PR #44311 完整报告

vllm-project/vllm

[Rust Frontend] Fix several hf chat template rendering issues

合并时间：2026-06-03 16:04

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44311

执行摘要

本次 PR (#44311) 修复了 Rust 前端 HF chat template 渲染中的两个关键兼容性问题：数字序列化泄漏内部表示以及 map 字段与 Python dict 方法冲突。通过全局移除 serde_json 的 arbitrary_precision 特性并引入自定义 TemplateMap 类型，解决了 MiniMax M2.5 等模型的工具调用模板渲染失败。改动涉及 9 个文件，新增 183 行，删除 56 行，已合并至 main。

功能与动机

HF chat template 在 Python Jinja2 中运行正常，但移植到基于 MiniJinja 的 Rust 实现后暴露出两个差距：

- 数字精度泄漏：启用 arbitrary_precision 后，serde_json::Value 通过 MiniJinja 序列化时会泄漏 $serde_json::private::Number 内部结构（minijinja issue #641）。
- 字段方法冲突：MiniJinja 的默认 Serde map 路径优先解析 map 中的同名字段，导致 dict.items() 被字段值覆盖（minijinja issue #903）。

这两个问题直接影响 MiniMax M2.5、Qwen 系列等模型，当工具调用参数中包含 items 键时模板渲染出错。

实现拆解

1. 回退任意精度数字：在 rust/Cargo.toml 中移除 serde_json 的 arbitrary_precision 特性，数字回退到标准 serde_json 序列化。更新 rust/src/chat/src/renderer/hf/tojson.rs 和 rust/src/tool-parser/src/parameters.rs 中的测试期望值，新增 serialized_json_numbers_do_not_leak_serde_private_representation 测试验证泄漏消除。
2. 引入自定义 Object 类型：新建 rust/src/chat/src/renderer/hf/value.rs，定义 TemplateValue 和 TemplateMap。TemplateMap 实现 Object trait 时，call_method 始终返回 UnknownMethod，将 dict 方法调用导向 MiniJinja 的 pycompat 回调，避免字段影子。
3. 集成到渲染器：在 rust/src/chat/src/renderer/hf/mod.rs 中将 TemplateToolFunction::arguments 和 TemplateToolDefinition::parameters 类型由 serde_json::Value 改为 TemplateValue，并在构造处调用 to_template_value。同时添加单元测试方法验证 items 键不再冲突。
4. 调整配套测试：更新 rust/src/chat/tests/roundtrip.rs 中的轮转测试数据，显式加入 items 字段，覆盖字段影子场景。

rust/src/chat/src/renderer/hf/value.rs

核心新增文件，定义 TemplateValue 和 TemplateMap，解决字段影子问题

```
// rust/src/chat/src/renderer/hf/value.rs

use std::sync::Arc;

use indexmap::IndexMap;
use minijinja::value::{Enumerator, Object, ObjectExt, ObjectRepr};
use minijinja::{Error as TemplateError, ErrorKind as TemplateErrorKind, State};
use serde::Serialize;
use serde_json::Value as JsonValue;

/// A transparent wrapper around `minijinja::Value` constructed via `to_template_value`.
/// It ensures that map objects use `TemplateMap` to avoid field-method shadowing.
#[derive(Debug, Serialize)]
#[serde(transparent)]
pub(super) struct TemplateValue(minijinja::Value);

/// Recursively convert a `serde_json::Value` into a `TemplateValue`.
/// Arrays and primitives are passed through directly; maps are wrapped in `TemplateMap`.
pub(super) fn to_template_value(value: JsonValue) -> TemplateValue {
    TemplateValue(match value {
        JsonValue::Array(values) => values
            .into_iter()
            .map(to_template_value)
            .map(|value| value.0)
            .collect::<minijinja::Value>(),
        JsonValue::Object(values) => minijinja::Value::from_object(TemplateMap(
            values
                .into_iter()
                .map(|(key, value)| (key, to_template_value(value).0))
                .collect(),
        )),
        // For primitive values, use `from_serialize` which now uses standard number
        representation.
        value => minijinja::Value::from_serialize(value),
    })
}

// Custom Object implementation that forwards field access but always returns UnknownMethod
// for method calls, ensuring that pycompat's unknown_method_callback is used instead.
#[derive(Debug)]
struct TemplateMap(IndexMap<String, minijinja::Value>);

impl Object for TemplateMap {
    fn repr(self: &Arc<Self>) -> ObjectRepr {
        ObjectRepr::Map
    }
```

```
    fn get_value(self: &Arc<Self>, key: &minijinja::Value) -> Option<minijinja::Value> {
        self.0.get(key.as_str()?).cloned()
    }

    fn get_value_by_str(self: &Arc<Self>, key: &str) -> Option<minijinja::Value> {
        self.0.get(key).cloned()
    }

    fn enumerate(self: &Arc<Self>) -> Enumerator {
        self.mapped_rev_enumerator(|this| {
            Box::new(this.0.keys().map(|key| minijinja::Value::from(key.as_str())))
        })
    }

    fn enumerator_len(self: &Arc<Self>) -> Option<usize> {
        Some(self.0.len())
    }

    fn call_method(
        self: &Arc<Self>,
        _state: &State<'_, '_>,
        _method: &str,
        _args: &[minijinja::Value],
    ) -> std::result::Result<minijinja::Value, TemplateError> {
        // Always return UnknownMethod so that Python dict methods like `items()` are handled
        // by MiniJinja's pycompat rather than being shadowed by a map field with the same
        name.
        Err(TemplateError::from(TemplateErrorKind::UnknownMethod))
    }
}
```

rust/src/chat/src/renderer/hf/tojson.rs

测试模块更新，验证数字序列化不再泄漏内部表示并调整期望值

```
// rust/src/chat/src/renderer/hf/tojson.rs (tests 模块 )

#[test]
fn tojson_uses_standard_serde_json_number_spelling() {
    let payload = serde_json::from_str(r#"{"x":2,"y":1.00}"#).unwrap();
    let rendered = render("{{ payload|tojson }}", payload);
    // 启用了 arbitrary_precision 时会被序列化为 {"x": 2, "y": 1.00},
    // 但现在使用标准序列化，`1.00` 被规范化为 `1.0`。
    assert_eq!(rendered, r#"{"x": 2, "y": 1.0}"#);
}

#[test]
fn serialized_json_numbers_do_not_leak_serde_private_representation() {
    let payload: serde_json::Value = serde_json::from_str(r#"{"x":2,"y":1.00}"#).unwrap();
    let rendered = render("{{ payload }}", payload);
```

```
    // 确保不再泄漏 `$serde_json::private::Number` 内部结构。
    assert!(!rendered.contains("$serde_json::private::Number"));
    assert_eq!(rendered, r#"{"x": 2, "y": 1.0}"#);
}
```

评论区精华

- Codex 机器人："移除 arbitrary_precision 会影响其他依赖 serde_json::Value 的 crate，例如 DeepSeek V4 助手工具调用渲染路径，数字 1.00 会变成 1.0。建议限定作用域。"（P1 级别）
- njhill（批准者）："Thanks! I wonder if codex can help with fixes on the minijinja side..." 表明希望上游也解决这两个问题。

团队接受了全局移除的取舍，未对 Codex 建议做出额外响应。

风险与影响

风险：

- 全局移除 arbitrary_precision 属于跨 crate 依赖变更，可能影响 DeepSeek V4 等非 HF 渲染路径中的数字格式化行为，导致精度不如 Python 版本。
- TemplateMap 是新代码，虽然有一定测试覆盖，但边界情况（如嵌套多层 map、空 map）可能存在 bug。
- 部分测试断言被注释掉（如 5.00、1e0、9223372036854775807.5），表明已知精度妥协。

影响：

- 范围：仅影响 Rust 前端（v1 架构）的 HF chat template 渲染器，不影响其他路径。
- 用户：修复了 MiniMax M2.5、Qwen 等模型的工具调用渲染，正向影响；数字格式标准化对大多数用户无感知。
- 团队：维护成本降低（不再依赖上游 Arbitrary precision 封装），但需注意后续引入 serde_json::Value 序列化时的行为变化。

关联脉络

- PR #43582：引入了 arbitrary_precision 特性，本 PR 撤销了该变更并改用更稳妥的标准序列化。
  - Issue #641 / #903（minijinja 上游）：对应两个兼容性问题的根因，当前方案在 MiniJinja 侧通过 pycompat 回避了字段冲突，数字精度问题则通过规避 arbitrary_precision 解决，没有直接修改 MiniJinja 核心。
  - 后续可以跟踪上游修复进度，若 MiniJinja 原生支持 dict 方法优先或正确处理 arbitrary_precision，可以回退部分本地 Hack。