 ) ...
    scoring_func=self.scoring_func,
    e_score_correction_bias=self.gate.e_score_correction_bias,
    hash_indices_table=self.gate.tid2eid, # 传递哈希表给路由器
    # ... ( 其他参数 ) ...
)

```

vllm/models/deepseek_v4/cpu/cpu_sparse.py

实现了 CPU 版本的稀疏 MLA 注意力，包含关键的自定义操作注册和类交换逻辑，是理解 CPU 注意力如何绕过 GPU 特定代码路径的关键。

```

# 注册一个自定义操作，将整个注意力准备和计算过程包装为一个不透明操作。
# 这是为了绕过 torch.compile 对张量变异的跟踪问题，因为 SWA/compressor/indexer
# 的状态没有显式作为张量参数传递。

```

```

def _deepseek_v4_cpu_prepare_and_attn(
    hidden_states, qr, kv, qr_scale, kv_score, indexer_kv_score,
    indexer_weights, positions, o_padded, layer_name,
) -> torch.Tensor:
    layer_name_str = _resolve_layer_name(layer_name)
    # 从全局上下文获取当前层实例（绕过 compile）
    self = get_forward_context().no_compile_layers[layer_name_str]
    # 执行实际的注意力计算（内部会调用 CPU 内核）
    self._prepare_and_attn_fn(
        hidden_states, qr, kv, qr_scale, kv_score,
        indexer_kv_score, indexer_weights, positions, o_padded,
    )
    o = o_padded[:, :self.n_local_heads, :]
    return self._o_proj(o, positions) # 输出投影保持 eager

```

```

direct_register_custom_op(
    op_name="deepseek_v4_cpu_prepare_and_attn",
    op_func=_deepseek_v4_cpu_prepare_and_attn,
    mutates_args=["o_padded"], # 声明 o_padded 会被原地修改
    fake_impl=_deepseek_v4_cpu_prepare_and_attn_fake,
)

```

vllm/model_executor/layers/fused_moe/router/cpu_router.py

实现了 CPU 版本的 MoE 路由器，支持多种路由方案（包括 DeepSeek-V4 特有的 sqrtsoftplus），是理解 CPU 如何适配统一 MoE 框架的关键。

```
class CPURouter(BaseRouter):
    """覆盖 CPU MoE 专家支持的所有路由方案：plain softmax top-k、
    grouped top-k（带可选校正偏差）、自定义路由函数，以及 DeepSeek-V4 的
    sqrtsoftplus 方案（通过移植的 biased_topk_cpu/hash_topk_cpu 内核）。
    """
    def _compute_routing(self, hidden_states, router_logits, indices_type, *, input_ids=None):
        if self.use_grouped_topk:
            # 组 top-k 路由，支持 e_score_correction_bias
            return _grouped_topk(...)
        elif self.scoring_func == "sqrtsoftplus":
            # DeepSeek-V4 特有的路由方案：使用 sqrt(softplus(logit)) 作为权重，
            # 专家选择使用偏差校正分数，但返回的权重是无偏值。
            assert (self._hash_indices_table is not None or self.e_score_correction_bias is not
                    None)
            return _sqrtsoftplus_bias_topk(
                router_logits=router_logits,
                top_k=self.top_k,
                hash_indices_table=self._hash_indices_table,
                # ... (其他参数) ...
            )
        elif self.custom_routing_function is None:
            # 标准 softmax top-k 路由
            return _softmax_topk(router_logits, self.top_k, self.renormalize)
        else:
            # 自定义路由函数
            topk_weights, topk_ids = self.custom_routing_function(...)
            # 确保输出为连续的 float32/int32，因为 cpu_fused_moe 不处理步长
            return topk_weights.to(torch.float32).contiguous(), topk_ids.to(torch.int32).contiguous()
```

评论区精华

本 PR 的 review 讨论相对较少，主要由 bot 评论和 CI 触发命令组成。唯一的实质性 review 来自 [jikunshang](#) 的批准。提交历史显示作者进行了多次迭代修复，包括：

1) 修复了 CPU 平台缺少 `DeepseekV4ForConditionalGeneration` 存根的问题； 2) 修复了 `CPUExpertsInt4` 的 `moe_problem_size` 计算错误； 3) 修复了 DeepSeek-V4 在 CPU 上默认的 `block_size` 配置错误； 4) 移除了不适用的单元测试。这些修复表明 PR 在合并前经过了细致的调试和验证。

- CI 触发与修复 (testing): 所有 CI 问题均已解决，内核测试和模型评估全部通过。
- 代码质量与重构 (design): 重构成功实现了 CPU 后端的模块化和解耦，提高了代码的可维护性和一致性。

风险与影响

- 风险：

1. 平台兼容性风险：新增的 CPU 内核大量使用了 AVX512/AMX 指令集，在不支持这些指令集的旧 CPU 上可能无法运行。csrc/cpu/sgl-kernels/ 中的代码需要运行时或编译时检测 CPU 特性。
 2. 架构侵入性风险：虽然 CPU 路径已尽力与 GPU 路径解耦，但仍在 vllm/model_executor/layers/sparse_attn_indexer.py 和 vllm/v1/attention/backends/mla/indexer.py 等共享文件中添加了 elif current_platform.is_cpu() 分支，这增加了后续维护时意外影响 CPU 路径的可能性。
 3. 量化格式支持不完整：当前 CPU MoE 专家不支持 FP4 缓存（assert not self.use_fp4_cache），且不支持融合的共享专家分组 GEMM 路径（self.is_fused_shared_expert_enabled = False），这限制了部分模型配置在 CPU 上的完整功能。
 4. 测试覆盖局限性：内核测试虽然全面，但主要验证与 eager PyTorch 实现的一致性，对于复杂的端到端场景（如长序列、并发请求）的覆盖可能不足。GSM8K 评估提供了端到端验证，但仅覆盖了基础生成质量。
 - 影响：用户影响：该 PR 显著扩展了 vLLM 的硬件支持范围，使用户能够在 CPU 上运行 DeepSeek-V4 模型，这对于成本敏感、隐私优先或 GPU 资源受限的场景具有重要价值。系统影响：引入了约 9500 行新增代码和 600 行删除代码，主要集中在新的 CPU 后端模块。这增加了代码库的复杂性和维护负担，但也为后续的 CPU 优化奠定了基础。CPU 内核的引入可能对编译时间和二进制大小产生轻微影响。团队影响：为团队增加了 CPU 推理能力的专业知识要求。CPU 内核的维护需要对 AVX512/AMX 指令集和 SGLang 内核有深入理解。模块化重构使 CPU MoE 专家与 GPU 后端保持一致，降低了长期维护成本。
- 风险标记：核心路径变更，平台特定代码，量化格式限制

关联脉络

- PR #56058 [Security][Rust Frontend] Normalize HTTP method labels in metrics: 同为 vLLM 的安全 / 稳定性修复，涉及不同模块（Rust 前端 vs CPU 后端），但都体现了对生产环境鲁棒性的关注。
- PR #56090 [Frontend][EPD] Use JSON arrays for multimodal metadata: 同为性能优化和重构，EPD 代理的优化与 CPU 后端的优化都旨在提升推理效率，但涉及不同子系统（前端 / 协议层 vs 模型执行层）。