PR #35434 完整报告

sgl-project/sglang

[CPU] Fix wrongly causal-masked bidirectional attention

合并时间：2026-08-29 10:49

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35434

执行摘要

- 一句话：修复 CPU 后端双向注意力被误加因果掩码
- 推荐动作：值得精读，尤其是内核 stage-2 循环中 num_keys 计算与掩码条件如何用一个 is_causal 布尔参数同时支持因果与双向两种模式，以及 C++ 内核与 Python 后端之间参数透传的完整链路（声明、schema 注册、backend 调用、测试接线）。该 PR 是理解 SGLang 后端抽象层如何对齐不同硬件实现（CPU AMX 与 Triton）的较好样例；如果你的团队维护 CPU 或注意力后端，建议同步关注测试覆盖缺口。

功能与动机

PR body 明确指出：extend_attention_cpu 总是将 extend 阶段掩码为 causal，对 decoder 自注意力正确，但对 encoder-only 双向自注意力（如 BERT 系 reranker/embedder，bge-reranker）是错误的，模型会静默得到被因果掩码的注意力分数并产生错误输出。修复前 BAAI/bge-reranker-base 对 "hi" 和 "The giant panda ..." 两个文档返回相同分数 2.611328125，修复后分数正确分离为 -8.15625 与 6.1875。

实现拆解

1. 内核扩展（python/sglang/kernels/aot/csrc/cpu/extend.cpp）：为 extend_attention_kernel_impl 与 extend_attention_cpu 新增 bool is_causal 参数（默认 true），stage-2 循环的 num_keys 由固定的 std::min(seq_len_extend, m + BLOCK_M) 改为 is_causal ? std::min(seq_len_extend, m + BLOCK_M) : seq_len_extend，即非因果时遍历完整 extend 范围；同时因果掩码条件从 n + n_size - 1 > m 收紧为 is_causal && n + n_size - 1 > m，非因果场景跳过掩码分支。
2. 后端接线（python/sglang/srt/layers/attention/intel_amx_backend.py）：在 forward_extend 中新增 is_causal = True 的默认判断，当 layer.is_cross_attention 或 layer.attn_type == AttentionType.ENCODER_ONLY 时置为 False，并作为最后一个参数传入 extend_attention_fwd，同时新增 AttentionType 的导入。该模式与 triton_backend.py 保持一致。
3. 算子注册（python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp）：同步更新 extend_attention_cpu 的 C++ 声明与 TORCH_LIBRARY_FRAGMENT(sgl_kernel, m) 中 extend_attention_cpu 的 schema 字符串，追加 bool is_causal=True 参数，保证 Python 侧可通过 torch.ops.sgl_kernel.extend_attention_cpu 透传该参数。

4. 测试配套（test/registered/cpu/test_extend.py）：为 _test_extend_attention_once 增加 is_causal=True 参数，将参考实现 _run_sdpa_forward_extend 的 causal 参数改为 (not is_cross_attn) and is_causal，并在调用 torch.ops.sgl_kernel.extend_attention_cpu 时补传 tree_mask=None 与 is_causal；新增 test_extend_attention_bidirectional，覆盖 b_seq_len_prefix 全零、不同 extend 长度的双向注意力场景，与 PyTorch SDPA 参考结果对比验证。

关键文件：

- python/sglang/kernels/aot/csrc/cpu/extend.cpp（模块 注意力内核；类别 source；类型 core-logic；符号 extend_attention_kernel_impl, extend_attention_cpu）：核心内核变更：stage-2 循环根据 is_causal 决定 key 范围并条件化因果掩码，是修复双向注意力的关键逻辑。
- python/sglang/srt/layers/attention/intel_amx_backend.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 IntelAMXAttnBackend.forward_extend）：后端接线层：决定 is_causal 取值并透传给内核，且与 triton_backend.py 的既有模式对齐，是修复能落地的关键调用侧变更。
- test/registered/cpu/test_extend.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_extend_attention_bidirectional, _test_extend_attention_once）：新增 test_extend_attention_bidirectional，验证 is_causal=False 时内核输出与 PyTorch SDPA 参考一致，是本次修复的回归保障。
- python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp（模块 算子注册；类别 source；类型 data-contract；符号 extend_attention_cpu）：同步算子声明与 TORCH_LIBRARY_FRAGMENT 注册，为 extend_attention_cpu 追加 is_causal 参数，保证 Python 与 C++ 边界参数一致。

关键符号：extend_attention_kernel_impl, extend_attention_cpu, IntelAMXAttnBackend.forward_extend, _test_extend_attention_once, test_extend_attention_bidirectional

关键源码片段

python/sglang/kernels/aot/csrc/cpu/extend.cpp

核心内核变更：stage-2 循环根据 is_causal 决定 key 范围并条件化因果掩码，是修复双向注意力的关键逻辑。

```
// stage 2: 计算三角形部分；当 is_causal 为 false 时退化为完整方块（双向注意力）
if (!is_cross_attn && !kv_from_cache) {
  // 因果场景仅允许第 m 行 query 关注 [0, m + BLOCK_M) 的 key；
  // 双向场景（encoder-only）直接遍历整个 extend 范围，不做截断
  int num_keys = is_causal ? std::min(seq_len_extend, m + BLOCK_M) : seq_len_extend;
  for (int n = 0; n < num_keys; n += BLOCK_N) {
    int n_size = std::min(BLOCK_N, num_keys - n);

    // n_size 是第二次 GEMM 的 K 维，向上填充到 TILE_K
    const int padded_n_size = div_up(n_size, TILE_K) * TILE_K;
```

```
    // 取 key 并做 VNNI 打包
    pack_vnni<scalar_t>(
        /* dst */ Btmp,
        /* src */ k_extend + (seq_extend_start_loc + n) * ke_strideN + head_kv_id * ke_strideH,
        /* N */ n_size,
        /* K */ head_size,
        /* ld_src */ ke_strideN,
        /* ld_dst */ BLOCK_N);

    // 计算 s_i <- Q @ K
    at::native::cpublas::brgemm(
        /* M */ m_size,
        /* N */ n_size,
        /* K */ head_size,
        /* lda */ q_strideM,
        /* ldb */ BLOCK_N,
        /* ldc */ BLOCK_N,
        /* add_C */ false,
        /* A */ q_ptr,
        /* B */ Btmp,
        /* C */ s_i);

    // 应用 tree mask（投机解码 TARGET_VERIFY）或因果掩码
    if (tree_mask != nullptr) {
      // [Note] tree mask 用于 EAGLE topk > 1（TreeMaskMode::QLEN_ONLY）。
      // mask[bs][m + row][n + col] == false 表示 query 位置 (m + row)
      // 不能关注 key 位置 (n + col)，softmax 前将分数置为 -inf。
      // tree mask 已蕴含因果约束，无需再叠加 causal 判断
      // ... 按 tree_mask 填充掩码 ...
    } else if (is_causal && n + n_size - 1 > m) {
      // 仅当 is_causal 且当前 key 块末尾 (n + n_size - 1) 严格超过
      // 第一个 query 位置 m 时才需要施加因果掩码；非因果场景跳过
      // ... 按 causal 三角形填充掩码 ...
    }
  }
}
```

python/sglang/srt/layers/attention/intel_amx_backend.py

后端接线层：决定 is_causal 取值并透传给内核，且与 triton_backend.py 的既有模式对齐，是修复能落地的关键调用侧变更。

```
# 预取本批次 extend 阶段需要的元数据（seq_lens、extend_seq_lens、extend_start_loc、tree_
mask）
seq_lens, extend_seq_lens, extend_start_loc, tree_mask = self.extend_metadata
_, max_extend_len = self.forward_metadata
if seq_lens.dtype != torch.int64:
    seq_lens = seq_lens.to(torch.int64)

# 判定是否需要在 extend 阶段施加因果掩码：
```

```
# 1. cross attention 的 key 来自 encoder 输出缓存，天然不适用因果掩码；
# 2. encoder-only 自注意力（如 BERT 系 reranker）需要完整的双向注意力；
# 其余情况保持默认的因果掩码行为，与 Triton 后端判断逻辑一致
is_causal = True
if layer.is_cross_attention or layer.attn_type == AttentionType.ENCODER_ONLY:
    is_causal = False

# Gemma4 的 KV 共享层传入 k=v=None，共用层已把 extend K/V 写入缓存
self.extend_attention_fwd(
    q.view(-1, layer.tp_q_head_num, layer.qk_head_dim),
    k,
    v,
    o.view(-1, layer.tp_q_head_num, layer.v_head_dim),
    self.token_to_kv_pool.get_key_buffer(layer.layer_id),
    self.token_to_kv_pool.get_value_buffer(layer.layer_id),
    self.req_to_token_pool.req_to_token,
    forward_batch.req_pool_indices,
    seq_lens,
    extend_seq_lens,
    extend_start_loc,
    max_extend_len,
    layer.scaling,
    layer.logit_cap,
    layer.is_cross_attention,
    layer.sliding_window_size + 1,
    forward_batch.encoder_lens,
    sinks,
    tree_mask,
    is_causal,
)
```

test/registered/cpu/test_extend.py

新增 test_extend_attention_bidirectional，验证 is_causal=False 时内核输出与 PyTorch SDPA 参考一致，是本次修复的回归保障。

```
def _test_extend_attention_once(
    self,
    B,
    N_CTX,
    H_Q,
    H_KV,
    D,
    DV,
    sliding_window=None,
    has_sink=False,
    mla=False,
    is_cross_attn=False,
    *,
    b_seq_len_prefix=None,
```

```
    b_seq_len_extend=None,
    kv_from_cache=False,
    is_causal=True, # 新增参数：控制参考实现与内核的因果掩码行为
):
    # ... 构造 q_extend、k_extend、v_extend、缓存与索引 ...

    # 参考实现：only cross attn 或非 causal 时不做因果掩码
    self._run_sdpa_forward_extend(
        q_extend,
        o_ref,
        k_buffer,
        v_buffer,
        req_to_tokens,
        b_req_idx,
        b_seq_len,
        b_seq_len_prefix,
        b_seq_len_extend,
        scaling=sm_scale,
        enable_gqa=enable_gqa,
        causal=(not is_cross_attn) and is_causal,
        is_cross_attn=is_cross_attn,
        encoder_lens=encoder_lens,
    )

    # 内核调用：补传 tree_mask 与 is_causal
    torch.ops.sgl_kernel.extend_attention_cpu(
        q_extend,
        None if kv_from_cache else k_extend,
        None if kv_from_cache else v_extend,
        o_extend,
        k_buffer,
        v_buffer,
        req_to_tokens,
        b_req_idx,
        b_seq_len,
        b_seq_len_extend,
        b_start_loc_extend,
        max_len_extend,
        sm_scale,
        logit_cap,
        is_cross_attn,
        sliding_window if sliding_window is not None else 0,
        encoder_lens,
        sinks if has_sink else None,
        None, # tree_mask
        is_causal,
    )

    torch.testing.assert_close(o_ref, o_extend, atol=1e-2, rtol=1e-2)
```

```
def test_extend_attention_bidirectional(self):
    # 覆盖 is_causal=False 场景：encoder-only 自注意力（例如 bge-reranker）
    # prefix 全为 0，extend 长度各不相同，验证双向注意力下的数值正确性
    self._test_extend_attention_once(
        B=4,
        N_CTX=123,
        H_Q=16,
        H_KV=4,
        D=128,
        DV=96,
        b_seq_len_prefix=[0, 0, 0, 0],
        b_seq_len_extend=[41, 90, 123, 5],
        is_causal=False,
    )
```

评论区精华

本 PR 没有实质性的 review 技术讨论：Fridge003 直接批准（APPROVED），review_comments 为空。PR 评论区仅包含作者触发 CI 的命令，以及一条关于 CI 失败归属的说明：作者指出 XPU CI 失败将由 sgl-project/sglang#36529 修复，与本变更无关。

- CI 失败与依赖修复 (other): XPU CI 失败与本变更无关，等待上游 PR 修复；未影响合并。

风险与影响

- 风险：
  1. 默认参数保持兼容：is_causal 默认值为 true，现有未显式传入该参数的调用（如 Gemma 4 KV 共享层路径，内核注释明确提到该场景由内核自行因果掩码）行为不变，向后兼容风险较低。
  2. 正确性依赖 AttentionType 标记：is_causal 判定依赖 layer.attn_type == AttentionType.ENCODER_ONLY，若某个 encoder-only 模型未正确设置该标记，仍会被误判为因果；该模式与 Triton 后端一致，但 CPU 后端缺少对这类未标记模型的守卫。
  3. 性能影响：非因果场景下 num_keys 遍历完整 extend 范围，计算量按双向注意力正常增长，但仅影响 encoder-only 层，decoder 与 cross-attention 路径无变化。
  4. 测试覆盖缺口：新增测试 test_extend_attention_bidirectional 的 b_seq_len_prefix 全为 0，未覆盖 prefix 加 extend 混合、以及 tree_mask 与 is_causal=False 组合的路径；tree_mask 分支在非因果下是否需跳过因果子约束未在测试中验证。
  5. 发布耦合：sgl-kernel 的 C++ 算子签名变更需要 CPU wheel 重新编译，若发布节奏不同步，可能出现 Python 侧传参与旧库不匹配的 ABI 问题。 - 影响：影响范围集中在 Intel AMX CPU 后端：修复了 BERT 系 reranker/embedder（如 bge-reranker-base）在 CPU 上输出错误分数的问题，属于静默正确性 bug 的修复。对 decoder-only 模型、cross-attention 路径无行为变化。对团队而言，该 PR 补齐了 CPU 后端与 Triton 后端在 is_causal 语义上的一致性，为后续 CPU 上支持更多 encoder 结构模型扫清障碍；测试文件的改动也为 CPU 注意力回归测试增加了双向场景覆盖。 - 风险标记：核心注意力内核变更，默认参数保证向后兼容，依赖 AttentionType 标记正确，缺少

prefix+extend 混合双向场景测试，需要 sgl-kernel 重新编译

关联脉络

- PR #36914 [Fix] Lazy-import aiter in DSv4 paged_decode to unbreak CPU CI: 同属 CPU 后端与 CPU CI 维护，反映 CPU 路径的持续修复与 CI 保障，与本 PR 的 CPU 注意力修复相互印证。
- PR #35453 [Fix] Support LSE on the RadixAttention extra-kwargs graph path: 同为注意力计算路径的正确性 bugfix，虽后端不同（RadixAttention/kv-cache），但体现 SGLang 在注意力语义正确性上的持续收敛。
- PR #36852 [ROCm][Bugfix] Use token-level KV indices in the aiter ASM context-prefill gather: 同为 extend/prefill 阶段注意力计算的正确性修复，属于不同硬件后端（AMD/ROCm）上同类注意力内核问题的平行修复。