

PR #54821 完整报告

vllm-project/vllm

[Frontend][Rust] Reject empty structured-output values

合并时间: 2026-09-12 23:12

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54821>

执行摘要

本 PR 把 Rust 前端对结构化输出 (structured outputs) 的输入校验收敛到请求边界: 在 `rust/src/engine-core-client/src/protocol/structured_outputs.rs` 新增共享的 `StructuredOutputsParams::validate()`, 并让 Serde 反序列化、OpenAI `response_format`、gRPC 转换三条入口统一调用, 使空白 grammar、空 json 字符串、空 choice、空 structural tag、非法 JSON schema 在进入调度器之前就变成干净的 400 / `INVALID_ARGUMENT`。改动共 4 个文件、+175/-11, 含 10 个新增单测; review 过程中 Codex 与 @arpera 的 agentic review 先后指出 gRPC 与 `response_format` 两处漏检路径, 最终三处入口全部复用同一校验函数, PR 已由 @vadiklyutiy 合并。

功能与动机

动机来自 Rust 前端与 Python 前端输入契约的不一致。PR body 写道: 「Empty (whitespace-only) `json/grammar` structured-output values were forwarded past the Rust frontend instead of being rejected at the request boundary」——这些值会一路走到语法编译才失败, 以「per-request error」的形式暴露, 而不是请求边界上的 400; 作者同时澄清「No engine abort; grammar-compile failures are carried per-request to the scheduler.」。

该改动从 #51450 按 @arpera 的请求拆分出来以便独立落地, 目标是「aligns the Rust frontend with the Python frontend's input contract」。分支名 `fix/ghsa-85xf-rust-empty-structured-output` 中的 GHSA 编号暗示其与一条安全类报告相关 (材料中未给出该 GHSA 的具体内容, 此处仅作线索记录)。

实现拆解

变更入口: Rust 前端的结构化输出参数类型 `StructuredOutputsParams` 及其三条构造 / 转换路径。

1. 协议层提供可失败校验 (engine-core-client) - 文件与符号:

`rust/src/engine-core-client/src/protocol/structured_outputs.rs` 的

`StructuredOutputsParams::validate()`。 - 变更: 新增 `pub fn validate(&self) ->`

`Result<>`, 按 `StructuredOutputConstraint` 分支拦截空白 `Grammar`、字符串形态且空白或不可解析的 `Json`、空 `Choice`、空白 `StructuralTag`; `json_object`、非空 JSON 对象与 `regex` 走默认放行分支。 - 原因与影响: 构造函数 `from_constraint` 保持不可失败, 内部调用点无需改造, 可失败性只在跨越请求边界时引入, 避免把校验撒进所有构造路径。

2. Serde / HTTP 请求体路径接入 - 文件与符号：同文件的 `impl TryFrom<WireStructuredOutputsParams> for StructuredOutputsParams`。 - 变更：由直接 `Ok(Self { .. })` 改为先构造 `params`，再 `params.validate()? 后返回`。 - 原因与影响：`structured_outputs` 字段的反序列化失败会自然上抛为 `Error::InvalidStructuredOutputsParams`，进而转成 400——这正是 @vadiklyutiy 要求补测的路径。
3. gRPC 路径补齐 (Codex P1) - 文件与符号：`rust/src/server/src/grpc/convert.rs` 的 `convert_structured_output`。 - 变更：在 `Ok(Some(params))` 之前执行 `params.validate().map_err(|error| Status::invalid_argument(error.to_report_string()))?`，并引入 `thiserror_ext::AsReport`。 - 原因与影响：该函数使用无校验构造函数（`grammar() / choice() / structural_tag()`），不走 `TryFrom`，因此是此前唯一未被覆盖的入口；`Json` 分支本身已把字符串解析为 `serde_json::Value`，解析失败会先以 `invalid json schema` 返回。
4. OpenAI `response_format` 路径补齐 (agentic review 发现) - 文件与符号：`rust/src/server/src/routes/openai/utls/structured_outputs.rs` 的 `convert_from_response_format`。 - 变更：由「`match` 分支内直接 `Ok(Some(...))`」改为先聚合 `params`，末尾统一 `params.validate().map_err(|error| ApiError::invalid_request(error.to_report_string(), Some("response_format")))?`。 - 原因与影响：空白 `json_schema`、空 `choice`、空 `structural_tag` 现在也会 400；`convert_from_response_format_value` (completions 端点) 通过委托间接获得同样行为。
5. 测试配套 (无独立测试文件，全部位于源文件的 `#[cfg(test)] mod tests`)

位置	新增用例	覆盖目标
structured_outputs.rs	structured_outputs_rejects_empty_json_string、 validate_rejects_empty_grammar_from_constructor、 validate_rejects_empty_json_string_from_constructor、 validate_rejects_malformed_json_string、 validate_rejects_empty_choice、 validate_rejects_empty_structural_tag、 validate_accepts_non_empty_grammar	validate() 各分支与正向放行
grpc/convert.rs	grpc_rejects_empty_grammar_before_engine	gRPC 转换在触达引擎前拒绝空白 grammar

位置	新增用例	覆盖目标
chat _co mple tion s/co nver t.rs	prepare_chat_request_rejects_empty_json_schema、 prepare_chat_request_rejects_whitespace_only_structured_outputs_g rammar	HTTP 路径断 言 Stat usCode: :BAD_R EQUES T 与错 误文案

关键源码片段

rust/src/engine-core-client/src/protocol/structured_outputs.rs

核心文件：新增共享的 `StructuredOutputsParams::validate()`，并在 `TryFrom<WireStructuredOutputsParams>` 末尾接入，是 Serde / HTTP 路径的校验落点，同时承载 7 个新增单测。

共享校验方法（本 PR 的核心新增逻辑）：

```
impl StructuredOutputsParams {
    /// 请求边界的统一校验入口：Serde 反序列化、OpenAI response_format、
    /// gRPC 转换三条路径都会复用它，保证「非法约束不入调度」。
    pub fn validate(&self) -> Result<()> {
        match &self.constraint {
            // grammar 只做空白检查，真正的语法编译依旧交给 EngineCore；
            // 这里拦住空串是为了返回干净的 400，而不是让它变成一次请求失败。
            StructuredOutputConstraint::Grammar(grammar) if grammar.trim().is_empty() => {
                Err(Error::InvalidStructuredOutputsParams {
                    message: "structured_outputs.grammar cannot be an empty string".to_string(),
                })
            }
            // json 约束允许两种形态：直接给 JSON 对象，或给「字符串形式的 schema」。
            // 后者必须非空且自身可被解析，否则语义上等价于空语法。
            StructuredOutputConstraint::Json(Value::String(value)) => {
                if value.trim().is_empty() {
                    return Err(Error::InvalidStructuredOutputsParams {
                        message: "structured_outputs.json cannot be an empty string".to_string(),
                    });
                }
                serde_json::from_str::<Value>(value).map_err(|_| {
                    Error::InvalidStructuredOutputsParams {
                        message: "Invalid JSON grammar specification.".to_string(),
                    }
                })?;
            }
        }
        Ok(())
    }
}
```

```

    }
    // 空候选列表无法构造任何合法 token 集合，直接拒绝。
    StructuredOutputConstraint::Choice(choice) if choice.is_empty() => {
        Err(Error::InvalidStructuredOutputsParams {
            message: "structured_outputs.choice cannot be empty".to_string(),
        })
    }
}
// structural_tag 整体是一段 JSON 字符串，空白同样视为无效输入。
StructuredOutputConstraint::StructuralTag(tag) if tag.trim().is_empty() => {
    Err(Error::InvalidStructuredOutputsParams {
        message: "structured_outputs.structural_tag cannot be an empty string"
            .to_string(),
    })
}
// json_object、非空 json 对象以及 regex 走默认放行分支。
_ => Ok(),
}
}
}

```

Serde / HTTP 请求体路径的接入点：

```

impl TryFrom<WireStructuredOutputsParams> for StructuredOutputsParams {
    type Error = Error;

    fn try_from(raw: WireStructuredOutputsParams) -> Result<Self> {
        use StructuredOutputConstraint::*;
        // 上文由 insert_constraint! 宏逐个收集 json / regex / choice / grammar /
        // structural_tag，并在出现第二个约束时报「multiple structured output constraints」
        // （细节从略，本 PR 未改动该逻辑）。

        let params = Self {
            // 缺失任何约束时的原有文案保持不变，避免影响既有测试与客户端的错误提示。
            constraint: constraint.map(|_, c| c).ok_or_else(|| {
                Error::InvalidStructuredOutputsParams {
                    message: "missing structured output constraint".to_string(),
                }
            })?,
            options: StructuredOutputOptions { /* 各字段由 raw 原样透传 */ },
            backend: raw.backend,
        };
        // 关键变更：请求体反序列化成功后立即校验；失败即 400，
        // 不会再落到语法编译阶段才以「按请求失败」的形式暴露。
        params.validate()?;
        Ok(params)
    }
}

```

[rust/src/server/src/grpc/convert.rs](#)

gRPC 入口: `convert_structured_output` 在返回参数前显式调用 `validate()`, 修复 Codex 标记的 P1 漏检路径 (构造函数不经过 `TryFrom`), 并新增 `grpc_rejects_empty_grammar_before_engine` 用例。

```
/// 把 gRPC 的结构化输出 oneof 映射为 engine-core 参数对象。
/// 早前这条路径直接使用无校验的构造函数 (`from_constraint`),
/// 绕过了 Serde 侧的 `TryFrom` 校验, 是 Codex 标记的 P1 漏检点。
fn convert_structured_output(
    d: &pb::DecodingParameters,
) -> Result<Option<StructuredOutputsParams>, Status> {
    let so = match d.structured_output.as_ref() {
        None => return Ok(None),
        Some(so) => so,
    };
    use pb::decoding_parameters::StructuredOutput;
    let params = match so {
        // Json 分支自身已把字符串解析为 serde_json::Value,
        // 解析失败会先以 "invalid json schema" 返回, 不依赖本轮新增校验。
        StructuredOutput::Json(schema) => {
            let json: serde_json::Value = serde_json::from_str(schema)
                .map_err(|_| Status::invalid_argument(format!("invalid json schema: {e}")))?;
            StructuredOutputsParams::json(json)
        }
        StructuredOutput::Regex(regex) => StructuredOutputsParams::regex(regex.clone()),
        StructuredOutput::Choice(choices) => {
            StructuredOutputsParams::choice(choices.choices.clone())
        }
        // 空字符串 / 纯空白语法在这里不再被无条件放行。
        StructuredOutput::Grammar(grammar) => StructuredOutputsParams::grammar(grammar.clone()),
        StructuredOutput::JsonObject(true) => StructuredOutputsParams::json_object(),
        StructuredOutput::JsonObject(false) => return Ok(None),
        StructuredOutput::StructuralTag(tag) => {
            StructuredOutputsParams::structural_tag(tag.clone())
        }
    };
    // 与 HTTP 路径共用同一个 `validate()` , 把失败前置为 INVALID_ARGUMENT,
    // 而不是让引擎在生成阶段以 per-request 错误的形式返回。
    params
        .validate()
        .map_err(|error| Status::invalid_argument(error.to_report_string()))?;
    Ok(Some(params))
}
```

`rust/src/server/src/routes/openai/utils/structured_outputs.rs`

OpenAI `response_format` 入口: `convert_from_response_format` 改为先构造参数再统一 `validate()`, 补上 agentic review 指出的空 / 空白 schema 漏检路径。

```
/// 把类型化的 `response_format` 与原始 `structured_outputs` blob 转换为
```

```

/// engine-core 的 `StructuredOutputsParams` (对齐 Python 前端的转换语义)。
pub fn convert_from_response_format(
    response_format: Option<&ResponseFormat>,
    structured_outputs: &Option<serde_json::Value>,
) -> Result<Option<StructuredOutputsParams>, ApiError> {
    // 显式 structured_outputs 走 Serde, 校验已在 `TryFrom` 内部完成。
    if let Some(raw) = structured_outputs {
        return Ok(Some(deserialize_structured_outputs(raw)?));
    }

    let Some(fmt) = response_format else {
        return Ok(None);
    };
    // 改动点: 不再在 match 分支里直接 Ok(Some(...)),
    // 而是先构造出 params, 再在函数末尾统一校验。
    let params = match fmt {
        ResponseFormat::Text => return Ok(None),
        ResponseFormat::JsonObject => StructuredOutputsParams::json_object(),
        ResponseFormat::JsonSchema { json_schema } => {
            // schema 为字符串时, 空白或非 JSON 文本会被 validate() 拦下。
            StructuredOutputsParams::json(json_schema.schema.clone())
        }
        ResponseFormat::StructuralTag { .. } => {
            // Python 前端会把整个 response_format (含 type 字段) 序列化成字符串下发。
            let tag_json = serde_json::to_string(fmt).map_err(|e| {
                ApiError::invalid_request(
                    format!("failed to serialize structural_tag: {e}"),
                    Some("response_format"),
                )
            })?;
            StructuredOutputsParams::structural_tag(tag_json)
        }
    };
    // 非法约束在这里变成 400, 而不是等到语法编译阶段才暴露给调用方。
    params.validate().map_err(|error| {
        ApiError::invalid_request(error.to_report_string(), Some("response_format"))
    })?;
    Ok(Some(params))
}

```

评论区精华

「For a gRPC request with `DecodingParameters.structured_output.grammar` set to an empty or whitespace-only string, this guard is never reached... route the gRPC conversion through the same validation or a shared fallible constructor.」

——chatgpt-codex-connector[bot], P1

「This is right suggestion. @KernelClint, please address it.」 ——@arpera

「It looks like description is a bit wrong... There is no engine abortion, please fix it.」——@arpera（作者随后修正为「Grammar-compile failures are carried per-request to the scheduler」）

「The PR tests the validator and gRPC conversion, but not the Serde/HTTP path that motivated the fix. It should assert a 400 before scheduling.」——@vadiklyutiy 要求补 whitespace-only grammar 的 HTTP 回归测试

「LGTM I'd prefer to also get approval from @BugenZhao or @njhill as owners of rust frontend」——@vadiklyutiy 的合并意见

三条讨论线（gRPC 漏检、描述纠错、HTTP 补测）都由「外部审查」驱动：Codex 机器人抓构造函数绕过，@arpera 的 agentic review 抓 `response_format` 路径，@vadiklyutiy 抓测试缺口，作者逐条响应并在回复中确认修复点。

风险与影响

- 兼容性收紧：convert_from_response_format 现在对 ResponseFormat::JsonSchema 也执行 validate()，若客户端把 schema 传成空白字符串或非 JSON 文本，此前会被放行、现在直接 400。这是有意的契约对齐，但对依赖旧宽松行为的调用方属于行为变化。
- 多入口一致性依赖人工维护：构造函数仍是不可失败的 from_constraint，新增任何入口都需要记得调用 validate()。本次 review 中这类遗漏出现两次（gRPC、response_format），说明该模式虽清晰但存在复发风险。
- 校验范围不对称：validate() 覆盖 grammar / json / choice / structural_tag，regex 未做空白检查（落入 _ => Ok()）。是否需要与 Python 前端进一步对齐，材料中未讨论。
- 测试层级：新增用例全部在源文件 mod tests 内，HTTP 侧用例调用 prepare_chat_request 而非真实 axum 路由，因此「400 before scheduling」更多是函数级保证，缺少端到端集成覆盖。
- 影响面：仅涉及 Rust 前端服务端与 engine-core-client 协议层，不触及模型执行、注意力内核、KV 缓存等运行时核心路径，性能与时延无实质影响；收益是减少无效请求进入调度器与语法编译。

关联脉络

- #51450：本 PR 由其拆分而来，说明结构化输出校验属于一个更大的前端改造计划，本 PR 只承担「边界校验」这一小块以便独立合并。
- #56405 / #55047：同属 rust/src/server/src/grpc/convert.rs 这一 gRPC 转换 / 错误面收敛线，前者规范引擎错误的对外码，本 PR 则把非法输入前置为 Status::invalid_argument，方向一致：让 gRPC 客户端拿到语义正确的状态码。
- #56386 / #56378：Rust 前端的输入参数处理（HF revision、离线与缓存目录、chat template 的 generation 块）持续补齐，与本 PR 一样都在缩小 Rust 前端与 Python 前端的输入契约差异。
- 审查模式：与 #56546 等 PR 类似，本仓库的 Rust 前端改动大量依赖 Codex 机器人与 @arpera 组织的 agentic review 补足人工审查盲区，本 PR 是该模式收益最直观的案例之

