PR #46507 完整报告

vllm-project/vllm

[Rust Frontend] Make Granite4 string argument scanning incremental

合并时间：2026-06-26 11:54

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46507

执行摘要

本 PR 针对 Granite4 工具调用中长字符串参数在流式传输时的重复扫描问题，引入增量扫描状态机，将 64KB 字符串解析时间从 107ms 降至 0.166ms（约 650 倍提升）。核心是在 utils.rs 中新增 JsonStringScanState 和 take_json_string，并在 Granite4 解析器中集成以支持跨 chunk 增量扫描。同时重构了已有的一次性解析接口 json_str 以复用该扫描逻辑。

功能与动机

Granite4 模型在流式输出工具调用时，arguments 字段可能直接是一个 JSON 对象（{"data":"..."}），也可能是一个 JSON 字符串（"{\"data\":\"...\"}"）。对于后者，原实现使用 json_str 一次性解析，每次新 chunk 到达时都从整个缓冲区的开头重新扫描，导致长字符串参数的解析时间随缓冲区大小二次增长。PR 描述中的基准数据定量揭示了问题：64KB 输入耗时 107.961ms，且大小翻倍时耗时增长约 4x。为解决此问题，需要为字符串形式也引入扫描状态，使其能记住已处理位置，从而线性缩放。

实现拆解

1. 增量扫描原语（rust/src/parser/src/utils.rs）：定义 JsonStringScanState（scanned_len 和 escape），新增 take_json_string 函数。该函数消耗 Partial<&str> 和一个可变状态，从状态记录的偏移量开始扫描，遇到完整字符串返回长度，遇到不完整输入保存偏移并返回 Incomplete。
2. 保持向后兼容：重写 json_str，使其通过构造临时 JsonStringScanState 调用 take_json_string，然后通过新函数 decode_json_str 解码。这样既保留了旧接口，又避免了重复实现。
3. Granite4 解析器集成（rust/src/parser/src/tool/json/granite4.rs）：引入 Granite4ArgsScan 枚举，替换 Granite4Mode::Args 中的 json_scan 字段。args_event 函数在首次遇到字符串起始引号时创建 String 状态，后续 chunk 直接使用该状态调用 take_json_string。
4. 测试与基准：在 utils.rs 中新增 4 个单元测试覆盖完整字符串、跨 chunk 恢复、转义追踪和拒绝非字符串起始。新增 rust/src/parser/benches/granite4.rs 基准，以 7 字节分块模拟流式输入，验证线性缩放特性。更新 Cargo.toml 注册 bench 目标。

rust/src/parser/src/utils.rs

核心改动所在：定义了增量扫描状态 JsonStringScanState、增量扫描函数 take_json_string、一次性解析 json_str 的重构以及新的解码函数 decode_json_str。这是整个 PR 的基础设施。

```
/// 流式 JSON 字符串词法状态：记录已经扫描的长度和转义状态。
#[derive(Debug, Clone, Default, PartialEq, Eq)]
pub struct JsonStringScanState {
    scanned_len: usize,
    escape: bool,
}

/// 从上次扫描位置继续解析一个原始的 JSON 字符串，返回字符串长度（含引号）。
/// 当输入不完整时保存状态并返回 `Incomplete`。
pub fn take_json_string(
    input: &mut Partial<&str>,
    state: &mut JsonStringScanState,
) -> ModalResult<usize> {
    let text = **input;
    if text.is_empty() {
        return incomplete();
    }
    let bytes = text.as_bytes();
    if bytes[0] != b'"' {
        return Err(json_scan_error(
            "JSON string",
            StrContextValue::CharLiteral('"'),
        ));
    }
    // 从上游保存的位置开始扫描：初始化为引号后第一个字节，后续从 saved scanned_len 继续
    let mut index = if state.scanned_len == 0 {
        1 // 跳过起始引号
    } else if state.scanned_len <= bytes.len() {
        state.scanned_len
    } else {
        return incomplete();
    };

    while index < bytes.len() {
        let byte = bytes[index];
        index += 1;

        if state.escape {
            state.escape = false;
            continue;
        }
        match byte {
            b'\\' => state.escape = true,
            b'"' => {
                // 遇到结束引号，消耗输入并返回整个字符串长度
                input.next_slice(index);
                return Ok(index);
            }
            _ => {}
```

```
        }
    }
    // 输入耗尽，保存当前已扫描长度（即整个输入长度）
    state.scanned_len = text.len();
    incomplete()
}

/// 一次性解析一个完整的 JSON 字符串（用于非流式场景）。
pub fn json_str(input: &mut Partial<&str>) -> ModalResult<String> {
    let text = **input;
    let checkpoint = input.checkpoint();
    let mut state = JsonStringScanState::default();
    let len = take_json_string(input, &mut state)?;
    decode_json_str(&text[..len]).inspect_err(|_| {
        input.reset(&checkpoint);
    })
}

/// 解码一个完整的 JSON 字符串字面量（仅含引号和内容）。
pub fn decode_json_str(raw: &str) -> ModalResult<String> {
    serde_json::from_str::<String>(raw).map_err(|_| {
        json_scan_error(
            "JSON string",
            StrContextValue::Description("valid JSON string"),
        )
    })
}
```

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

消费侧集成：引入 Granite4ArgsScan 枚举以区分对象和字符串扫描状态，修改 args_event 函数使其在遇到字符串起始时创建字符串扫描状态，并跨 chunk 复用 take_json_string。

```
/// Granite4 解析器的参数扫描状态：可以是对象扫描或字符串扫描。
#[derive(Debug, Clone, PartialEq, Eq)]
enum Granite4ArgsScan {
    Object(JsonObjectScanState),
    String(JsonStringScanState),
}

/// 解析 arguments 值：根据第一个字符决定是对象还是字符串，然后持续扫描。
fn args_event(
    input: &mut JsonToolInput<'_>,
    args_scan: &mut Option<Granite4ArgsScan>,
) -> ModalResult<Granite4Event> {
    // 如果已有扫描状态，直接继续
    if let Some(scan) = args_scan {
        return match scan {
            Granite4ArgsScan::Object(scan) => {
                let len = take_json_object(input, scan)?;
```

```
                Ok(Granite4Event::ObjectArgsDelta {
                    len,
                    complete: scan.complete(),
                })
            }
            Granite4ArgsScan::String(scan) => string_args_event(input, scan),
        };
    }

    // 首次遇到 arguments 值，探测实际类型
    match peek(any).parse_next(input)? {
        '{' => {
            let mut scan = JsonObjectScanState::default();
            let len = take_json_object(input, &mut scan)?;
            let complete = scan.complete();
            *args_scan = Some(Granite4ArgsScan::Object(scan));
            Ok(Granite4Event::ObjectArgsDelta { len, complete })
        }
        '"' => {
            // 字符串起始：创建字符串扫描状态并立即扫描
            *args_scan = Some(Granite4ArgsScan::String(JsonStringScanState::default()));
            let Some(Granite4ArgsScan::String(scan)) = args_scan else {
                unreachable!("Granite4 string args state just set");
            };
            string_args_event(input, scan)
        }
        _ => Err(json_scan_error(
            "Granite4 tool arguments",
            StrContextValue::Description("object or string"),
        )),
    }
}

/// 处理字符串参数：使用 `take_json_string` 扫描完整字符串后再解码。
fn string_args_event(
    input: &mut JsonToolInput<'_>,
    scan: &mut JsonStringScanState,
) -> ModalResult<Granite4Event> {
    let text = **input;
    let checkpoint = input.checkpoint();
    let len = take_json_string(input, scan)?;
    let decoded = decode_json_str(&text[..len])?;
    Ok(Granite4Event::StringArgs { decoded })
}
```

评论区精华

> BugenZhao（审核人）：“I'm wondering if we can now simply let the full `json_str` to reuse the newly introduced streaming `take_json_string` by constructing a temporary

> state. Most of the logic here looks the same."
>
> reidliu41（作者）："Thanks for your feedback. Done — json_str now reuses take_json_string with a temporary JsonStringScanState, then decodes the completed string via decode_json_str."

该对话体现了在引入新功能时主动重构旧接口以保持一致的工程实践，最终代码更加精简且复用性更好。

风险与影响

- 正确性风险：增量扫描状态（scanned_len、escape）在多 chunk 场景下需要精确维护。4 个单元测试覆盖了核心边界，且 take_json_string 的逻辑与之前 json_str 的扫描部分等价，风险可控。
- 性能影响：仅优化了字符串参数路径，对象路径保持不变。字符串参数路径的渐进复杂度从 $O(n^2)$ 降为 $O(n)$，对长参数效果显著。
- 兼容性：未改动任何外部 API 或模型接口，仅影响 Granite4 解析器的内部实现。其他解析器（如 Llama3、Gemma4）不受影响。
- 维护影响：JsonStringScanState 和 take_json_string 设计为通用原语，未来可被其他需要增量字符串扫描的解析器复用。

关联脉络

本 PR 是 Rust 前端解析器持续优化中的一环。此前已有 #46602（Gemma4 解析器迁移至统一接口）、#46719（提取渲染器测试工具）等重构，均在 rust/src/parser 目录下进行。这些变更共同朝着更结构化、更高效的流式解析架构演进。本 PR 专注于解决 Granite4 字符串参数这一特定场景的性能退化，其增量扫描思路也可推广到类似场景。