

PR #45458 完整报告

vllm-project/vllm

[Feature][Frontend] Report multimodal token counts in usage.prompt_tokens_details

合并时间: 2026-06-15 13:20

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/45458>

执行摘要

- 一句话: 在 `usage.prompt_tokens_details` 中报告多模态 token 计数
- 推荐动作: 值得精读。PR 展示了如何以最小侵入性扩展 OpenAI 兼容 API, 重点关注 `_make_prompt_tokens_details` 中 `if not enable_prompt_tokens_details / if num_cached_tokens is None and not mm_token_counts` 的短路逻辑, 以及 `cast` 安全地处理类型。设计上使用字典替代字段枚举体现了良好的可扩展性。

功能与动机

PR body 指出: `'usage.prompt_tokens already includes the image/audio/video placeholder tokens, but clients cannot tell how many tokens each modality contributed.'` 用户需要按模态拆分的 token 计数, 以进行精细化计费、监控或优化。

实现拆解

1. 数据结构扩展: 在 `vllm/entrypoints/openai/engine/protocol.py` 的 `PromptTokenUsageInfo` 类中添加 `multimodal_tokens: dict[str, int] | None` 字段, 并附 docstring 说明含义。
2. 辅助函数: 在 `vllm/entrypoints/openai/chat_completion/serving.py` 中新增 `_get_mm_token_counts(engine_input)`, 从请求的 `mm_placeholders` (`MultiModalPlaceholders`) 中按模态名汇总所有 `PlaceholderRange.length` 之和, 返回字典; 新增 `_make_prompt_tokens_details(enable_prompt_tokens_details, num_cached_tokens, mm_token_counts)`, 根据开关和是否有数据构造 `PromptTokenUsageInfo` 实例, 当功能关闭或无可报数据时返回 `None`。
3. 集成到请求处理流程: 在 `_create_chat_completion` 中, 对每个 `engine_input` 调用 `_get_mm_token_counts`, 并将结果传给流式与非流式生成器。生成器在构造最终 `usage` 时, 调用 `_make_prompt_tokens_details` 合并 `cached_tokens` 与 `multimodal_tokens`。
4. 单元测试: 在 `tests/entrypoints/openai/chat_completion/test_serving_chat.py` 中添加 `test_mm_prompt_tokens_details`, 覆盖文本请求无多模态、多模态多范围求和、功能关闭 / 无数据返回 `None`、以及零 `cached_tokens` 仍报告等场景。

关键文件:

- `vllm/entrypoints/openai/chat_completion/serving.py` (模块 前端; 类别 `source`; 类型 `core-logic`; 符号 `_get_mm_token_counts`, `_make_prompt_tokens_details`): 核心实现文

件，新增 `_get_mm_token_counts` 和 `_make_prompt_tokens_details` 两个辅助函数，并修改 `_create_chat_completion` 以传递多模态 token 计数给流式和非流式生成器。

- `vllm/entrypoints/openai/engine/protocol.py`（模块 协议；类别 source；类型 core-logic）：数据结构定义文件，`PromptTokenUsageInfo` 新增 `multimodal_tokens` 字段，是功能的数据契约基础。
- `tests/entrypoints/openai/chat_completion/test_serving_chat.py`（模块 测试；类别 test；类型 test-coverage；符号 `test_mm_prompt_tokens_details`）：单元测试，覆盖 `_get_mm_token_counts` 和 `_make_prompt_tokens_details` 的主要逻辑分支，包括多模态多范围求和、功能关闭、无数据等场景。

关键符号： `_get_mm_token_counts`, `_make_prompt_tokens_details`

关键源码片段

`vllm/entrypoints/openai/chat_completion/serving.py`

核心实现文件，新增 `_get_mm_token_counts` 和 `_make_prompt_tokens_details` 两个辅助函数，并修改 `_create_chat_completion` 以传递多模态 token 计数给流式和非流式生成器。

```
def _get_mm_token_counts(engine_input: EngineInput) -> dict[str, int]:
    """Sum per-modality placeholder tokens from ``mm_placeholders``.

    Keyed by modality name; ``PlaceholderRange.length`` is the placeholder's
    prompt token span, so each sum matches the placeholder tokens already
    counted in ``usage.prompt_tokens``.
    """
    # `cast` 用于让类型检查器接受动态 key 读取
    mm_placeholders = cast(
        "MultiModalPlaceholders | None", engine_input.get("mm_placeholders")
    )
    # 遍历所有模态，汇总每模态所有占位符的 length
    return {
        modality: sum(p.length for p in ranges)
        for modality, ranges in (mm_placeholders or {}).items()
        if ranges # 忽略空列表
    }

def _make_prompt_tokens_details(
    enable_prompt_tokens_details: bool,
    num_cached_tokens: int | None,
    mm_token_counts: dict[str, int] | None,
) -> PromptTokenUsageInfo | None:
    """Build ``prompt_tokens_details`` from cached + multimodal token counts."""
    # 功能未开启 → 不返回
    if not enable_prompt_tokens_details:
        return None
    # 既没有缓存信息也没有多模态 token → 不返回
    if num_cached_tokens is None and not mm_token_counts:
```

```
    return None
# 合并构造, multimodal_tokens 为 None 时字典会被 Pydantic 序列化为 null
return PromptTokenUsageInfo(
    cached_tokens=num_cached_tokens,
    multimodal_tokens=mm_token_counts or None,
)
```

评论区精华

核心讨论围绕数据结构的可扩展性。审核者 DarkLight1337 建议使用字典而非硬编码字段（`image_tokens`, `audio_tokens`, `video_tokens`），作者 Sunt-ing 接受并改为 `multimodal_tokens: dict[str, int]`，使新增模态无需改动协议。后续还涉及返回类型简化（直接返回字典而非 `Optional[dict]`）和 docstring 格式（MkDocs 用单反引号）。所有讨论均已 resolved。

- 字段设计：硬编码 vs 字典 (design): 采用字典方案，未来新增模态无需修改协议。
- 辅助函数返回类型简化 (design): 确保调用侧 `_make_prompt_tokens_details` 已正确处理空字典。
- docstring 格式 (style): 统一使用单反引号包裹。

风险与影响

- 风险：低风险。新字段 `multimodal_tokens` 为 `Optional`，不存在时客户端可安全忽略。功能受 `--enable-prompt-tokens-details` 开关门控，默认不启用。实现仅从已有的 `mm_placeholders` 汇总长度，不涉及推理路径修改。主要风险在于如果 `PlaceholderRange.length` 定义或计算方式将来改变，需要同步更新，但当前逻辑与 `prompt_tokens` 计数一致，已通过测试覆盖基本场景。
- 影响：对用户：使用 Chat Completions API 且启用 token details 的请求，可在响应中得到 `multimodal_tokens` 明细，便于调优和理解成本。对系统：新增代码量小（+93/-14 行），无性能测试表明有影响，仅增加一次字典汇总操作（ $O(\text{number of placeholders})$ ）。对团队：设计上避免了硬编码，未来支持新模态（如 depth map）只需确保其 `PlaceholderRange` 被正确填充即可，无需修改协议定义。文本请求行为完全不变。
- 风险标记：依赖 `mm_placeholders` 正确性，功能默认关闭，无默认行为变化

关联脉络

- PR #27122 [SGLang] Report multimodal token counts in `usage.prompt_tokens_details`: 本 PR 直接参考并实现了 [sgl-project/sglang#27122](#) 提出的功能，README 和 PR body 中明确引用。