PR #44887 完整报告

vllm-project/vllm

[Rust Frontend] Populate `cached_token_count` in responses

合并时间：2026-06-11 11:50

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44887

报告：PR #44887 - [Rust Frontend] Populate cached_token_count in responses

1. 执行摘要

该 PR 填补了 Rust 前端在 OpenAI 和 inference API 响应中缺失 prompt_tokens_details.cached_tokens 字段的功能空白。通过提取 TokenUsage 结构体、引入 ApiServerOptions 配置聚合、并贯通 prefill_stats 到 Usage 的数据流，实现了与 Python 前端的 feature parity。同时新增 --enable-prompt-tokens-details 命令行标志，保持 opt-in 行为，避免默认暴露缓存命中信息。

2. 功能与动机

Issue #44824 指出 Rust 前端从未填充 prompt_tokens_details.cached_tokens 字段，尽管该数据已到达前端并被 Prometheus 指标消费。Python 前端在 --enable-prompt-tokens-details 下已支持，因此这是 Rust 端的功能缺口。PR 目标：让所有 Rust 前端路径（/v1/completions、/v1/chat/completions、/inference/v1/generate）在开启标志后能返回缓存令牌数，与 Python 前端行为一致。

3. 实现拆解

1. 提取 TokenUsage 结构体（rust/src/llm/src/output.rs）：将分散的 prompt_token_count、output_token_count、cached_token_count 封装为统一结构体，在各层传递，减少参数列表长度。
2. 修改 Usage 构造方法（rust/src/server/src/routes/openai/utils/types.rs）：from_counts 新增 cached_tokens 参数，仅在值 >0 时序列化；新增 from_token_usage 便捷方法，根据 enable_prompt_tokens_details 标志决定是否传入缓存数。
3. 引入 ApiServerOptions 配置聚合（rust/src/server/src/state.rs）：在 AppState 中用单一结构体替代三个独立布尔字段（enable_log_requests、enable_request_id_headers、enable_prompt_tokens_details），降低状态管理复杂度。
4. 添加命令行标志（rust/src/cmd/src/cli.rs 及测试）：为 serve 和 frontend 子命令新增 --enable-prompt-tokens-details，并在 to_frontend_config 中映射到 ApiServerOptions；测试验证标志传递和 args-json 配置。
5. 在各路由入口启用（chat_completions.rs、completions.rs、generate.rs）：将函数签名中的 log_request: bool 替换为 ApiServerOptions 结构体，调用

Usage::from_token_usage 替代 from_counts，从而按标志填充。

配套变更：所有 chat 和 text 模块的 output 路径（如 default/tool.rs、structured.rs、decoded.rs）也同步调整以返回 TokenUsage 结构体。

rust/src/server/src/routes/openai/utils/types.rs

核心变更：修改 Usage 结构体和 from_counts/from_token_usage 方法，新增 cached_tokens 支持，并添加单元测试。

```
/// 从各部分计数构造 Usage，支持可选缓存令牌数
pub fn from_counts(
    prompt_tokens: usize,
    completion_tokens: usize,
    cached_tokens: Option<usize>, // 缓存令牌数，None 表示不报告
) -> Self {
    Self {
        prompt_tokens,
        total_tokens: prompt_tokens + completion_tokens,
        completion_tokens: Some(completion_tokens),
        prompt_tokens_details: cached_tokens
            .filter(|&c| c > 0) // 仅在缓存数 > 0 时包含
            .map(|c| PromptTokenUsageInfo { cached_tokens: c }),
    }
}

/// 从全量 TokenUsage 构造 Usage，并根据标志决定是否暴露缓存详情
pub fn from_token_usage(usage: TokenUsage, enable_prompt_tokens_details: bool) -> Self {
    Self::from_counts(
        usage.prompt_token_count,
        usage.output_token_count,
        enable_prompt_tokens_details.then_some(usage.cached_token_count),
    )
}

#[cfg(test)]
mod usage_tests {
    use vllm_llm::TokenUsage;
    use super::Usage;

    #[test]
    fn token_usage_hides_prompt_token_details_by_default() {
        let usage = Usage::from_token_usage(
            TokenUsage { prompt_token_count: 5, output_token_count: 2, cached_token_count: 3 },
            false,
        );
        assert!(usage.prompt_tokens_details.is_none());
    }

    #[test]
```

```
    fn token_usage_includes_prompt_token_details_when_enabled() {
        let usage = Usage::from_token_usage(
            TokenUsage { prompt_token_count: 5, output_token_count: 2, cached_token_count: 3 },
            true,
        );
        assert_eq!(usage.prompt_tokens_details.unwrap().cached_tokens, 3);
    }
}
```

rust/src/server/src/state.rs

用 ApiServerOptions 结构体替代多个独立布尔字段，统一管理 API server 行为选项。

```
/// HTTP/API-server behavior switches.
pub api_server_options: ApiServerOptions, // 聚合所有 API server 选项

impl AppState {
    pub fn new(served_model_names: Vec<String>, chat: ChatLlm) -> Self {
        Self {
            served_model_names,
            chat,
            api_server_options: ApiServerOptions::default(), // 默认关闭
            server_info: None,
            api_key_hashes: Vec::new(),
            server_load: AtomicU64::new(0),
            lora_manager: LoraManager::new(),
        }
    }

    /// 统一设置 API server 行为选项
    pub fn with_api_server_options(mut self, options: ApiServerOptions) -> Self {
        self.api_server_options = options;
        self
    }
}
```

rust/src/server/src/routes/openai/chat_completions.rs

路由入口修改：从 state 获取 api_server_options 并传递给收集和流式函数，用于控制是否暴露缓存详情。

```
// 从状态获取选项，替代之前的单个 log_request 布尔值
let api_server_options = state.api_server_options;

// 在收集路径中传递
let response = collect_chat_completion(
    chat_stream,
    prepared.request_id,
    prepared.response_model,
    created,
    api_server_options, // 传入完整选项
```

```
        prepared.options,
    ).await?;

    // 在流式路径中传递
    let chunk_stream = chat_completion_chunk_stream(
        chat_stream,
        prepared.request_id,
        prepared.response_model,
        created,
        api_server_options,
        prepared.options,
    );
```

5. 评论区精华

- njhill建议默认总是填充 cached_token_count，使 --enable-prompt-tokens-details 成为空操作，认为没必要用 flag 控制。
- BugenZhao回应引用 Python 侧 PR #10174 的讨论，指出某些提供商可能不希望暴露缓存令牌详情，因此保留 flag 并保持 opt-in。最终决定保留 flag。

6. 风险与影响

- 功能默认关闭：用户需要显式指定 --enable-prompt-tokens-details 才能看到 cached_tokens，对现有行为无影响。
- 数据暴露风险：启用后响应中包含缓存令牌数，可能被用于推断系统负载或缓存命中率，但属于新字段且需主动开启。
- 兼容性：该字段已是 OpenAI 标准，Python 前端已有，下游解析无影响。
- 性能：仅响应序列化增加少量计算，无性能风险。

7. 关联脉络

该 PR 是 Rust 前端功能 parity 的一部分（关联 Issue #44280）。它基于 #44884（未在历史列表），并直接关闭 #44824。与近期历史 PR 无直接重叠，但延续了 Rust 前端持续对齐 Python 前端的趋势。