PR #43481 完整报告

vllm-project/vllm

[Rust Frontend] Add InternLM2 tool parser

合并时间：2026-06-01 16:58

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43481

执行摘要

本 PR 为 Rust 前端 vllm-chat 新增 InternLM2 模型工具调用解析器，复用共享的 JsonToolCallParser 核心，并扩展配置以支持多候选参数键。变更影响 Rust 工具解析器模块，不涉及 Python 侧。代码质量高，测试覆盖完善，讨论澄清了与 Python 参考的有意行为差异，建议合并。

功能与动机

根据 PR 描述，目的是将 Python 的 internlm2_tool_parser.py 移植到 Rust，使 vllm-chat 能够自动识别 InternLM2 系列模型（ internlm2-chat-* 、 internlm2_5-* ）的工具调用格式。InternLM2 使用 <|action_start|><|plugin|>{...}<|action_end|> 特殊 token 包裹 JSON 工具调用，且支持 parameters 或 arguments 作为参数键名。

实现拆解

1. 新增 Internlm2ToolParser 结构体（ rust/src/tool-parser/src/json/internlm2.rs ）：定义配置常量，实现 ToolParser trait，设置 preserve_special_tokens 为 true 。
2. 扩展共享配置 JsonToolCallConfig （ rust/src/tool-parser/src/json/mod.rs ）：将 arguments_key 从 &'static str 改为 &'static [&'static str] ，新增 json_arguments_key 解析函数，使用 winnow 组合子实现多候选键匹配和更好的错误上下文。
3. 更新现有解析器：四个已有 JSON 解析器（ Hermes 、 Llama 、 Mistral 、 Qwen ）的配置从字符串调整为单元素数组，行为不变。
4. 注册解析器与模型模式（ rust/src/chat/src/parser/tool/mod.rs ）：添加 INTERNLM 名称常量，注册 Internlm2ToolParser ，添加模式匹配 "internlm2" 并附带负面排除注释。
5. 添加测试：在 tests.rs 中验证路由逻辑，在 internlm2.rs 中添加 12 个单元测试覆盖完整解析、流式增量、多块提取、边界情况等。
6. 模块导出（ rust/src/tool-parser/src/lib.rs ）：公开导出 Internlm2ToolParser 。

rust/src/tool-parser/src/json/internlm2.rs

新增 InternLM2 工具解析器的核心实现，定义了配置和解析逻辑。

```
use super::{JsonToolCallConfig, JsonToolCallParser, JsonToolCallWhitespace};
use crate::{Result, Tool, ToolParser, ToolParserOutput};

/*
 * InternLM2 工具调用配置
```

```
 * 起始标记：`<|action_start|><|plugin|>`
 * 结束标记：`<|action_end|>`
 * 参数键同时接受 `parameters` 或 `arguments`，优先使用第一个遇到的
 */
const INTERNLM2_CONFIG: JsonToolCallConfig = JsonToolCallConfig {
    parser_name: "InternLM2",
    start_marker: "<|action_start|><|plugin|>",
    end_marker: "<|action_end|>",
    marker_whitespace: JsonToolCallWhitespace::Optional,
    delimiter: None,
    name_key: "name",
    arguments_key: &["parameters", "arguments"],
};

/// InternLM2 特殊 token 包裹的 JSON 工具调用解析器
pub struct Internlm2ToolParser {
    inner: JsonToolCallParser,
}

impl Internlm2ToolParser {
    fn new(_tools: &[Tool]) -> Self {
        Self {
            inner: JsonToolCallParser::new(INTERNLM2_CONFIG),
        }
    }
}

impl ToolParser for Internlm2ToolParser {
    fn create(tools: &[Tool]) -> Result<Box<dyn ToolParser>> {
        Ok(Box::new(Self::new(tools)))
    }

    // 保留 `<|action_start|>` 等特殊 token，对应 Python 的
    // `adjust_request(skip_special_tokens=False)`
    fn preserve_special_tokens(&self) -> bool {
        true
    }
    // parse_into, finish 等方法委托 inner 解析器处理
}
```

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

共享 JSON 工具解析核心的修改，是关键架构变更点。

```
/// 解析 JSON 对象键，接受 `candidates` 候选列表中的任意一个。
/// 使用 `winnow` 的 `verify` 先解析完整 JSON 字符串，再检查是否在候选列表中。
/// 若失败，通过循环为每个候选键添加 `Expected` 上下文，使错误信息枚举所有合法键。
fn json_arguments_key(
    input: &mut JsonToolInput<'_>,
    candidates: &'static [&'static str],
```

```
) -> ModalResult<()> {
    let start = input.checkpoint(); // 记录输入位置用于错误上下文
    json_str
        .verify(|key: &String| candidates.contains(&key.as_str()))
        .void()
        .parse_next(input)
        .map_err(|err| {
            // 为每个候选键添加 Expected 上下文
            err.map(|context_error| {
                candidates.iter().fold(context_error, |context_error, candidate| {
                    context_error.add_context(
                        &*input,
                        &start,
                        StrContext::Expected(StrContextValue::StringLiteral(candidate)),
                    )
                })
            })
        })
}
```

评论区精华

- BugenZhao指出 json_arguments_key 中硬编码的候选键遍历方式可以使用 for 循环改进，作者随后采纳并实现为循环附加上下文，使错误信息更完整。
- BugenZhao进一步确认跳过 prelude 文本是否为故意行为，作者解释这是 Python 非流式路径的遗留代码，流式路径无等价处理，决定不实现以保持一致性。

风险与影响

- 核心配置类型变更：arguments_key 从字符串改为切片引用，需确保所有使用处已同步更新。
- 行为差异：已知三项差异（参数值类型限制、未知键硬错误、字段顺序要求），文档已记录但未修复，可能影响依赖精确 Python 行为的用户。
- 模型路由：模式 "internlm2" 经测试验证正确区分 InternLM2 、 InternLM v1 、 InternLM3 和 Intern-S1 ，风险可控。
- 影响范围：仅影响 Rust 前端 vllm-chat ， Python 侧不受影响；新增代码良好模块化，无侵入性。

关联脉络

本 PR 是 Rust 前端工具解析器覆盖推进的一部分，此前已包含 Hermes 、 Llama 、 Mistral 、 Qwen 等解析器。未来可能继续移植 DeepSeek 等其他模型的解析器，并逐步处理文档中记录的已知差异。