PR #47985 完整报告

vllm-project/vllm

[MRV2] Add encoder cache profiling implementation

合并时间：2026-07-21 11:18

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47985

执行摘要

- 一句话：MRV2 新增编码器缓存 profiling 防止 OOM
- 推荐动作：建议关注该 PR 的设计决策：将 processor cache 设为可选参数以避免 profiling 污染，以及 profile 完成后清理临时缓存。该模式适用于类似需要模拟推理但不影响状态的场景。后续建议补充测试验证 profile 结果准确性。

功能与动机

Issue 评论中指出：'This PR will help us to migrate to MRV2, or otherwise some of our VLM runs would OOM because MRV2 now doesn't take encoder cache memory usage into account and would over-allocate KV cache.' 因此该变更是为了在 MRV2 中准确估算 encoder 缓存内存，避免 KV cache 过度分配导致的 OOM。

实现拆解

1. 扩展 MultiModalBudget（vllm/multimodal/encoder_budget.py）：为 __init__ 增加 enable_cache: bool = True 参数，允许 profiling 时禁用 processor cache，避免缓存干扰；新增 get_dummy_encoder_profile_inputs 函数，利用 registry 生成指定 modality 的虚拟输入。
2. 新增 EncoderRunner.profile_encoder_cache（vllm/v1/worker/gpu/mm/encoder_runner.py）：接收虚拟输入和 budget，调用 execute_mm_encoder 进行一次虚拟 encoder 推理，将输出通过 encoder_cache.encoder_outputs.update 存入临时键。
3. 集成到 ModelRunner.profile_run（vllm/v1/worker/gpu/model_runner.py）：在原有的 profile_run 开头，若支持多模态且非递归调用，则构造 MultiModalBudget(enable_cache=False)，获取虚拟输入后调用 encoder_runner.profile_encoder_cache；profile 结束后调用 reset_encoder_cache 清理临时数据。
4. 注释清理（vllm/v1/worker/gpu/mm/encoder_cache.py）：将 reset_mm_cache 中的 TODO 注释改为 NOTE，说明 MRV2 的 encoder cache profiling 不再需要 MM cache。
5. 无测试配套：本次改动未包含新的单元测试或集成测试，依赖现有 CI 覆盖。

关键文件：

- vllm/v1/worker/gpu/mm/encoder_runner.py（模块 编码器运行；类别 source；类型 core-logic；符号 profile_encoder_cache）：新增 profile_encoder_cache 方法，是

encoder profiling 的核心执行逻辑。

- vllm/multimodal/encoder_budget.py（模块 预算计算；类别 source；类型 core-logic；符号 get_dummy_encoder_profile_inputs, MultiModalBudget.init）：新增 get_dummy_encoder_profile_inputs 函数，并修改 MultiModalBudget 支持可选缓存。
- vllm/v1/worker/gpu/model_runner.py（模块 模型运行；类别 source；类型 data-contract；符号 profile_run）：在 profile_run 入口集成 encoder profiling，是调用的主要入口。
- vllm/v1/worker/gpu/mm/encoder_cache.py（模块 编码器缓存；类别 source；类型 core-logic）：注释调整，说明 MRV2 profiling 不需要 MM cache。

关键符号：profile_encoder_cache, get_dummy_encoder_profile_inputs, MultiModalBudget.init, profile_run

关键源码片段

vllm/v1/worker/gpu/mm/encoder_runner.py

新增 profile_encoder_cache 方法，是 encoder profiling 的核心执行逻辑。

```
@torch.inference_mode()
def profile_encoder_cache(
    self,
    dummy_mm_inputs: list[tuple[str, MultiModalKwargsItem]],
    budget: MultiModalBudget,
) -> None:
    """Profile multimodal encoder and temporary encoder cache memory."""
    # 如果 encoder budget 非正，直接跳过
    if (encoder_budget := budget.get_encoder_budget()) <= 0:
        return

    # 如果是 embedding-only 模式（所有 modality limit=0），也跳过
    if not budget.mm_max_toks_per_item:
        logger.info(
            "Skipping encoder profiling for embedding-only mode "
            "(all modality limits=0 with enable_mm_embeds=True).",
        )
        return

    # 确保 dummy 输入已生成
    assert dummy_mm_inputs, "Dummy inputs should be generated for encoder profiling"
    dummy_modality = dummy_mm_inputs[0][0]
    max_mm_items_per_batch = len(dummy_mm_inputs)

    logger.info_once(
        "Encoder cache will be initialized with a budget of %s tokens, "
        "and profiled with %s %s items of the maximum feature size.",
        encoder_budget,
        max_mm_items_per_batch,
        dummy_modality,
    )
```

```
    # 执行一次虚拟 encoder 推理
    dummy_encoder_outputs = self.execute_mm_encoder(dummy_mm_inputs)

    # 校验输出数量和批次大小一致
    sanity_check_mm_encoder_outputs(
        dummy_encoder_outputs,
        expected_num_items=max_mm_items_per_batch,
    )
    # 将 encoder 输出存到 encoder_cache 中，用于内存计算
    self.encoder_cache.encoder_outputs.update(
        (f"tmp_{i}", output) for i, output in enumerate(dummy_encoder_outputs)
    )
```

vllm/multimodal/encoder_budget.py

新增 get_dummy_encoder_profile_inputs 函数，并修改 MultiModalBudget 支持可选缓存。

```
# 新增获取 dummy encoder profile 输入的函数
def get_dummy_encoder_profile_inputs(
    mm_registry: MultiModalRegistry,
    budget: MultiModalBudget,
) -> list[tuple[str, MultiModalKwargsItem]]:
    # 如果 encoder budget 非正或 mm_max_toks_per_item 为空（embedding-only），返回空
    if budget.get_encoder_budget() <= 0 or not budget.mm_max_toks_per_item:
        return []

    # 选取 token 消耗最多的 modality 作为代表
    modality = budget.get_modality_with_max_tokens()
    max_items_per_batch = budget.mm_max_items_per_batch[modality]
    # 通过 registry 生成一个 dummy 多模态输入
    dummy_mm_inputs = mm_registry.get_dummy_mm_inputs(
        budget.model_config,
        mm_counts={modality: 1},
        processor=budget.processor,
    )
    # 获取该 modality 的 dummy item
    dummy_mm_item = dummy_mm_inputs["mm_kwargs"][modality][0]
    assert dummy_mm_item is not None, "Dummy item should be generated"

    # 复制多份以匹配最大批次大小
    return [(modality, dummy_mm_item)] * max_items_per_batch
```

评论区精华

Review 中 gty111 提出设计问题：为什么要在 encoder_budget.py 中新建类而不直接复用 MultiModalBudget？Isotr0py 回应这是为了验证禁用 processor cache 是否影响 profile 结果，随后决定通过给 MultiModalBudget 增加 enable_cache 参数复用现有类，避免了代码重复。该讨论体现了设计上的权衡：profiling 时不需要 processor cache，但正常推理需要，通过参数化统一了代码路径。

- 复用 MultiModalBudget 的设计 (design): 采用参数化方式复用 MultiModalBudget，避免了代码冗余。

风险与影响

- 风险：
  1. 缺少测试覆盖：无新增测试文件，profile 逻辑的正确性依赖已有集成测试，可能遗漏边界条件（如 *embedding-only* 模式、skip_mm_profiling 配置）。
  2. profile 影响内存：profile_run 中额外分配了 *encoder* 输出 tensor（存入 *encoder_cache*），但随后被 *reset_encoder_cache* 清理，不会积累。
  3. 配置兼容：skip_mm_profiling 配置若被误用可能导致 profile 跳过，但已有条件检查。
  4. 多模态模型差异：不同模型的 *encoder* 行为不同，虚拟输入可能不充分代表实际内存（如动态分辨率），但作为初始估计足够。 - 影响：用户影响：启用 MRV2 的多模态推理用户将受益于更准确的 KV *cache* 分配，降低 OOM 风险。无其他 API 或行为变更。系统影响：启动时增加一次 *encoder* 推理开销（profile_run 阶段），但仅有一次，且对单次推理影响可忽略。团队协作：该 PR 为 MRV2 迁移扫清障碍，后续可基于此进一步优化 *encoder cache* 管理。
- 风险标记：缺少测试覆盖，核心路径变更

关联脉络

- 暂无明显关联 PR