

PR #56405 完整报告

vllm-project/vllm

[Rust Frontend][gRPC] Surface engine generation errors

合并时间: 2026-09-12 06:31

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56405>

执行摘要

PR #56405 修复了 Rust 前端 gRPC 服务中“引擎故障被当成成功响应”的问题: 此前 `FinishReason::Error` 与 `Abort` 一样被映射为 `ABORTED`, 客户端无法区分真实故障与正常终止; 本次改为在 `Generate` 与 `GenerateStream` 两个 RPC 上以 `Status::internal` 暴露引擎错误, 流式场景保留已投递输出后报错终止, 客户端取消语义不变。改动集中在 `convert.rs` 的 Result 化转换与 `inference.rs` 两条 RPC 路径的适配, 并新增覆盖 unary/streaming 及“错误发生在部分输出前后”两种时序的回归测试, 无 protobuf 变更。

功能与动机

PR body 明确说明问题: "Engine failures currently return a successful gRPC response with an `ABORTED` finish reason. Return `INTERNAL` instead for both `Generate` and `GenerateStream`." 即引擎失败时客户端拿到的是“成功 + `ABORTED`”, 与正常取消 (`Abort`) 无法区分, 错误详情也被吞掉。本 PR 修正现存 gRPC 的 `Error` → `Aborted` 映射, 并作为 #55399 (引擎状态拒绝与 HTTP 层处理) 的配套修复, 让错误从引擎一路传导到 gRPC 状态码。

实现拆解

- 转换层语义变更 (`convert.rs`): `to_finish_info` 返回值从 `pb::FinishInfo` 改为 `Result<pb::FinishInfo, Status>`, 在 `FinishReason::Error` 分支直接 return `Err(Status::internal("engine failed during generation"))`; `Abort` 与 `Repetition` 仍映射为 `ABORTED`, 保持取消语义。`to_sequence_output` 同步改为返回 `Result`, 并在组装 proto 前先通过 `transpose()`? 转换 `finish_info`, 使错误能在源头被捕获。
- Unary 路径适配 (`inference.rs` 的 `generate`): 原“请求完成”日志移到 `to_sequence_output(...)?` 之后, ? 把引擎错误帧直接变成 RPC 层 `INTERNAL` 失败响应, 错误发生在聚合阶段时同样生效。
- Streaming 路径适配 (`inference.rs` 的 `generate_stream`): `TextDelta` 分支改为先经 `to_sequence_output` 的 `Result` 映射再包装成 `GenerateResponse`; 循环内新增 `failed = response.is_err()` 判断, `tx.send` 后一旦发现错误帧立即 `break`, 停止继续拉取引擎流——已投递的输出保持不变, `tonic` 以最后一个 `Err` 作为流终止状态码。
- 回归测试 (`tests.rs`): 新增 `engine_error_finish_returns_internal_for_unary_and_streaming`, 以 `streaming × prefix` 双重循环覆盖 unary/streaming 两条 RPC 以及错误发生在部分输出之前 / 之后两种时序, 断言 unary `expect_err`、streaming 先收到已交付 token 再收到 `tonic::Code::Internal`, 且错误帧不会变成 finish event; 既有 `convert.rs` 单测调

用同步改为 expect。

5. 配套验证：PR body 报告在 vLLM 0.29 引擎与 Rust backport 下实测 Qwen3-0.6B：native gRPC 返回 INTERNAL，下游 HTTP/SSE 报错后紧跟 [DONE]，后续请求正常；cargo test --locked -p vllm-server --lib grpc:: 48 项通过，无 proto/schema/ 配置变更。

rust/src/server/src/grpc/convert.rs

核心语义变更所在：to_finish_info 将 FinishReason::Error 从 ABORTED 映射改为返回 Status::internal，并带动 to_sequence_output Result 化，是本次改动的语义源头。

```
/// 将 `DecodedTextEvent::TextDelta` 转换为 gRPC `SequenceOutput`。
///
/// 返回 `Result` 而不是直接返回 proto，是为了让引擎的 `Error` 结束能够沿
/// RPC 边界以 gRPC `INTERNAL` 状态码向上传播，而不是伪装成一次成功完成。
pub fn to_sequence_output(
    delta: &str,
    token_ids: &[u32],
    logprobs: Option<&DecodedLogprobs>,
    finished: Option<&Finished>,
    opts: &ResponseOpts,
) -> Result<pb::SequenceOutput, Status> {
    // 先转换 finish_info: 若引擎以 Error 结束，这里直接返回 Err，
    // 后续代码链路会把它变成 RPC 错误而不是 finish event。
    let finish_info = finished.map(|f| to_finish_info(f, token_ids)).transpose()?;
    let (lp_values, rank_values, candidates) = match logprobs {
        Some(lp) if opts.output_logprobs => output_logprobs_to_proto(lp),
        _ => (vec![], vec![], vec![]),
    };

    Ok(pb::SequenceOutput {
        index: 0, // TODO: multi-sequence (n > 1) 暂不支持
        text: if opts.output_text {
            delta.to_string()
        } else {
            String::new()
        },
        num_tokens: token_ids.len() as u32,
        token_ids: if opts.output_token_ids {
            token_ids.to_vec()
        } else {
            vec![]
        },
        logprobs: lp_values,
        ranks: rank_values,
        candidate_tokens: candidates,
        finish_info,
    })
}
```

/// 将 `Finished` 转换为 proto `FinishInfo`。

```

///
/// `FinishReason::Error` 是本次变更的唯一分支：此前它与 `Abort` 一起被映射为
/// `ABORTED`，导致客户端无法区分“引擎真的失败”和“请求被正常终止”；现在改为
/// 返回 `Status::internal`。`Abort`（含客户端取消）与 `Repetition` 仍映射为
/// `ABORTED`，保持既有取消语义不变。
fn to_finish_info(finished: &Finished, token_ids: &[u32]) -> Result<pb::FinishInfo, Status> {
    use pb::finish_info::FinishReason as PbFinishReason;

    let (finish_reason, stop_reason) = match &finished.finish_reason {
        FinishReason::Stop(reason) => {
            let sr = match reason {
                Some(StopReason::TokenId(id)) => {
                    Some(pb::finish_info::StopReason::StopTokenId(*id))
                }
                Some(StopReason::Text(s)) => {
                    Some(pb::finish_info::StopReason::StopString(s.clone()))
                }
                // EOS 驱动的 stop: engine-core 匹配到主 EOS token id 但未回显为
                // stop_reason；匹配到的 token 按构造必然是终止输出批次里的最后一个
                // token（参见 vllm 的 check_stop, vllm/v1/core/sched/utils.py），
                // 所以直接从 token_ids 尾部恢复它。
                None => token_ids.last().copied().map(pb::finish_info::StopReason::EosTokenId),
            };
            (PbFinishReason::Stop as i32, sr)
        }
        FinishReason::Length => (PbFinishReason::Length as i32, None),
        // 引擎失败：不再生成成功的 finish event，直接以 INTERNAL 终止 RPC。
        FinishReason::Error => return Err(Status::internal("engine failed during generation")),
        FinishReason::Abort | FinishReason::Repetition(_) => (PbFinishReason::Aborted as i32,
            None),
    };

    Ok(pb::FinishInfo {
        num_output_tokens: finished.usage.output_token_count as u32,
        finish_reason,
        stop_reason,
        kv_transfer_params: finished.kv_transfer_params.as_ref().and_then(json_to_proto_struct),
        ec_transfer_params: finished.ec_transfer_params.as_ref().and_then(json_to_proto_struct),
    })
}

```

rust/src/server/src/grpc/inference.rs

Generate 与 **GenerateStream** 两条 RPC 路径适配 Result 化转换，确定流式错误终止语义，是行为变更的落点。

```

tokio::spawn(
    async move {
        futures::pin_mut!(stream);
        while let Some(event) = stream.next().await {

```

```

let response = match event {
  Err(error) => Err(log_text_error(&task_span, started_at, "stream", error)),
  Ok(DecodedTextEvent::Start {
    prompt_token_ids,
    prompt_logprobs,
  }) => {
    let prompt_info = convert::to_prompt_info(
      &prompt_token_ids,
      prompt_logprobs.as_ref(),
      &response_opts,
    );
    Ok(pb::GenerateResponse {
      prompt_info: Some(prompt_info),
      outputs: None,
    })
  }
  Ok(DecodedTextEvent::TextDelta {
    decoded,
    sampled: SampledDelta { token_ids, logprobs },
    finished,
  }) => convert::to_sequence_output(
    &decoded.text,
    &token_ids,
    logprobs.as_ref(),
    finished.as_deref(),
    &response_opts,
  )
  // 引擎 Error 会在这里变成 Err，交由下方逻辑终止流。
  .map(|outputs| pb::GenerateResponse {
    prompt_info: None,
    outputs: Some(outputs),
  }),
};

// 一旦本帧已经是一个错误帧，就停止继续拉取引擎流：
// 已投递给客户端的输出保持不变，随后 channel 关闭，
// tonic 会把最后一个 Err 作为流式调用的终止状态码返回。
let failed = response.is_err();
if tx.send(response).await.is_err() || failed {
  break;
}
}
info!(
  parent: &task_span,
  elapsed_ms = started_at.elapsed().as_millis() as u64,
  "gRPC inference stream closed"
);
}
.instrument(request_span),

```

```
);
```

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

新增回归测试覆盖 unary/streaming 双路径及错误发生在部分输出前后的时序，是本次语义变更的验证基线。

```
#[tokio::test(flavor = "multi_thread", worker_threads = 2)]
#[serial]
async fn engine_error_finish_returns_internal_for_unary_and_streaming() {
    // 双重遍历：先测 unary / streaming 两种 RPC，再测错误发生在部分输出前（prefix 为空）
    // 与部分输出后（prefix 为单个 token）两种时序。
    for streaming in [false, true] {
        for prefix in [vec![], vec![b'h' as u32]] {
            let mut output_specs = Vec::new();
            if !prefix.is_empty() {
                // 先交付一个真实 token，模拟部分输出已送到客户端。
                output_specs.push((prefix.clone(), None));
            }
            // 再以 Error finish 结束，验证它必须变成 RPC 错误而非 finish event。
            output_specs.push((vec![], Some(EngineCoreFinishReason::Error)));
            let (mut client, server_task, engine_task) =
                grpc_test_server(b"engine-grpc-error", output_specs).await;
            let request = pb::GenerateRequest {
                request_id: "test-engine-error".to_string(),
                model: "test-model".to_string(),
                prompt: Some(pb::generate_request::Prompt::Text("hello".to_string())),
                stopping: Some(pb::StoppingCriteria {
                    max_new_tokens: 10,
                    ..Default::default()
                }),
                response: Some(pb::ResponseOptions {
                    output_token_ids: true,
                    ..Default::default()
                }),
                ..Default::default()
            };
            let status = if streaming {
                // streaming: 已交付的 token 必须保留，最后以 INTERNAL 状态码终止。
                let mut stream =
                    client.generate_stream(request).await.expect("start stream").into_inner();
                let mut received_tokens = Vec::new();
                let status = loop {
                    match stream.message().await {
                        Ok(Some(response)) => {
                            if let Some(output) = response.outputs {
                                assert!(
                                    output.finish_info.is_none(),
                                    "error became a finish event"
                                );
                                received_tokens.push(output.token_id);
                            }
                        }
                        Err(e) => {
                            // 这里会捕获到 INTERNAL 错误
                            break e;
                        }
                    }
                };
            } else {
                // unary
                let response = client.generate(request).await;
                let status = response.status();
            };
            // 验证返回的错误码
            assert_eq!(status, StatusCode::INTERNAL_SERVER_ERROR);
        }
    }
}
```

```

        );
        received_tokens.extend(output.token_ids);
    }
}
Ok(None) => panic!("engine error ended as a successful stream"),
Err(status) => break status,
}
};
assert_eq!(received_tokens, prefix);
status
} else {
    // unary: 整个请求直接以错误结束，不应返回成功的响应。
    client.generate(request).await.expect_err("engine error must fail the RPC")
};
assert_eq!(status.code(), tonic::Code::Internal);

engine_task.await.expect("mock engine task");
server_task.abort();
}
}
}

```

评论区精华

Thanks @alec-flowers! —— 维护者 njhill 直接批准，无实质技术驳回。

- claude[bot] 提示：“This pull request is from a fork — automated review is disabled.”
即 fork 来源 PR 未走自动化 review，语义取舍主要由 PR body 与单测承载。

讨论虽少，但 PR body 本身已交代关键设计决策：错误用 **INTERNAL** 暴露、流式先交付后终止、**Abort**（真实客户端取消）保持不变。

风险与影响

- 对外行为变更：对已按“成功 + ABORTED”消费响应的 gRPC 客户端，命中引擎错误时将从成功响应变为 INTERNAL RPC 错误，属破坏性语义修正；实测 HTTP/SSE 层正确输出错误后 [DONE]，但其他自研客户端需要适配。
- 流式终止路径：generate_stream 依赖 tonic 把最后一个 Err 帧作为流终止状态；若 tx.send 因客户端断开失败，break 仍生效，但错误详情可能无法送达（客户端已断），可接受。
- 语义边界：FinishReason::Repetition 仍映射 ABORTED，与 Error 的区分依赖引擎侧语义清晰，建议在 #55399 的引擎拒绝设计中统一。
- 范围可控：仅涉 Rust 前端 3 个文件，未触及 Python 引擎核心与 proto 定义，回归测试覆盖主要路径，风险中等偏低。

关联脉络

本 PR 是 #55399 的配套修复：后者补充“引擎状态拒绝 + HTTP 处理”，本 PR 修正现存 gRPC **Error** → **Aborted** 映射，两者共同把错误语义在 Rust 前端各协议层对齐。结合仓库近

期 Rust 前端活跃度（如 #56260 对齐 DeepSeek 工具调用渲染、#56460 的 httpx 迁移），可以看到 vLLM 的 Rust 前端（vllm-server）正在系统性补齐错误处理与协议兼容细节，gRPC 服务从“基础可用”走向“语义正确”。