PR #47494 完整报告

vllm-project/vllm

[Rust Frontend] Align sampling validation with Python

合并时间：2026-07-28 20:02

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47494

执行摘要

Rust 前端新增了对 OpenAI 采样参数（temperature、top_p、min_p、frequency_penalty、presence_penalty、repetition_penalty）的验证，使无效值在请求进入引擎前即返回 400 错误，与 Python 前端行为一致。该验证被置于 text 层的降层过程中，因此 HTTP 和 gRPC 等所有入口均自动受益。

功能与动机

此前 Rust 前端直接将采样参数透传给 Python 引擎，未做范围检查。例如 temperature=5.0、top_p=0.0、min_p=2.0 会直达引擎并触发解码层错误，而不是像 Python 前端那样返回清晰的 400 响应。本次 PR 填补了这一空白，提升了 API 的一致性和用户体验。

实现拆解

1. 新增验证子模块(rust/src/text/src/lower/sampling.rs)：定义了 SamplingParamsError 枚举和七个验证函数，每个参数使用独立函数（validate_temperature 等），并提供了统一的入口 validate_resolved_sampling_params。
2. 集成到降层流程(rust/src/text/src/lower.rs)：在 lower_sampling_params 函数构造 EngineCoreSamplingParams 之后、返回之前插入 validate_resolved_sampling_params(&params)?;，使所有调用路径自动校验。
3. 错误类型传播(rust/src/text/src/error.rs, rust/src/text/src/lib.rs)：在 Error 枚举中新增 SamplingParams 变体，并在 lib 中公开导出 SamplingParamsError。
4. 服务层错误映射测试(rust/src/server/src/error.rs)：新增测试确保 SamplingParamsError::OutOfRange 被映射为 HTTP 400 和 invalid_request_error。
5. gRPC 集成测试(rust/src/server/src/grpc/tests.rs)：发送非法 top_p=2.0 验证 gRPC 端点返回 InvalidArgument 并包含错误信息。

rust/src/text/src/lower.rs

验证被嵌入到降层流程中的关键位置，确保所有端点共享

```
// rust/src/text/src/lower.rs（相关改动部分）

// 在 lower_sampling_params 末尾，EngineCoreSamplingParams 构造完成后立即验证
pub fn lower_sampling_params(...) -> Result<EngineCoreSamplingParams> {
    // ... 前面构造 params ...
    let params = EngineCoreSamplingParams {
```

```
        temperature,
        top_p,
        min_p,
        frequency_penalty,
        presence_penalty,
        repetition_penalty,
        // ... 其他字段 ...
    };
    // [!] 新增：在 vocab 范围校验之前，先校验采样参数本身的合法性
    validate_resolved_sampling_params(&params)?;
    validate_vocab_range(&params, &sampling_limits)?;
    Ok(params)
}

#[cfg(test)]
mod tests {
    // ...

    #[test]
    fn lower_sampling_params_rejects_invalid_sampling_ranges() {
        let cases = [
            ("temperature", SamplingParams { temperature: Some(5.0), ..Default::default() }),
            ("top_p", SamplingParams { top_p: Some(0.0), ..Default::default() }),
            ("min_p", SamplingParams { min_p: Some(2.0), ..Default::default() }),
            ("repetition_penalty", SamplingParams { repetition_penalty: Some(0.0), ..Default::
            default() }),
            ("frequency_penalty", SamplingParams { frequency_penalty: Some(100.0), ..Default::
            default() }),
            ("presence_penalty", SamplingParams { presence_penalty: Some(100.0), ..Default::
            default() }),
        ];

        for (expected_parameter, sampling_params) in cases {
            let error = lower_sampling_params_with_limits(sampling_params, sample_sampling_
            limits())
                .unwrap_err();
            assert!(
                matches!(error, Error::SamplingParams(SamplingParamsError::OutOfRange {
                parameter, .. }) if parameter == expected_parameter),
                "{expected_parameter} should be rejected"
            );
        }
    }

    #[test]
    fn lower_sampling_params_rejects_non_finite_sampling_values() {
        // 验证 Infinite 和 NaN 均被拒绝
        for (expected_parameter, sampling_params) in [
            ("temperature", SamplingParams { temperature: Some(f32::INFINITY), ..Default::
```

```
            default() }),
            ("repetition_penalty", SamplingParams { repetition_penalty: Some(f32::NAN), ..Default::
            default() }),
        ] {
            let error = lower_sampling_params_with_limits(sampling_params, sample_sampling_
            limits())
                .unwrap_err();
            assert!(
                matches!(error, Error::SamplingParams(SamplingParamsError::NotFinite {
                parameter, .. }) if parameter == expected_parameter),
                "{expected_parameter} should reject non-finite values"
            );
        }
    }

    #[test]
    fn lower_sampling_params_accepts_python_compatible_repetition_penalty_above_two() {
        // Python 对 repetition_penalty 无上限，2.5 应被接受
        let params = lower_sampling_params_with_limits(
            SamplingParams { repetition_penalty: Some(2.5), ..Default::default() },
            sample_sampling_limits(),
        ).expect("repetition_penalty > 2 should be accepted");
    }
}
```

rust/src/server/src/grpc/tests.rs

验证 gRPC 端点也能正确拒绝无效采样参数

```
// rust/src/server/src/grpc/tests.rs ( 新增测试 )

#[tokio::test(flavor = "multi_thread", worker_threads = 2)]
#[serial]
async fn unary_generate_invalid_sampling_params_returns_invalid_argument() {
    let (mut client, server_task, _engine_task) = grpc_test_server(
        b"engine-grpc-invalid-sampling",
        default_stream_output_specs(),
    ).await;

    // 发送 top_p = 2.0（超出 (0,1] 范围）的请求，期望返回 InvalidArgument
    let status = client
        .generate(pb::GenerateRequest {
            request_id: "test-invalid-sampling".to_string(),
            model: "test-model".to_string(),
            prompt: Some(pb::generate_request::Prompt::Text("hi".to_string())),
            sampling: Some(pb::RandomSampling {
                top_p: 2.0,
                ..Default::default()
            }),
            ..Default::default()
```

```
        })
        .await
        .expect_err("should fail when top_p is out of range");

    assert_eq!(status.code(), tonic::Code::InvalidArgument);
    assert!(status.message().contains("top_p"));

    server_task.abort();
}
```

评论区精华

> BugenZhao: "In the long term, I think it would be better to move the validation logic down to the text or llm layer, so that all endpoints can benefit from it and there is no risk of drift." 作者随后将验证下沉至 lower 模块，实现了统一校验。
>
> BugenZhao: "Shall we do this after EngineCoreSamplingParams is constructed, so that we don't have to pass different parameters independently here?" 作者采纳建议，将调用移至参数构造完成后。
>
> cinnamonica02: 指出 gRPC 的 build_sampling_params 缺少范围检查。 作者回应新的验证位于底层，gRPC 路径同样经过 lower_sampling_params，自动受益。

风险与影响

- 低风险：所有校验规则与 Python 端一致，单元测试覆盖了范围边界、非有限值和 repetition_penalty 无上限场景，gRPC 集成测试覆盖了非法值场景。
- 正面影响：用户将收到清晰的 400 错误，而非引擎内部 500；无效请求更早被拒绝，节省引擎计算资源；统一校验降低了前端与 Python 行为偏离的风险。

关联脉络

无直接强关联的历史 PR。本 PR 是 Rust 前端系列改进（如 #49496、#49992）的一部分，逐步缩小与 Python 前端的差距，提升 API 健壮性。