PR #43965 完整报告

vllm-project/vllm

[Rust Frontend] Support continuous_usage_stats stream option

合并时间：2026-06-12 01:51

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43965

执行摘要

该 PR 为 Rust 前端的 chat completions 和 completions 端点增加对 OpenAI stream_options.continuous_usage_stats 的支持，使得每个流式分块携带累计 token 用量。通过引入 ContinuousUsage 结构体和 yield_chunk! 宏，在请求转换和流式响应生成中处理该选项，并添加了充分的测试覆盖。变更聚焦于 per-request 选项，放弃了 CLI 默认值，设计简洁合理。

功能与动机

Rust 前端已支持 include_usage 以在流结束时返回一个用量分块，但缺少 continuous_usage_stats 选项让每个分块都携带累计用量。这导致与 Python vLLM 行为不一致。PR body 明确指出："The Rust routes already supported stream_options.include_usage, but still rejected continuous_usage_stats." 通过此 PR，用户可以在请求中设置 continuous_usage_stats: true（同时需要 include_usage: true），获得与 Python 端一致的流式用量统计。

实现拆解

1. 新增 ContinuousUsage 工具结构体：在 rust/src/server/src/routes/openai/utils/usage.rs 中创建 ContinuousUsage，提供 set_prompt_tokens、add_output_tokens、set_final_counts 和 to_usage 方法，用于在流式响应过程中累计并输出 token 用量。该结构体只计数，最终用法块仍由权威的 TokenUsage 生成。
2. 请求转换层增加字段：在 completions/convert.rs 和 chat_completions/convert.rs 的 ResponseOptions 中加入 include_continuous_usage 字段。在对应的 prepare_*_request 函数中，仅当 include_usage 为 true 且请求中的 continuous_usage_stats 为 true 时，才将该字段设置为 true。这确保了连续用法只能在用户要求最终用法时启用。
3. 流式响应生成集成：在 completions.rs 的 completion_chunk_stream 和 chat_completions.rs 的 chat_completion_chunk_stream 中，引入 ContinuousUsage 实例。在 Start 事件时记录 prompt tokens，在文本增量事件时累加 output tokens。定义 yield_chunk! 宏，在每次 yield 前判断 include_continuous_usage，若为真则附加当前累计计数。同时，在 Finish 事件中使用 set_final_counts 更新最终计数，并 yield 最终用法块（保持原有行为）。

4. 移除 validate.rs 中的拒绝检查：之前 validate.rs 中直接拒绝 continuous_usage_stats 选项，现已移除，使该选项可以正常传递。
5. 测试覆盖：在 tests.rs 中添加两个集成测试，分别验证 chat 和 completions 端点开启 continuous_usage_stats 后，每个流式分块的 usage 字段都存在，且最终无 choice 的用法块包含正确总数。在 completions/convert.rs 和 chat_completions/convert.rs 的模块测试中添加单元测试，验证选项映射和门控逻辑。

关键源码片段

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

流式响应生成核心文件，引入 ContinuousUsage 和 yield_chunk! 宏，在每个流式分块中附加用法数据，是功能实现的关键。

```
use crate::routes::openai::utils::usage::ContinuousUsage;

// 在 chat_completion_chunk_stream 函数中：
let mut continuous_usage = ContinuousUsage::default();

// 宏：在 yield 前根据 include_continuous_usage 追加 usage
macro_rules! yield_chunk {
    ($chunk:expr) => {{
        let mut chunk = $chunk;
        if include_continuous_usage {
            chunk.usage = Some(continuous_usage.to_usage());
        }
        y.yield_ok(chunk).await;
    }};
}

// 处理 Start 事件：记录 prompt token，yield 第一块
Ok(ChatEvent::Start { prompt_token_ids, .. }) => {
    continuous_usage.set_prompt_tokens(prompt_token_ids.len());
    let mut chunk = start_chunk(&request_id, &response_model, created);
    if return_token_ids {
        chunk.prompt_token_ids = Some(prompt_token_ids.to_vec());
    }
    yield_chunk!(chunk); // 使用宏，可能附加连续用法
    // 当 echo=true 时，发射最后助手消息内容作为 delta 块
    if let Some(echo_text) = &echo {
        yield_chunk!(block_delta_chunk(
            &request_id,
            &response_model,
            created,
            AssistantBlockKind::Text,
            echo_text.clone(),
        ));
    }
}
```

```
// 处理 BlockDelta 事件：累加 output token 后 yield
Ok(ChatEvent::BlockDelta { kind, delta, .. }) => {
    // ... 构造 chunk ...
    let delta_token_count = /* 计算 delta token 数 */;
    continuous_usage.add_output_tokens(delta_token_count);
    yield_chunk!(chunk);
}
```

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

新增的 ContinuousUsage 结构体，是连续用法统计的核心工具，提供累计计数和转换方法。

```
// 累计 prompt 与 output token 计数，供流式分块使用
#[derive(Debug, Clone, Default)]
pub(crate) struct ContinuousUsage {
    prompt_tokens: usize,
    output_tokens: usize,
}

impl ContinuousUsage {
    /// 记录 stream 开始时报告的 prompt token 计数
    pub(crate) fn set_prompt_tokens(&mut self, prompt_tokens: usize) {
        self.prompt_tokens = prompt_tokens;
    }

    /// 累加新解码的 output token 到运行 completion 计数
    pub(crate) fn add_output_tokens(&mut self, output_tokens: usize) {
        // 使用 saturating_add 防止溢出
        self.output_tokens = self.output_tokens.saturating_add(output_tokens);
    }

    /// 用生成结束时报告的最终计数替换运行计数
    pub(crate) fn set_final_counts(&mut self, prompt_tokens: usize, output_tokens: usize) {
        self.prompt_tokens = prompt_tokens;
        self.output_tokens = output_tokens;
    }

    /// 构建流式用法快照，不含 prompt cache 细节
    pub(crate) fn to_usage(&self) -> Usage {
        Usage::from_counts(self.prompt_tokens, self.output_tokens, None)
    }
}
```

评论区精华

- BugenZhao：建议先只支持 per-request 选项，CLI 默认值并非必需，且会增加维护负担。作者采纳并移除了 CLI 相关代码，使 PR 聚焦于请求选项。
- BugenZhao：指出每个 yield 前调用 maybe_attach_usage 易出错，建议抽象为 stream adaptor。最终采用简单的 yield_chunk! 宏，在保持可读性的同时减少了重复。

风险与影响

- 风险：流式响应路径的核心分支被修改，可能影响常规请求。但通过保持 include_continuous_usage 仅在选项开启时起作用，且测试覆盖了默认情况，风险较低。ContinuousUsage 不包含 prompt cache 细节，若未来需要扩展结构即可。
- 影响：用户可以通过请求选项开启连续用法，与 Python 行为一致。性能开销极低，仅增加一次字段赋值。团队代码结构清晰，宏方式易于维护。

关联脉络

该 PR 是 Rust 前端持续增强的一部分，之前的 #42331 和 #45030 完善了指标与监控。此 PR 进一步丰富了 Rust 前端的 OpenAI 兼容性，使得流式用法统计与 Python 后端对齐。后续可能将类似的 per-request 选项（如 return_tokens_as_token_ids）也按相同模式实现。