

PR #51026 完整报告

vllm-project/vllm

[Bugfix][Rust Frontend] Tolerate NaN-corrupted logprobs in engine-core

合并时间: 2026-09-16 11:10

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/51026>

[Bugfix][Rust Frontend] Tolerate NaN-corrupted logprobs in engine-core

执行摘要

本 PR 修复 Rust 前端 engine-core 客户端的一个服务级故障: 当模型产生 NaN logits 时, 采样器计算出的 sampled rank 为 0, 而客户端解码器将 rank 0 视为坏数据并在共享输出读取路径上整体拒绝编码帧, 导致单个请求即可让 `/health` 永久返回 503、所有后续请求失败直至重启。修复方案是让解码层接受并保留 rank 0 与原始 logprob 值, 将 NaN/-inf 到 `-9999.0` 的 sentinel 转换留在 OpenAI 响应序列化层 (该逻辑已存在), 配套两个回归测试锁定层边界。

功能与动机

PR body 明确指出这是一个 total-outage bug: `a single request asking for logprobs/prompt_logprobs while the model produces NaN logits permanently kills the client.`

根因链路有三环:

1. 引擎侧合法输出: `vllm/v1/sample/sampler.py` 将采样 rank 计算为 `(logprobs >= sampled_logprob).sum(-1)`, 当 sampled logprob 为 NaN 时, IEEE-754 规定 NaN 与任何值比较均为 false, rank 恒为 0。
2. 客户端误判: `PositionLogprobs::from_decoded_row` 把 `sampled_rank == 0` 判为 wire 数据损坏并 `bail_ext_value_decode!`。
3. 故障放大: 失败是 frame-scoped 的, 一个坏行导致整批 `EngineCoreOutputs` 解码失败, 且发生在共享 `output-reader` 路径上, 被记录为 sticky 健康错误——单请求数据异常演变为服务器永久不可用。

实现拆解

1. 根因定位: 从 Python 采样器 rank 计算出发, 确认 NaN 语义下 rank 0 是引擎合法输出, 再追踪到 Rust 侧 `rust/src/engine-core-client/src/protocol/logprobs.rs` 的拒绝逻辑。
2. 解码层修复: 在 `PositionLogprobs::from_decoded_row` 中删除 `if sampled_rank == 0 { bail_ext_value_decode!(...) }`, 保留引擎原始 logprob 值; 为 `TokenLogprob` 的 rank 字段补充 NaN 语义文档说明。
3. 设计收敛: 首个 commit (07be0300) 曾尝试把 rank 0 与 NaN 值在解码层直接 clamp 为 `-9999.0`; BugenZhao review 后第二个 commit (3344086d) 改为保留原值,

sentinel 转换统一收口到 OpenAI 序列化层，形成“解码层保真、序列化层转换”的分层约定。

4. 测试配套：zero_sampled_rank_does_not_fail_frame_mates 覆盖批量帧中坏行不拖累健康请求；chat_logprobs_clamps_nan_and_negative_infinity_for_json 锁定 HTTP 边界 -9999.0 行为。测试结果：cargo nextest 118 passed, clippy/fmt 通过，无需 GPU。

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

核心修复文件：移除 `PositionLogprobs::from_decoded_row` 中对 sampled rank 0 的拒绝校验，并补充 TokenLogprob 的 NaN 语义文档，是消除服务级故障的关键改动。

```
// rust/src/engine-core-client/src/protocol/logprobs.rs
// PositionLogprobs::from_decoded_row: 将一行 logprobs wire 数据转换为按位置分组的形式。
impl PositionLogprobs {
    fn from_decoded_row(token_ids: &[u32], logprobs: &[f32], sampled_rank: u32) ->
    Result<Self> {
        // 先校验 token_ids 与 logprobs 长度一致，防止 wire 数据被截断导致错位。
        if token_ids.len() != logprobs.len() {
            bail_ext_value_decode!(
                "logprobs row length mismatch: token_ids={}, logprobs={}",
                token_ids.len(),
                logprobs.len()
            );
        }

        // 注意：这里不再拒绝 sampled_rank == 0。
        // 引擎在 vllm/v1/sample/sampler.py 中把采样 rank 计算为
        // `(logprobs >= sampled_logprob).sum(-1)`；当 sampled_logprob 为 NaN 时，
        // IEEE-754 规定 NaN 与任何值的比较都为 false，因此 rank 恒为 0。
        // 这是引擎的合法输出而非 wire 数据损坏：如果在此处 bail，
        // 整个批次的 EngineCoreOutputs 帧都会解码失败（frame-scoped），
        // 进而触发共享输出读取路径的 sticky 健康错误，导致服务级故障。
        let mut entries = Vec::with_capacity(token_ids.len());
        for (index, (&token_id, &logprob)) in token_ids.iter().zip(logprobs.iter()).enumerate() {
            // 第一个条目是采样 / 选中 token，携带引擎返回的真实 vocab rank；
            // 其余条目按引擎返回的 top-k 候选顺序编号（从 1 开始）。
            let rank = if index == 0 {
                sampled_rank
            } else {
                index as u32
            };
            entries.push(TokenLogprob {
                token_id,
                // 保留引擎原始数值（含 NaN 与正负无穷）：
                // sentinel 转换（-9999.0）统一由 HTTP 序列化层
                // 在 rust/src/server/src/routes/openai/utls/logprobs.rs 中完成。
                logprob,
                rank,
            });
        }
    }
}
```

```

        Ok(Self { entries })
    }
}

```

rust/src/engine-core-client/src/protocol/logprobs/tests.rs

新增核心回归测试 `zero_sampled_rank_does_not_fail_frame_mates` 与 wire 构造辅助 `logprobs_value`，验证 NaN/-inf/rank 0 的坏行不再拖垮同批次健康请求。

```

// rust/src/engine-core-client/src/protocol/logprobs/tests.rs
// logprobs_value: 把 token_ids / probs / ranks 组装成引擎 wire 层的
// ndarray 元组 (ids 行 x 列、probs 行 x 列、ranks 每行一个) ,
// 用于构造 EngineCoreOutputs 帧。
fn logprobs_value(ids: &[i64], probs: &[f32], ranks: &[i64]) -> Value {
    let rows = ranks.len();
    let cols = ids.len() / rows;
    Value::Array(vec![
        ndarray_value(
            "<i8",
            &[rows, cols],
            Value::Ext(3, ids.iter().flat_map(|v| v.to_le_bytes()).collect()),
        ),
        ndarray_value(
            "<f4",
            &[rows, cols],
            Value::Ext(3, probs.iter().flat_map(|v| v.to_le_bytes()).collect()),
        ),
        ndarray_value(
            "<i8",
            &[rows],
            Value::Ext(3, ranks.iter().flat_map(|v| v.to_le_bytes()).collect()),
        ),
        Value::Nil,
    ])
}

```

```

#[test]
fn zero_sampled_rank_does_not_fail_frame_mates() {
    // 构造一个共享 EngineCoreOutputs 批次: req-nan 的 sample 与 prompt
    // logprobs 均为 NaN/-inf 且 rank 为 0, req-ok 是完全正常的数据行。
    let frames = vec![Bytes::from(encode_value(&Value::Array(vec![
        Value::from(0),
        Value::Array(vec![
            Value::Array(vec![
                Value::from("req-nan"),
                Value::Array(vec![Value::from(7), Value::from(8)]),
                logprobs_value(&[1, 2], &[f32::NAN, f32::NEG_INFINITY], &[0]),
                logprobs_value(&[7, 8], &[f32::NAN, f32::NEG_INFINITY], &[0]),
                Value::Nil,
                Value::from(EngineCoreFinishReason::Length as u8),
            ]),
        ]),
    ]))

```

```

    ]),
    Value::Array(vec![
        Value::from("req-ok"),
        Value::Array(vec![Value::from(9)]),
        logprobs_value(&[3, 4], &[-3.0, -0.5], &[7]),
        Value::Nil,
        Value::Nil,
        Value::from(EngineCoreFinishReason::Length as u8),
    ]),
    ],
    Value::Nil,
    Value::from(0.0),
    Value::Nil,
    Value::Nil,
    ]))];

```

```

// 修复前：整批解码返回 ExtValueDecode 错误，healthy 请求一起失败；
// 修复后：rank 0 与 NaN/-inf 原样保留，healthy 行 (-3.0/-0.5、rank 7/1)
// 不受影响。expect_test 快照断言三个 Logprobs 的完整内容：
// - bad 行：TokenLogprob { token_id: 1, logprob: NaN, rank: 0 },
// TokenLogprob { token_id: 2, logprob: -inf, rank: 1 }
// - prompt 行：TokenLogprob { token_id: 7, logprob: NaN, rank: 0 },
// TokenLogprob { token_id: 8, logprob: -inf, rank: 1 }
// - good 行：TokenLogprob { token_id: 3, logprob: -3.0, rank: 7 },
// TokenLogprob { token_id: 4, logprob: -0.5, rank: 1 }
let decoded = decode_engine_core_outputs(&frames).unwrap().into_request_batch().unwrap();
let bad = decoded.outputs[0].new_logprobs.as_ref().unwrap();
let prompt = decoded.outputs[0].new_prompt_logprobs_tensors.as_ref().unwrap();
let good = decoded.outputs[1].new_logprobs.as_ref().unwrap();
expect_test::expect![[r#"
    (
        Logprobs { /* ... 见上方注释 ... */ },
        Logprobs { /* ... 见上方注释 ... */ },
        Logprobs { /* ... 见上方注释 ... */ },
    )
"#]]
.assert_debug_eq(&(&**bad, &**prompt, &**good));
}

```

rust/src/server/src/routes/openai/utils/logprobs.rs

在 OpenAI 响应转换层补充测试，锁定既有 **-9999.0** sentinel 行为，为解码层保留 NaN/-inf 提供序列化安全网。

```

// rust/src/server/src/routes/openai/utils/logprobs.rs
// 验证 OpenAI chat 响应边界对 NaN 与负无穷的既有 sentinel 转换：
// 解码层保留的 NaN/-inf 在这里统一变成 -9999.0，避免 JSON 出现非法数值。
#[test]
fn chat_logprobs_clamps_nan_and_negative_infinity_for_json() {
    let mut logprobs = sample_logprobs();

```

```

logprobs.positions[0].entries[0].logprob = f32::NAN;
logprobs.positions[0].entries[0].rank = 0;
logprobs.positions[0].entries[1].logprob = f32::NEG_INFINITY;

let response = decoded_logprobs_to_openai_chat(&logprobs, -1, false).unwrap();
let json = serde_json::to_string_pretty(&response).unwrap();

// 快照预期: content[0].logprob 与 top_logprobs 中 A/B 均为 -9999.0,
// 有限值 C = -2.0 保持不变, bytes 65/66/67 对应 "A"/"B"/"C"。
expect_test::expect![[r#"
{
  "content": [
    {
      "token": "A",
      "logprob": -9999.0,
      "bytes": [65],
      "top_logprobs": [
        { "token": "A", "logprob": -9999.0, "bytes": [65] },
        { "token": "B", "logprob": -9999.0, "bytes": [66] },
        { "token": "C", "logprob": -2.0, "bytes": [67] }
      ]
    }
  ]
}
"#]]
.assert_eq(&json);
}

```

评论区精华

BugenZhao (review) : I think this is not necessary? We can preserve the original value within the frontend pipeline, but only clamp it when it's going to be serialized in the serving layer, which we already have

这是本 PR 最有价值的交锋：第一版实现试图在解码层把非有限 logprobs 提前映射为 `-9999.0`，BugenZhao 指出一是重复实现（serving 层已有 clamp），二是丢失引擎原始数值信息。最终采纳该意见，第二个 commit 保留 rank 0 与原始 NaN/-inf 值。Codex 自动 review 无重大发现，BugenZhao 最终 APPROVED。

风险与影响

- rank 0 流入下游：解码校验放宽后，任何假设 rank ≥ 1 的下游逻辑需兼容 rank 0；当前 chat 路径按候选顺序展示而不依赖 rank，风险有限，但流式与非 chat 路径暂无显式测试。
- NaN 序列化覆盖不全：测试只覆盖 chat JSON 路径；若 completions、streaming 等路径未 clamp，serde_json 可能输出非标准 JSON。
- 根因未消除：引擎层 NaN logits 的产生根源仍在，本修复只把故障从“服务级崩溃”降级为“单请求数据异常”，与 Python 前端 #48585 的容错边界保持一致。

关联脉络

本 PR 处于 vLLM Rust 前端容错能力建设的一条明确线上：与 #48585 (Python 前端非有限 logprobs 修复) 互为同类缺陷的跨语言收敛，与 #50725/#50002/#50283 共享“单点数据异常放大为进程级故障”的 blast-radius 模式。同仓库近期 #56990 (Rust 前端迭代 token 直方图) 也在 `rust/engine-core-client` 与 `rust/server` 模块上持续演进，说明 Rust 前端正在从“功能补全”走向“生产级稳定性加固”。修复中确立的“解码层保真、序列化层转换”分层约定，可作为后续 Rust 前端处理非有限数值的统一原则。