PR #45111 完整报告

vllm-project/vllm

[Attention] Re-enable cross-layer KV cache layout for MLA via stride-aware kernels

合并时间：2026-06-22 21:57

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45111

执行摘要

- 一句话：修复三个 MLA 内核 stride bug，重新启用跨层 KV 缓存布局
- 推荐动作：建议关注注意力系统和 MLA 模型的开发者精读。该 PR 展现了从 issue 分析到内核修复再到测试验证的完整工程实践，尤其是如何构造 torch.as_strided 跨层视图进行位精确等价测试的方法值得推广。

功能与动机

Issue #37032 报告 GLM-4.7-Flash 启用 KV offloading 时输出乱码。PR #37090 紧急禁止 MLA 模型使用跨层 KV 缓存布局，牺牲了该布局带来的内存和性能优势。本 PR 确定根因为少数内核未正确使用缓存的 block 维 stride，修复后可安全恢复跨层布局，同时保持对未验证后端的保守行为。

实现拆解

1. 内核 stride 修复：修改 vllm/v1/attention/ops/triton_decode_attention.py 中两个 stage-1 解码 kernel，将 page_stride 参数化；修改 csrc/libtorch_stable/attention/mla/sm100_cutlass_mla_kernel.cu 中 stride_C 从硬编码改为读取 kv_c_and_k_pe_cache.stride(0)/stride(1)；修改 csrc/libtorch_stable/cache_kernels.cu 中 indexer_k_quant_and_cache 的 block 基地址计算改用 kv_cache.stride(0)。
2. Per-backend opt-in：在 TritonMLABackend、CutlassMLABackend、FlashAttnMLABackend、FlashMLABackend、FlashInferMLABackend 五个后端中静态重写 get_kv_cache_stride_order()，在 include_num_layers_dimension=True 时返回 (1,0,2,3)（num_blocks 居前），表示支持跨层布局。MLACommonBackend 保持恒等排列 (0,1,2,3) 作为安全默认，未验证后端保持 opt-out。
3. 新增测试：新增 tests/kernels/attention/test_mla_cross_layer_kernel_equivalence.py，包含 7 个测试函数，用 torch.as_strided 构造跨层视图，验证 concat_and_cache_mla 写入、FlashMLA 密集 / 稀疏解码、FlashInfer MLA 解码、FA3 解码、indexer_k_quant_and_cache 的位精确等价和无泄漏。在 test_triton_decode_attention.py 新增参数化测试 test_decode_attention_cross_layer_view，覆盖 MLA/GQA/MHA 路径。在 test_cutlass_mla_decode.py 新增 test_cutlass_mla_decode_cross_layer_view。
4. 更新布局测试：test_kv_cache_layout.py 重命名旧测试为 test_mla_common_backend_rejects_cross_layer_kv_cache，新增参数化测试

test_verified_mla_backends_support_cross_layer_kv_cache 验证五个后端正返回 (1,0,2,3)。

关键文件：

- tests/kernels/attention/test_mla_cross_layer_kernel_equivalence.py（模块 跨层内核测试；类别 test；类型 test-coverage；符号 test_concat_and_cache_mla_into_unified_slot_view, write, test_flashmla_dense_decode_unified_slot_view, run）：新增测试套件，验证所有 MLA 内核在跨层视图下与连续缓存位精确等价，是发现和验证 stride bug 的核心测试。
- tests/kernels/attention/test_triton_decode_attention.py（模块 Triton 解码测试；类别 test；类型 test-coverage；符号 test_decode_attention_cross_layer_view, run）：新增 test_decode_attention_cross_layer_view 参数化测试，验证 Triton 解码内核处理跨层视图的正确性，覆盖 MLA/GQA/MHA 三种注意力模式。
- tests/kernels/attention/test_cutlass_mla_decode.py（模块 Cutlass 解码测试；类别 test；类型 test-coverage；符号 test_cutlass_mla_decode_cross_layer_view, run）：新增 test_cutlass_mla_decode_cross_layer_view，验证 sm100 Cutlass MLA decode 内核在跨层视图下的正确性，这是修复的 kernel 之一。
- tests/v1/kv_connector/unit/test_kv_cache_layout.py（模块 缓存布局测试；类别 test；类型 test-coverage；符号 test_mla_common_backend_rejects_cross_layer_kv_cache, test_verified_mla_backends_support_cross_layer_kv_cache, test_deepseek_v32_indexer_rejects_cross_layer_kv_cache）：更新布局测试，反映 MLACommonBackend 默认拒绝跨层，而五个已验证后端 opt-in。
- vllm/v1/attention/ops/triton_decode_attention.py（模块 Triton 解码 kernel；类别 infra；类型 core-logic；符号 _page_stride）：核心修复文件：引入 _page_stride 辅助函数，并修改 decode_attention_fwd 调用处，使内核使用缓存实际的 page-dim stride。
- vllm/v1/attention/backends/mla/triton_mla.py（模块 MLA 后端；类别 source；类型 core-logic；符号 get_kv_cache_stride_order）：Triton MLA 后端 opt-in 跨层布局，override get_kv_cache_stride_order 返回 (1,0,2,3)。
- csrc/libtorch_stable/cache_kernels.cu（模块 缓存内核；类别 other；类型 core-logic）：修复 indexer_k_quant_and_cache 的 block base 寻址，使用 kv_cache.stride(0) 替代硬编码乘积，避免跨层写入时污染相邻层。

关键符号：_page_stride, get_kv_cache_stride_order,
test_concat_and_cache_mla_into_unified_slot_view,
test_flashmla_dense_decode_unified_slot_view,
test_flashinfer_mla_dense_decode_unified_slot_view,
test_flashmla_fp8_sparse_decode_unified_slot_view,
test_indexer_k_quant_and_cache_into_unified_slot_view,
test_flashattn_mla_dense_decode_unified_slot_view,
test_decode_attention_cross_layer_view, test_cutlass_mla_decode_cross_layer_view,
test_mla_common_backend_rejects_cross_layer_kv_cache,
test_verified_mla_backends_support_cross_layer_kv_cache

关键源码片段

tests/kernels/attention/test_mla_cross_layer_kernel_equivalence.py

新增测试套件，验证所有 MLA 内核在跨层视图下与连续缓存位精确等价，是发现和验证 stride bug 的核心测试。

```
import pytest
import torch

pytestmark = pytest.mark.skipif(
    not torch.cuda.is_available(), reason="MLA cache kernels require CUDA"
)

def test_concat_and_cache_mla_into_unified_slot_view():
    """验证 concat_and_cache_mla 写入跨层缓存视图的正确性，确保无泄漏到相邻层。"""
    from vllm import _custom_ops as ops

    torch.manual_seed(0)
    dev = "cuda"
    kv_lora_rank = 512
    pe = 64
    entry = kv_lora_rank + pe
    page = 64
    num_blocks = 32
    ntok = 200

    # 随机输入数据
    kv_c = torch.randn(ntok, kv_lora_rank, device=dev, dtype=torch.bfloat16)
    k_pe = torch.randn(ntok, pe, device=dev, dtype=torch.bfloat16)
    slot = torch.randperm(num_blocks * page, device=dev, dtype=torch.int64)[:ntok]
    scale = torch.tensor(1.0, device=dev)

    def write(cache):
        ops.concat_and_cache_mla(kv_c, k_pe, cache, slot, "auto", scale)

    # 参考：连续单层缓存，形状 (num_blocks, page, entry)
    ref = torch.zeros(num_blocks, page, entry, device=dev, dtype=torch.bfloat16)
    write(ref)

    # 跨层缓存：每个 block 包含 3 层，取中间层的视图（stride(0) = 完整 slot 大小）
    layer_page_elems = page * entry
    n_layers = 3
    unified_slot_elems = n_layers * layer_page_elems
    big = torch.zeros(num_blocks, unified_slot_elems, device=dev, dtype=torch.bfloat16)
    flat = big.view(-1)
    offset = layer_page_elems # 中间层
    view = torch.as_strided(
        flat,
        size=(num_blocks, page, entry),
```

```
        stride=(unified_slot_elems, entry, 1),
        storage_offset=offset,
    )
    write(view)

    # 位精确等价检验
    max_diff = (ref.float() - view.float()).abs().max().item()
    assert max_diff == 0.0, f"max|Δ| = {max_diff}"

    # 验证相邻层未被污染
    neighbour_lo = torch.as_strided(
        flat, (num_blocks, layer_page_elems), (unified_slot_elems, 1), 0
    )
    neighbour_hi = torch.as_strided(
        flat,
        (num_blocks, layer_page_elems),
        (unified_slot_elems, 1),
        2 * layer_page_elems,
    )
    assert neighbour_lo.abs().max().item() == 0.0
    assert neighbour_hi.abs().max().item() == 0.0
```

tests/kernels/attention/test_triton_decode_attention.py

新增 test_decode_attention_cross_layer_view 参数化测试，验证 Triton 解码内核处理跨层视图的正确性，覆盖 MLA/GQA/MHA 三种注意力模式。

```
@pytest.mark.parametrize(
    "H_Q,H_KV,D_QK,D_V,is_mla",
    [
        (16, 1, 576, 512, True), # MLA 路径 (grouped kernel, v = trans(k))
        (32, 8, 128, 128, False), # GQA 路径 (grouped kernel)
        (32, 32, 128, 128, False), # MHA 路径 (normal kernel)
    ],
)
@pytest.mark.parametrize("PAGE_SIZE", [16])
def test_decode_attention_cross_layer_view(H_Q, H_KV, D_QK, D_V, is_mla, PAGE_SIZE):
    """验证解码内核使用缓存的实际 page-dim stride，而非假设 pages 连续排列。
    跨层缓存中每层的视图 stride(0) 被 num_layers 放大，输出必须与连续缓存完全一致。"""
    B = 3
    seq_len = 1027
    CACHE_SIZE = 16384
    NUM_LAYERS = 3
    LAYER_IDX = 1
    dtype = torch.bfloat16
    sm_scale = 1.0 / (D_QK**0.5)
    num_kv_splits = 8
    num_pages = CACHE_SIZE // PAGE_SIZE

    # 构建连续缓存的参考输出
```

```
    k_ref = torch.randn(num_pages, PAGE_SIZE, H_KV, D_QK, dtype=dtype, device=DEVICE_
    TYPE)
    if is_mla:
        v_ref = k_ref[..., :D_V]
    else:
        v_ref = torch.randn(num_pages, PAGE_SIZE, H_KV, D_V, dtype=dtype, device=DEVICE_
        TYPE)

    # 跨层缓存：相邻层填充随机垃圾，确保 packed-pages 寻址会读到错误数据
    k_xl = torch.randn(num_pages, NUM_LAYERS, PAGE_SIZE, H_KV, D_QK, dtype=dtype, device=
    DEVICE_TYPE)
    k_view = k_xl[:, LAYER_IDX]
    k_view.copy_(k_ref)
    if is_mla:
        v_view = k_view[..., :D_V]
    else:
        v_xl = torch.randn(num_pages, NUM_LAYERS, PAGE_SIZE, H_KV, D_V, dtype=dtype,
        device=DEVICE_TYPE)
        v_view = v_xl[:, LAYER_IDX]
        v_view.copy_(v_ref)

    def run(k_buffer, v_buffer):
        o = torch.zeros(B, H_Q, D_V, dtype=dtype, device=DEVICE_TYPE)
        lse = torch.zeros(B, H_Q, dtype=dtype, device=DEVICE_TYPE)
        attn_logits = torch.empty((B, H_Q, num_kv_splits, D_V + 1), dtype=torch.float32, device=
        DEVICE_TYPE)
        decode_attention_fwd(q, k_buffer, v_buffer, o, lse, req_to_page, b_seq_len,
                             attn_logits, num_kv_splits, sm_scale, PAGE_SIZE, is_mla=is_mla)
        return o, lse

    o_ref, lse_ref = run(k_ref, v_ref)
    o_xl, lse_xl = run(k_view, v_view)
    assert torch.equal(o_ref, o_xl)
    assert torch.equal(lse_ref, lse_xl)
```

tests/kernels/attention/test_cutlass_mla_decode.py

新增 test_cutlass_mla_decode_cross_layer_view，验证 sm100 Cutlass MLA decode 内核在跨层视图下的正确性，这是修复的 kernel 之一。

```
@pytest.mark.skipif(
    not current_platform.has_device_capability(100),
    reason=CUTLASS_MLA_UNSUPPORTED_REASON,
)
@torch.inference_mode()
def test_cutlass_mla_decode_cross_layer_view():
    """验证 Cutlass MLA decode 内核使用缓存的 page-dim stride，而非假设 pages 连续排列。
    跨层视图的 stride(0) 被 num_layers 放大，输出必须与连续缓存完全一致。"""
    device = torch.device("cuda:0")
    torch.set_default_dtype(torch.bfloat16)
```

```
torch.set_default_device(device)
torch.manual_seed(42)

b, mean_sk, d, dv, block_size = 4, 512, 576, 512, 64
num_layers, layer_idx = 3, 1
scale = math.sqrt(d) ** (-1)

num_pages = b * (mean_sk // block_size)
cache_seqlens = torch.full((b,), mean_sk, dtype=torch.int32)
block_table = torch.arange(num_pages, dtype=torch.int32).view(b, mean_sk // block_size)

# 连续缓存参考
kv_contig = torch.randn(num_pages, block_size, d)
# 跨层缓存，相邻层填随机数据
kv_cross_layer = torch.randn(num_pages, num_layers, block_size, d)
kv_view = kv_cross_layer[:, layer_idx]
kv_view.copy_(kv_contig)
assert kv_view.stride(0) == num_layers * block_size * d

q_nope = torch.randn(b, 128, dv)
q_pe = torch.randn(b, 128, d - dv)
sm_count = num_compute_units(device.index)
workspace_size = ops.sm100_cutlass_mla_get_workspace_size(mean_sk, b, sm_count, num_
kv_splits=1)
workspace = torch.empty(workspace_size, dtype=torch.uint8)

def run(cache):
    out = torch.empty(b, 128, dv)
    lse = torch.empty(b, 128, dtype=torch.float32)
    ops.sm100_cutlass_mla_decode(out, lse, q_nope, q_pe, cache, cache_seqlens,
                                 block_table, workspace, scale, 1)
    return out, lse

out_contig, lse_contig = run(kv_contig)
out_view, lse_view = run(kv_view)

assert torch.equal(out_contig, out_view)
assert torch.equal(lse_contig, lse_view)
```

评论区精华

仅有一条 review 讨论：LucasWilkinson 在 vllm/v1/attention/ops/triton_decode_attention.py 中对 _page_stride 的实现提出改良建议，推荐使用 unflatten 比手动 stride 计算更清晰。ivanium 接受建议并立即修正。该讨论无争议，门关闭之前已 resolve。

- _page_stride 实现方式 (style): ivanium 修改了实现，改用了 unflatten 方案。

风险与影响

- 风险：核心注意力路径变更，影响所有 MLA 模型（DeepSeek、GLM-4.7 等）。主要风险：虽全面测试覆盖已验证后端，但未验证后端（ROCm AITER、tokenspeed、XPU）保持 identity 排列，完全不受影响。Triton decode attention 也服务非 MLA 模型，但测试已覆盖 GQA/MHA 路径，且修复仅在 stride 使用方式上，GPU 计算逻辑不变，回归风险低。性能影响：新增 stride 读取开销可忽略。兼容性：get_kv_cache_stride_order 协议扩展为每个后端可显式 opt-in，向后兼容。
- 影响：用户：MLA 模型（如 DeepSeek-V2/V3、GLM-4.7）在使用 KV offloading 时不再产生乱码，并可重新利用跨层缓存布局减少内存占用、提升吞吐。开发者：新增 MLA 后端时需覆写 get_kv_cache_stride_order 以选择是否支持跨层；测试套件为后续 kernel 的 stride 正确性提供了校验模板。系统：无外部接口变更，所有影响限于内部内核和注意力层。
- 风险标记：注意力路径核心变更，回归风险低但影响面广，未验证后端自动 opt-out

关联脉络

- PR #37090 [Bugfix] Disable cross-layer KV cache for MLA attention backends: 本 PR 直接修复该 PR 中识别的 stride bug，并重新启用跨层布局。
- PR #37032 [Bug]: GLM 4.7-flash returns gibberish when native KV cache offloading is on: 触发跨层禁用的问题报告，本 PR 彻底修复其根本原因。
- PR #34742 refactor(attention): add default get_kv_cache_stride_order implementation: 为 stride order 提供默认实现，本 PR 在其基础上为各 backend 做 opt-in。
- PR #41093 [P/D][Mooncake] Add cross-layer KV cache support to MooncakeConnector: 跨层布局的配套工作，本 PR 使 MLA 内核兼容此类布局。
- PR #44458 [4/N][KV-Cache Layout Refactor] Standardize KV cache layout: 布局标准化系列 PR 之一，本 PR 的内核修复是使 MLA 支持标准布局的前提。