PR #45216 完整报告

vllm-project/vllm

[Rust Frontend] Add standalone `granite4` tool parser

合并时间：2026-06-13 00:16

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45216

执行摘要

PR #45216 为 Rust 前端添加了独立的 granite4 工具调用解析器，支持 Granite 4 模型特有的 <tool_call> 格式以及 arguments 的两种形式（JSON 对象或转义字符串）。实现轻量，复用了共享 JSON 工具，并通过正确的模型路由匹配确保向后兼容。评审已批准，可以合并。

功能与动机

Granite 4 模型生成工具调用时，arguments 既可以是 JSON 对象也可以是转义的 JSON 字符串（见 issue #36827）。共享的 JSON 配置无法表达字符串参数的分步解析，因此需要一个独立的解析器。该 PR 是追踪 issue #44280 和前置 PR #44713 的后续。

实现拆解

1. 新增解析器核心：在 rust/src/tool-parser/src/json/granite4.rs 中定义 Granite4Mode 状态机和 Granite4Event 事件，实现 Granite4ToolParser 结构体。通过 parse_into 方法利用winnow解析器逐字节扫描缓冲区，将文本解析为事件序列，并由apply_event更新输出。关键设计：当 arguments 首字节为 { 时走对象路径（流式收集），为 " 时走字符串路径（解码后整体发射）。
2. 模块导出：在 rust/src/tool-parser/src/json/mod.rs 中添加 mod granite4; 和 pub use granite4::Granite4ToolParser;；在 rust/src/tool-parser/src/lib.rs 的 pub use json:: 列表中添加 Granite4ToolParser，使外部 crate 可用。
3. 注册解析器：在 rust/src/chat/src/parser/tool/mod.rs 的 names 模块添加 pub const GRANITE4: &str = "granite4"；在 ToolParserFactory::new 中添加 register_parser::<Granite4ToolParser>(names::GRANITE4) 和 register_pattern("granite-4", names::GRANITE4)，支持通过名称或模型 ID 自动路由。
4. 测试与兼容性：在 rust/src/chat/src/parser/tool/tests.rs 中添加断言，验证 ibm-granite/granite-4.0-h-tiny 路由到 GRANITE4。更新 rust/src/chat/src/lib.rs 中的错误报告字符串，将 granite4 加入可用解析器列表。

rust/src/tool-parser/src/json/granite4.rs

新增核心解析器实现，包含状态机和事件处理逻辑

```
//! rust/src/tool-parser/src/json/granite4.rs
//! Granite 4 工具解析器：支持 `<tool_call>` 标签，
//! `arguments` 可以是 JSON 对象或转义 JSON 字符串。

use winnow::prelude::*;
```

```rust
// ... 其他导入

/// 解析器状态：处于文本、头部、参数或关闭模式。
#[derive(Debug, Clone, PartialEq, Eq)]
enum Granite4Mode {
    Text,
    Header,
    /// 参数解析中：`None` 表示尚未决定是对象还是字符串，
    /// `Some` 表示正在流式解析对象值。
    Args { json_scan: Option<JsonObjectScanState> },
    Close,
}

/// 解析过程中触发的事件。
#[derive(Debug, Clone, PartialEq, Eq)]
enum Granite4Event {
    Text { len: usize },
    ToolCallStart,
    ToolCallHeader { function_name: String },
    /// 对象参数的增量字节；`complete` 表示闭合大括号已到。
    ObjectArgsDelta { len: usize, complete: bool },
    /// 字符串参数的完整解码结果。
    StringArgs { decoded: String },
    ToolCallEnd,
}

/// Granite 4 工具解析器主结构体。
pub struct Granite4ToolParser {
    buffer: String,
    mode: Granite4Mode,
    active_tool_index: Option<usize>,
    emitted_tool_count: usize,
}

impl Granite4ToolParser {
    fn new(_tools: &[Tool]) -> Self {
        Self { buffer: String::new(), mode: Granite4Mode::Text, active_tool_index: None, emitted_
        tool_count: 0 }
    }

    /// 将事件应用到输出，并根据事件转换状态。
    fn apply_event(&mut self, event: Granite4Event, output: &mut ToolParserOutput) -> Result<()
    > {
        match event {
            Granite4Event::Text { len } => output.normal_text.push_str(&self.buffer[..len]),
            Granite4Event::ToolCallStart => self.mode = Granite4Mode::Header,
            Granite4Event::ToolCallHeader { function_name } => {
                let tool_index = self.emitted_tool_count;
                self.emitted_tool_count += 1;
```

```
                self.active_tool_index = Some(tool_index);
                self.mode = Granite4Mode::Args { json_scan: None };
                output.calls.push(ToolCallDelta {
                    tool_index,
                    name: Some(function_name),
                    arguments: String::new(),
                });
            }
            Granite4Event::ObjectArgsDelta { len, complete } => {
                let arguments = self.buffer[..len].to_string();
                self.push_arguments(arguments, output)?;
                if complete { self.mode = Granite4Mode::Close; }
            }
            Granite4Event::StringArgs { decoded } => {
                self.push_arguments(decoded, output)?;
                self.mode = Granite4Mode::Close;
            }
            Granite4Event::ToolCallEnd => { self.active_tool_index = None; self.mode =
            Granite4Mode::Text; }
        }
        Ok(())
    }
    // 其他方法如 push_arguments, reset, create, parse_into 在完整文件中。
}
```

rust/src/chat/src/parser/tool/mod.rs

注册 granite4 解析器及其模型路由模式

```
// rust/src/chat/src/parser/tool/mod.rs ( 部分 )

/// 注册的解析器名称。
pub mod names {
    // 其他解析器名称 ...
    pub const GRANITE4: &str = "granite4"; // Granite 4 解析器名称
}

impl ToolParserFactory {
    /// 创建默认注册表并注册内置解析器。
    pub fn new() -> Self {
        let mut factory = Self::default();

        factory
            .register_parser::<Granite4ToolParser>(names::GRANITE4) // 注册解析器
            // 其他解析器 ...
            .register_pattern("granite-4", names::GRANITE4); // 模型 ID 模式路由

        factory
    }
}
```

评论区精华

BugenZhao 在审查时指出一处注释不准确：“<tool_call> 是 added token 但不是 special token，不应在注释中称为特殊 token”。作者已接受并修正。

风险与影响

- 风险：字符串参数解码错误可能导致格式异常，但实现与 Python 对齐；模型路由仅匹配 granite-4 前缀，自动与旧 Granite 模型分离，风险低。缺乏针对流式解析的独立单元测试，但路由测试已覆盖。
- 影响：仅影响 Rust 前端，新增功能，不会影响现有解析器。Granite 4 用户获得正确工具调用支持。

关联脉络

该 PR 是 Rust 前端工具解析器系列的一部分，前置 PR #44713 添加了基础支持，后续其他模型解析器可沿用同样模式。与 issue #36827 紧密相关，解决了真实模型行为差异。