

PR #43296 完整报告

vllm-project/vllm

[CI] Fix "test_awq_load[gemma4-moe-*)" failure

合并时间: 2026-05-22 15:14

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/43296>

执行摘要

- 一句话: 修复 Gemma4 视觉编码器 OOM, 重构 batch 分块为静态函数
- 推荐动作: 值得精读。本 PR 展示了一个典型的内存错误修复流程: 识别真正的 dominant 瞬时分配, 通过提取纯函数构造可测试的预算模型, 并利用单元测试覆盖边界条件。review 中关于平台可移植性的讨论也提醒了在多后端框架中谨慎使用 CUDA 特定 API。该修复虽针对特定模型, 但其方法可推广至其他视觉或多模态模型的内存优化。

功能与动机

修复 PR #43169 导致的回归: 在 22 GiB L4 上加载 26B AWQ 模型后仅剩 ~3 GiB 空闲, 原预算算法 (5% total memory) 计算出 chunk ≈ 53 , 导致 F.one_hot 分配 2.88 GiB 缓冲区 OOM, 进而使 CI 测试 `test_awq_load[gemma4-moe-*)` 失败。

实现拆解

1. 重构内存预算计算: 将 `_encoder_max_batch` 实例方法改为静态方法 `_encoder_chunk`, 接受 `free_bytes`、`total_bytes`、`position_embedding_size` 参数, 使用 `min(free // 2, total // 10)` 作为预算, 以 `F.one_hot` 的分配大小作为成本项, 并处理 `patches_per_item ≤ 0`、`cost ≤ 0` 等边界条件返回 1。
2. 在 `_process_image_input` 和 `_process_video_input` 中, 调用 `torch.cuda.mem_get_info()` (经 review 后可能已改用平台抽象) 获取当前空闲和总内存, 传入 `_encoder_chunk` 计算分块大小, 并移除了实例属性 `_encoder_budget_bytes` 和 `_encoder_bytes_per_patch` 的惰性初始化。
3. 在 `tests/models/multimodal/processing/test_gemma4.py` 中新增 5 个单元测试, 直接测试 `_encoder_chunk` 静态方法: 紧预算下 one-hot 分配不超预算、宽松 GPU 保持批处理、patches 为零安全、`position_embedding_size` 为零不除零、空闲内存为零回退到 1。这些测试无需 GPU, 可在 CPU 上快速执行。

关键文件:

- `vllm/model_executor/models/gemma4_mm.py` (模块 多模态模型; 类别 source; 类型 core-logic; 符号 `_encoder_max_batch`, `_encoder_chunk`, `_process_image_input`, `_process_video_input`): 核心文件, 将实例方法 `_encoder_max_batch` 重构为静态 `_encoder_chunk`, 修正内存预算为基于当前空闲显存并考虑 `F.one_hot` 成本; 在图像和视频处理中调用。

- tests/models/multimodal/processing/test_gemma4.py (模块测试; 类别 test; 类型 test-coverage; 符号 test_encoder_chunk_tight_budget_fits_in_free, test_encoder_chunk_roomy_gpu_keeps_batching, test_encoder_chunk_zero_patches_is_safe, test_encoder_chunk_zero_position_embedding_size_is_safe) : 新增 5 个单元测试直接测试 _encoder_chunk, 覆盖边界条件且无需 GPU, 可在 CPU CI 上运行。

关键符号: _encoder_chunk, test_encoder_chunk_tight_budget_fits_in_free, test_encoder_chunk_roomy_gpu_keeps_batching, test_encoder_chunk_zero_patches_is_safe, test_encoder_chunk_zero_position_embedding_size_is_safe, test_encoder_chunk_no_free_memory_falls_back_to_one

关键源码片段

vllm/model_executor/models/gemma4_mm.py

核心文件, 将实例方法 _encoder_max_batch 重构为静态 _encoder_chunk, 修正内存预算为基于当前空闲显存并考虑 F.one_hot 成本; 在图像和视频处理中调用。

```
# 静态方法: 计算视觉编码器最大安全 chunk 大小
@staticmethod
def _encoder_chunk(
    patches_per_item: int, # 每项的平均 patch 数
    free_bytes: int, # 当前 GPU 空闲显存 (bytes)
    total_bytes: int, # GPU 总显存 (bytes)
    position_embedding_size: int, # F.one_hot 的类别数
) -> int:
    '''Max chunk size whose F.one_hot transient fits in the budget.'''
    if patches_per_item <= 0:
        return 1
    # 预算: 空闲显存的一半, 但不超过总显存 10%, 为 profile_run 和后续编码留空间
    budget = min(free_bytes // 2, total_bytes // 10)
    if budget <= 0:
        return 1
    # 成本公式: F.one_hot 分配 (chunk, patches, 2, pos_emb_size) 的 int64 + 并发转换
    cost = patches_per_item * 4 * position_embedding_size * 8
    return max(1, budget // cost) if cost > 0 else 1
```

tests/models/multimodal/processing/test_gemma4.py

新增 5 个单元测试直接测试 _encoder_chunk, 覆盖边界条件且无需 GPU, 可在 CPU CI 上运行。

```
# 回归防护测试: 模拟 L4 22GB + 26B AWQ 紧预算场景
from vllm.utils.mem_constants import GiB_bytes

def test_encoder_chunk_tight_budget_fits_in_free():
    free = 3 * GiB_bytes # 实际空闲显存
    total = 22 * GiB_bytes # 总显存
    chunk = _encoder_chunk( # 调用静态方法
```

```

        _VIDEO_PATCHES_PER_FRAME, free, total, _POSITION_EMBEDDING_SIZE
    )
    # 验证 one_hot 分配大小不超过预算 (free // 2)
    one_hot_bytes = chunk * _VIDEO_PATCHES_PER_FRAME * 2 \
        * _POSITION_EMBEDDING_SIZE * 8
    assert one_hot_bytes <= free // 2

def test_encoder_chunk_roomy_gpu_keeps_batching():
    # 宽松 GPU (A100 80GB, 60GB 空闲) 应保持大 batch
    chunk = _encoder_chunk(
        _VIDEO_PATCHES_PER_FRAME, 60 * GiB_bytes, 80 * GiB_bytes,
        _POSITION_EMBEDDING_SIZE
    )
    assert chunk > 8 # 至少处理 8 个帧

# 更多测试: 零 patches、零 position_embedding_size、零空闲内存 → 都返回 1
def test_encoder_chunk_zero_patches_is_safe():
    assert _encoder_chunk(0, 60*GiB_bytes, 80*GiB_bytes, _POSITION_EMBEDDING_SIZE) == 1

def test_encoder_chunk_zero_position_embedding_size_is_safe():
    assert _encoder_chunk(_VIDEO_PATCHES_PER_FRAME, 60*GiB_bytes, 80*GiB_bytes, 0) ==
    1

def test_encoder_chunk_no_free_memory_falls_back_to_one():
    assert _encoder_chunk(_VIDEO_PATCHES_PER_FRAME, 0, 22*GiB_bytes, _POSITION_
    EMBEDDING_SIZE) == 1

```

评论区精华

1. ZeroDivisionError 防护: gemini-code-assist 指出 position_embedding_size 可能为 0 导致 cost=0 和除零错误, 建议加 if cost > 0 检查。作者采纳并修复。
 2. 平台可移植性: gemini-code-assist 指出 torch.cuda.mem_get_info() 会破坏 CUDA 之外的后端支持, 建议使用 vLLM 平台抽象。作者表示已修复 (具体修改材料未展示, 后续应验证)。
 3. 测试文件位置: Isotr0py 建议将测试文件从 tests/models/multimodal/test_gemma4_mm.py 移至 processing/ 子目录以便 CPU CI 运行。作者照做。
 4. CI 效果确认: 作者在 PR 评论中提到该修复也解决了另一个 CI 构建 (Multi-Modal Models Extended Generation 3) 的失败, 并贴出了新的通过结果。
- ZeroDivisionError 防护 (correctness): 作者接受建议并在后续提交中修复, 最终代码中包含 return max(1, budget // cost) if cost > 0 else 1。
 - 平台可移植性: torch.cuda.mem_get_info() (design): 作者回复 'fixed', 但最终代码是否改用抽象层在材料中未明确展示。PR 最终被合并, 推测已处理。
 - 测试文件位置调整 (testing): 作者照做, 测试文件最终放在 processing/ 下。

风险与影响

- 风险:

1. 平台兼容性风险: 如果最终合并版本仍直接使用 `torch.cuda.mem_get_info()`, 则无法在 AMD ROCm、Intel XPU、CPU 等非 CUDA 后端运行, 需额外适配。虽经 review 提出但实际修复情况不明, 建议后续确认。
2. 成本模型假设: 假设 `F.one_hot` 是唯一 dominant 瞬时内存, 若未来模型变体引入其他大型临时张量, 预算可能仍不准确。当前成本公式 `patches * 4 * pos_emb_size * 8` 中的系数 4 和 8 凭经验设定, 缺乏自动化校准。
3. 低显存退化: 上限总显存 10% 可能限制 batching 吞吐, 但较原 5% 稍好。在空闲内存极低时回退到 1, 性能可能下降, 但保证了稳定性。
4. 缺少跨后端抽象: 即使修复后改用平台 API, 仍依赖平台层提供 `get_device_free_memory()`, 若未统一实现则存在缺陷。

- 影响:

1. 用户影响: 使用 Gemma4 多模态模型的用户在低显存 GPU (如 L4 22GB) 上运行 AWQ 等大模型时不再因视觉编码器 OOM 崩溃。高显存用户仍保持批处理加速效果 (如 A100 80GB 上 `chunk=92`) 。
2. 系统影响: CI 中 `test_awq_load[gemma4-moe-*`] 测试恢复通过, MM Extended Generation 3 等后续流水线也受益。
3. 团队影响: 将核心逻辑提取为纯静态函数并添加无需 GPU 的单元测试, 降低了后续维护门槛, 并可作为类似内存敏感算法的设计模式。 - 风险标记: 平台兼容性风险, 成本模型假设, 缺少跨后端抽象

关联脉络

- PR #43169 [Perf][Gemma4] Batch vision encoder calls for image and video processing: 前序 PR #43169 引入了视觉编码器批处理, 但内存预算模型不准确导致 OOM。本 PR 修复其回归并改进预算算法。