

PR #48535 完整报告

vllm-project/vllm

[Front-end] [Messages] Populate `num\_cache\_creation\_tokens`

合并时间: 2026-07-19 01:04

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/48535>

执行摘要

- 一句话: 填充 Messages API 的缓存创建 token 数
- 推荐动作: 推荐精读 `KVCacheManager.estimate_cached_tokens` 和 `PrefillStats.finalize` 的设计, 它们展示了如何在避免昂贵全量匹配的前提下近似估计缓存创建 token 数。另外, `scheduler.py` 中 `prefill_stats` 获取时机的选择 (在块释放前) 值得注意, 体现了对资源生命周期的精细管理。

功能与动机

40912 引入了 `prompt_tokens_details` 支持, 但缺少区分缓存命中 (`cached_tokens`) 和缓存创建 (`created_cache_tokens`) 的能力。用户在使用 Anthropic Messages API 时需要准确的缓存创建 token 数, 以便正确计费和理解缓存行为。PR body 指出: "When `--enable-prompt-tokens-details` is toggled, compute the number of prompt tokens written to prefix cache and pass it through to OpenAI Chat Completions and eventually to Anthropic Messages API. Ensures `num_cache_creation_tokens` is populated."

实现拆解

实现过程包括以下步骤:

1. 补充 `PrefillStats` 模型: 在 `vllm/v1/metrics/stats.py` 的 `PrefillStats` 数据类中新增 `num_cache_creation_tokens: int` 字段, 并添加 `finalize(num_cached_tokens)` 方法, 该方法根据 `num_cached_tokens` (估计的请求缓存 token 数) 和已有的 `num_cached_tokens` (调度时确定的缓存 token 数) 计算出本次创建的新缓存 token 数。
2. 添加 `estimate_cached_tokens` 方法: 在 `vllm/v1/core/kv_cache_manager.py` 的 `KVCacheManager` 中新增 `estimate_cached_tokens(request)` 方法。它遍历每个 KV cache 组的块, 取每个组内 `block.block_hash_num_tokens` 的最大值作为该组的缓存 token 数, 然后对所有组取最小值作为整体估计。这种方法避免了昂贵的哈希全量匹配, 性

能较好。

3. 修改调度器输出逻辑：在 `vllm/v1/core/sched/scheduler.py` 的 `update_from_output` 中，当确定需要输出（有新 token、pooler 输出或停止）时，调用 `request.take_prefill_stats()` 获取 `PrefillStats`，然后调用其 `finalize` 方法，传入 `self.kv_cache_manager.estimate_cached_tokens(request)`，从而计算出 `num_cache_creation_tokens`。这一步在块被释放前执行，确保能访问到缓存状态。
4. 改造 Anthropic 使用量计算：在 `vllm/entrypoints/anthropic/serving.py` 中重写了 `_build_anthropic_usage` 函数，直接从 `UsageInfo.prompt_tokens_details` 中提取 `cached_tokens`（缓存读取）和 `created_cache_tokens`（缓存创建），并据此调整 `input_tokens` 的计算公式（`input_tokens = prompt_tokens - cache_read - cache_creation`）。同时更新了 `messages_full_converter` 的调用方式，简化了参数传递。
5. 更新测试覆盖：修改了 `tests/entrypoints/anthropic/test_anthropic_messages_conversion.py` 中的 `TestBuildAnthropicUsage` 测试类，添加了 `test_zero_cached_tokens`、`test_all_tokens_cached` 等用例，并移除了不再需要的 `TestGetCachedTokens` 辅助测试类。新增了 `tests/entrypoints/anthropic/test_messages.py` 中的 `test_anthropic_streaming_cache_usage` 集成测试，通过发送两次请求（冷启动和预热）验证 `cache_read_input_tokens` 和 `cache_creation_input_tokens` 的正确性。

关键文件：

- `vllm/entrypoints/anthropic/serving.py`（模块 Anthropic 入口；类别 source；类型 core-logic；符号 `_build_anthropic_usage`）：这是 Anthropic Messages API 入口，核心函数 `_build_anthropic_usage` 被完全重写，现在直接从 `UsageInfo.prompt_tokens_details` 提取 `cached_tokens` 和 `created_cache_tokens`，实现了 `num_cache_creation_tokens` 的填充。
- `vllm/v1/core/kv_cache_manager.py`（模块 缓存层；类别 source；类型 core-logic；符号 `estimate_cached_tokens`）：新增 `estimate_cached_tokens` 方法，用于估计请求已缓存的 token 数，该方法被调度器调用来计算 `num_cache_creation_tokens`。
- `vllm/v1/core/sched/scheduler.py`（模块 调度器；类别 source；类型 core-logic；符号 `update_from_output`）：调度器修改了输出逻辑，在 `update_from_output` 中调用 `estimate_cached_tokens` 并完成 `PrefillStats.finalize`，是数据流的枢纽。
- `vllm/v1/metrics/stats.py`（模块 统计；类别 source；类型 core-logic；符号 `PrefillStats`，`finalize`）：`PrefillStats` 新增 `num_cache_creation_tokens` 字段和 `finalize` 方法，用于记录缓存创建 token 数。
- `tests/entrypoints/anthropic/test_anthropic_messages_conversion.py`（模块 测试；类别 test；类型 test-coverage；符号 `TestBuildAnthropicUsage`，`test_cache_hit`，`test_zero_cached_tokens`，`test_all_tokens_cached`）：测试缓存使用量计算逻辑，新增多个测试用例验证 `_build_anthropic_usage` 在不同缓存情况下的正确性。

关键符号：`_build_anthropic_usage`，`KVCacheManager.estimate_cached_tokens`，`PrefillStats.finalize`，`Scheduler.update_from_output`

关键源码片段

vllm/entrypoints/anthropic/serving.py

这是 Anthropic Messages API 入口，核心函数 `_build_anthropic_usage` 被完全重写，现在直接从 `UsageInfo.prompt_tokens_details` 提取 `cached_tokens` 和 `created_cache_tokens`，实现了 `num_cache_creation_tokens` 的填充。

```
def _build_anthropic_usage(
    usage: UsageInfo | None,
) -> AnthropicUsage:
    """Build an AnthropicUsage from UsageInfo.

    Anthropic defines total_input == input_tokens + cache_read + cache_creation.
    vLLM's prompt_tokens is the total, so input_tokens = prompt_tokens - cache_read - cache_
    creation.
    Cache fields are taken from UsageInfo.prompt_tokens_details.
    When cache info is absent, cache fields are left unset so exclude_unset=True serialization
    omits them entirely.
    completion_tokens may be None on intermediate stream chunks; we coerce to 0.
    """

    kwargs = {}
    if usage is None:
        kwargs['input_tokens'] = 0
        kwargs['output_tokens'] = 0
    else:
        kwargs['output_tokens'] = usage.completion_tokens or 0
        input_tokens = usage.prompt_tokens

        # 从 prompt_tokens_details 中提取缓存命中与缓存创建信息
        if (details := usage.prompt_tokens_details) is not None:
            if (cache_read := details.cached_tokens) is not None:
                # 缓存读取 token 数，从 input_tokens 中扣除
                input_tokens -= cache_read
                kwargs['cache_read_input_tokens'] = cache_read

            if (cache_creation := details.created_cache_tokens) is not None:
                # 缓存创建 token 数，从 input_tokens 中扣除
                input_tokens -= cache_creation
                kwargs['cache_creation_input_tokens'] = cache_creation

        # 最终 input_tokens 不能为负数
        kwargs['input_tokens'] = max(0, input_tokens)
    return AnthropicUsage(**kwargs)
```

vllm/v1/core/kv_cache_manager.py

新增 `estimate_cached_tokens` 方法，用于估计请求已缓存的 token 数，该方法被调度器调用来计算 `num_cache_creation_tokens`。

```
def estimate_cached_tokens(self, request: Request) -> int:
    """Estimate the number of tokens cached by the request."""
```

```

cached_tokens: int | None = None
for group, blocks in zip(
    self.kv_cache_config.kv_cache_groups,
    self.get_blocks(request.request_id).blocks,
):
    if isinstance(
        group.kv_cache_spec,
        (CrossAttentionSpec, EncoderOnlyAttentionSpec),
    ):
        # Cross-attention 和 encoder-only 组不参与前缀缓存
        continue

    # 取当前组内所有块中最大的 block_hash_num_tokens 作为该组的缓存 token 数
    group_cached_tokens = 0
    for block in blocks:
        group_cached_tokens = max(
            group_cached_tokens,
            block.block_hash_num_tokens or 0,
        )

    # 对每个组取最小值为总体估计，确保不会高估（最弱的组决定整体缓存程度）
    cached_tokens = (
        group_cached_tokens
        if cached_tokens is None
        else min(cached_tokens, group_cached_tokens)
    )

return cached_tokens or 0

```

评论区精华

Review 中最核心的讨论围绕性能展开：

- tlrnchlsmth指出在调度器里调用 `find_longest_cache_hit` 对每个请求开销太大，建议在 `cache_blocks()` 中维护高水位标记，或仅在启用 `--enable-prompt-tokens-details` 时调用。
- yzong-rh回应说已改用 `KVCacheManager.estimate_cached_tokens` 方法，通过遍历块并取 `block.block_hash_num_tokens` 最大值来估计，避免了全量哈希，性能可接受。该实现最终被采纳。

此外，作者还澄清了在 `update_from_output` 中提前获取 `prefill_stats` 的原因：确保在 `KVCacheBlocks` 被释放前还能访问它们的缓存状态。

- `estimate_cached_tokens` 性能影响 (performance): 作者 yzong-rh 回应改用 `KVCacheManager.estimate_cached_tokens` 方法，通过遍历块并取 `block.block_hash_num_tokens` 最大值来估计，避免了全量哈希，性能可接受。最终采纳该方案。

风险与影响

- 风险:

1. 性能风险: `estimate_cached_tokens` 遍历所有块, 对于长 context 有 $O(N)$ 开销。但已通过仅取最大值 (而非全表哈希) 控制代价。另可通过 `--enable-prompt-tokens-details` 开关控制是否调用该路径。
2. 正确性风险: `estimate_cached_tokens` 使用 `block.block_hash_num_tokens` 估计, 该值反映块内前缀缓存匹配的 token 数, 但取最大值再取最小值可能低估或多估 (但实际上保守估计了最弱的组)。对于跨组非均匀缓存的情况, 可能得到略低的缓存 token 数, 但不会导致逻辑错误。
3. 兼容性风险: 新增的 `num_cache_creation_tokens` 字段在 Anthropic 响应中为可选, `--enable-prompt-tokens-details` 关闭时不填充, 不影响旧行为。OpenAI 响应通过 `PromptTokenUsageInfo.created_cache_tokens` 传递, 同样可选。
4. 核心路径变更: 调度器输出逻辑和缓存管理器新增方法属于核心路径, 需仔细回归测试。

- 影响:

- 用户影响: 使用 Anthropic Messages API 的用户现在可以获取 `cache_creation_input_tokens` 和 `cache_read_input_tokens`, 使缓存使用统计更精细。OpenAI Chat Completions 用户也可通过 `prompt_tokens_details.created_cache_tokens` 获取缓存创建信息。所有变更仅在启用 `--enable-prompt-tokens-details` 时生效, 向后兼容。
- 系统影响: 调度器在每次输出时调用 `estimate_cached_tokens`, 增加小幅计算开销。内部统计更准确, 有助于缓存策略优化。
- 团队影响: 维护成本较低, 新增方法有明确用途, 且测试覆盖充分。
- 风险标记: 性能风险: `estimate_cached_tokens` 遍历所有块, 核心路径变更: 调度器输出逻辑修改, 兼容性: 新增字段可选, 向后兼容

关联脉络

- PR #40912 Add support for `prompt_tokens_details` in Messages API: 本 PR 是 #40912 的跟进, 在已支持的 `prompt_tokens_details` 基础上填充 `cache_creation` 字段, 使缓存使用统计更精确。