

PR #44760 完整报告

vllm-project/vllm

[Rust Frontend] Support `parallel\_tool\_calls = false`

合并时间: 2026-06-15 15:50

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/44760>

执行摘要

Rust 前端新增 `parallel_tool_calls` 参数支持。当设置为 `false` 时，流式响应仅保留第一个工具调用，后续工具调用被静默抑制。变更集中在 `structured.rs` 的状态机中，路由层只负责传递参数，默认值为 `true` 保持向后兼容。该 PR 是 Rust 前端功能对等性 Roadmap (Issue #44280) 的一部分。

功能与动机

作为 Rust 前端功能对等性的一部分，支持 OpenAI 的 `parallel_tool_calls` 参数是重要的一环。部分用户场景（如工具执行顺序敏感或下游系统限制）需要禁用并行工具调用。Python 前端已支持，Rust 前端需要补齐该功能。

实现拆解

1. 数据模型扩展：在 `ChatRequest` (`rust/src/chat/src/request.rs`) 中新增 `parallel_tool_calls: bool` 字段。
2. 请求映射：在 `convert.rs` 的 `prepare_chat_request` 中通过 `request.parallel_tool_calls.unwrap_or(true)` 映射，默认值为 `true`。
3. 移除拒绝校验：删除 `validate.rs` 中对 `parallel_tool_calls` 的拒绝逻辑。
4. 核心抑制逻辑：在 `structured.rs` 的 `StructuredEventState` 中加入 `parallel_tool_calls` 和 `suppressing_tool_call` 标志。在 `start_tool_call` 中，当 `!parallel_tool_calls && index >= 1` 时设置抑制标志并跳过事件；`push_tool_call_arguments` 和 `close_open_tool_call` 在抑制状态下也静默跳过。
5. 输出处理器传递：`DefaultChatOutputProcessor` 和 `HarmonyChatOutputProcessor` 的 `process` 方法将 `parallel_tool_calls` 传递给 `structured_chat_event_stream`。
6. 测试配套：在 `convert.rs` 新增两个单元测试，验证映射和默认值行为；现有测试传递 `true` 保持原行为。

`rust/src/chat/src/output/structured.rs`

实现了核心抑制逻辑，新增 `parallel_tool_calls` 和 `suppressing_tool_call` 状态，控制流式响应中工具调用的过滤。

```
/// 结构化事件流的组装状态机。
struct StructuredEventState {
    message: AssistantMessage,
    open_text_block: Option<OpenTextBlock>,
```

```

open_tool_call: Option<OpenToolCall>,
next_tool_call_index: usize,
/// 是否允许多个工具调用并行（即是否暴露索引 > 0 的工具调用）。
parallel_tool_calls: bool,
/// 是否正在抑制当前工具调用（当 parallel_tool_calls = false 且索引 >= 1 时为 true）。
suppressing_tool_call: bool,
}

impl StructuredEventState {
/// 创建新的状态实例；parallel_tool_calls 决定是否抑制后续工具调用。
fn new(parallel_tool_calls: bool) -> Self {
    Self {
        message: AssistantMessage::default(),
        open_text_block: None,
        open_tool_call: None,
        next_tool_call_index: 0,
        parallel_tool_calls,
        suppressing_tool_call: false,
    }
}

/// 开始一个新的工具调用。如果 parallel_tool_calls = false 且索引 >= 1,
/// 则设置 suppressing_tool_call 为 true，并返回空事件列表（不生成 ToolCallStart 事件）。
fn start_tool_call(&mut self, id: String, name: String) -> Result<Vec<ChatEvent>> {
    let mut events = Vec::new();
    self.close_open_text_block(&mut events);
    self.close_open_tool_call(&mut events);

    let index = self.next_tool_call_index;
    self.next_tool_call_index += 1;
    if !self.parallel_tool_calls && index >= 1 {
        self.suppressing_tool_call = true;
        return Ok(events);
    }

    self.suppressing_tool_call = false;
    self.open_tool_call = Some(OpenToolCall {
        index,
        id: id.clone(),
        name: name.clone(),
        arguments: String::new(),
    });
    events.push(ChatEvent::ToolCallStart { index, id, name });
    Ok(events)
}

/// 添加工具调用参数增量。如果正在抑制，则直接返回空事件列表。
fn push_tool_call_arguments(&mut self, delta: String) -> Result<Vec<ChatEvent>> {
    if self.suppressing_tool_call {

```

```

        return Ok(Vec::new());
    }

    let mut events = Vec::new();
    let Some(open_tool_call) = self.open_tool_call.as_mut() else {
        return Err(Error::ToolCallStreamInvariant {
            message: "received tool-call arguments delta without an open tool call".to_string(),
        });
    };
    open_tool_call.arguments.push_str(&delta);
    events.push(ChatEvent::ToolCallArgumentsDelta {
        index: open_tool_call.index,
        delta,
    });
    Ok(events)
}

/// 关闭当前工具调用。如果正在抑制，则清除抑制标志但不生成事件。
fn close_open_tool_call(&mut self, events: &mut Vec<ChatEvent>) {
    if self.suppressing_tool_call {
        self.suppressing_tool_call = false;
        return;
    }
    let Some(open_tool_call) = self.open_tool_call.take() else {
        return;
    };
    // 原有逻辑：生成 ToolCallEnd 事件并附加到 message 的 tool_calls 中
}
}

```

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

负责将 OpenAI 请求中的 parallel_tool_calls 映射到 ChatRequest，并添加了对应的单元测试。

```

// 在 prepare_chat_request 函数中映射 parallel_tool_calls，默认值为 true
parallel_tool_calls: request.parallel_tool_calls.unwrap_or(true),

```

```

// 测试 parallel_tool_calls = false 映射
#[test]
fn prepare_chat_request_maps_parallel_tool_calls() {
    let mut request = base_request();
    request.parallel_tool_calls = Some(false);

    let prepared = prepare_chat_request(
        request,
        &served(&["Qwen/Qwen1.5-0.5B-Chat"]),
        ResolvedRequestContext::default(),
    )
    .expect("request is valid");

    assert!(!prepared.chat_request.parallel_tool_calls);
}

```

```
}

// 测试默认值 true
#[test]
fn prepare_chat_request_defaults_parallel_tool_calls_to_true() {
    let prepared = prepare_chat_request(
        base_request(),
        &served(&["Qwen/Qwen1.5-0.5B-Chat"]),
        ResolvedRequestContext::default(),
    )
    .expect("request is valid");

    assert!(prepared.chat_request.parallel_tool_calls);
}
```

评论区精华

- BugenZhao 建议将过滤下沉到 `structured_chat_event_stream` 内部，避免路由层重复实现，使流式与非流式共享相同语义。采纳后，实现集中在结构化流中。
- tahsintunan 指出 `should_emit_tool_call` 等函数在上游已保证 `index==0` 的情况下是冗余的，建议移除。作者同意并修复。
- 另有关于 `None` 映射的边角讨论，确认当前行为与文档一致。

风险与影响

- 向后兼容：默认值 `true` 保证了兼容性，但旧版本若传递 `parallel_tool_calls` 会返回错误（未实现），现改为静默接受，可能影响依赖该错误处理逻辑的客户端。
- 状态机复杂度：`suppressing_tool_call` 标志与事件流交互需谨慎，当前单元测试覆盖了主要路径，但未覆盖混合内容（如工具调用与推理块交错）的场景。
- 影响范围：仅 Rust 前端，且仅在使用工具调用时才生效，普通文本生成无影响。

关联脉络

该 PR 是 Rust 前端功能对等性 (Issue #44280) 的进展项。在此之前，Rust 前端已支持基本的工具调用生成与流式解析（相关 PR 如 #45413 引入流式解析引擎）。本 PR 补齐了 `parallel_tool_calls` 参数，使 Rust 前端更接近 Python 前端的工具调用功能矩阵。