PR #50368 完整报告

vllm-project/vllm

[Rust Frontend][gRPC] Add multimodal image inference

合并时间：2026-08-04 17:34

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50368

执行摘要

本 PR 为 vLLM Rust 前端的 gRPC Inference 服务补齐图像多模态推理：proto 契约新增 MediaItem/Modality，支持 HTTP(S) URL、data URI 与 raw bytes 三种传图方式；服务端通过 convert.rs 翻译为 MediaContentPart 后，复用 vllm-chat 既有的多模态预处理管线完成抓取、placeholder 展开与 MmFeatures 生成，并将 unary/stream 两条生成路径统一收口到 prepare_request，同时引入请求级 tracing 日志与 32 MB 消息上限。功能边界清晰（仅图像，音频/视频返回 UNIMPLEMENTED），但 SSRF 校验缺失和 ChatError 未归类是遗留问题。

功能与动机

PR body 明确写："Add image inputs to the Rust frontend gRPC Inference service"、"Reuse the existing multimodal preprocessing pipeline to fetch and process images, expand placeholder tokens, and attach engine-facing multimodal features"，并声明"This PR implements image inference only. Audio and video support will be added in a follow-up PR"。补充动机：此前 gRPC GenerateRequest 只有文本 /token_ids prompt，HTTP 侧的 /v1/chat/completions 已具备多模态能力，gRPC 侧缺口明显；review 中 BugenZhao 还指出 tonic 默认 4 MB 消息上限会拒绝带大图的请求（"bump the default max_decoding_message_size to avoid typical requests being rejected"），最终在合并前直接修掉。

实现拆解

1. proto 契约扩展（rust/proto/inference.proto）：新增 Modality 枚举（IMAGE/VIDEO/AUDIO）与 MediaItem 消息（oneof source：urldata_uri/raw_bytes，外加 mime_type、uuid），并在 GenerateRequest 追加 repeated MediaItem media = 14；VIDEO/AUDIO 枚举提前预留但服务端不实现，为后续 PR 留出契约空间。
2. 转换层（rust/src/server/src/grpc/convert.rs）：新增 media_parts_from_request 与 validate_media_uri。按 item.modality() 访问器分支：Unspecified 报 INVALID_ARGUMENT、非 Image 模态报 UNIMPLEMENTED 并带索引定位、缺 source 报 INVALID_ARGUMENT；URL/data URI 映射为 MediaContentPart::ImageUrl（scheme 白名单 http/https/data），raw bytes 映射为 ImageData 并透传 mime_type 与 uuid。
3. 会话层 API（rust/src/chat/src/lib.rs 与 rust/src/chat/src/multimodal.rs）：ChatRequestProcessor 新增私有 prepare_media，ChatLlm 暴露同名 pub 方法委托实现

，避免在 server 层重复多模态逻辑；MultimodalModelInfo::prepare_multimodal 由私有改为 pub(crate) 可见性；对外导出 MediaContentPart 与 MmFeatures。

4. gRPC 服务层重构（rust/src/server/src/grpc/inference.rs）：新增 PreparedGrpcRequest 与统一入口 prepare_request，unary 与 streaming 共用同一准备路径；自动生成缺失的 request_id、写入 arrival_time、建立 info_span! 请求级日志贯穿 preparation/submission/collection/stream；媒体场景强制 Prompt::TokenIds（placeholder 展开依赖 token_ids），否则返回 INVALID_ARGUMENT；调用 chat.prepare_media 原位展开 placeholder 并挂载 mm_features；新增 log_text_error 统一记录下游错误。
5. 测试与配置配套：tests.rs 新增 FakeMultimodalBackend（基于 qwen2_vl 配置构建 MultimodalModelInfo）与 setup_grpc_service_with_backend（支持注入 backend 与请求断言闭包），新增 3 个测试覆盖媒体展开结果、文本 prompt 拒绝、超过 tonic 默认上限的大请求；server/lib.rs 将 gRPC max_decoding_message_size 与 HTTP JSON body limit 共用 DEFAULT_REQUEST_BODY_LIMIT_BYTES（32 MB），routes.rs 同步复用；cli.rs/config.rs 修正服务名注释。

统一请求准备路径（inference.rs）

关键源码片段

媒体请求转换与 URI 校验（convert.rs）

```
/// Convert gRPC MediaItem 的列表为内部 MediaContentPart 列表。
///
/// 目前只支持 Image 模态；视频与音频在 proto 中已预留枚举，
/// 但服务端实现会以 UNIMPLEMENTED 拒绝，错误信息带索引便于定位。
pub fn media_parts_from_request(
    media: Vec<pb::MediaItem>,
) -> Result<Vec<MediaContentPart>, Status> {
    let mut parts = Vec::with_capacity(media.len());
    for (index, item) in media.into_iter().enumerate() {
        // proto3 缺省值为 0，用 modality() 访问器把未声明的请求挡在门外。
        match item.modality() {
            pb::Modality::Image => {}
            pb::Modality::Unspecified => {
                return Err(Status::invalid_argument(format!(
                    "media[{index}].modality is required"
                )));
            }
            other => {
                return Err(Status::unimplemented(format!(
                    "media[{index}].modality {other:?} is not supported by the gRPC service"
                )));
            }
        }
        // proto3 空字符串与“未设置”无法区分，按业务约定空值视为缺省元数据。
        let uuid = (!item.uuid.is_empty()).then_some(item.uuid);
```

```
        let mime_type = (!item.mime_type.is_empty()).then_some(item.mime_type);
        let part = match item.source {
            Some(pb::media_item::Source::Url(url)) => {
                // gRPC 侧只约束 scheme，host 级校验（防 SSRF）尚未落地。
                validate_media_uri(index, "url", &url, &["http", "https"])?;
                MediaContentPart::ImageUrl {
                    url,
                    detail: None,
                    uuid,
                }
            }
            Some(pb::media_item::Source::DataUri(uri)) => {
                validate_media_uri(index, "data_uri", &uri, &["data"])?;
                MediaContentPart::ImageUrl {
                    url: uri,
                    detail: None,
                    uuid,
                }
            }
            Some(pb::media_item::Source::RawBytes(bytes)) => MediaContentPart::ImageData {
                data: bytes,
                mime_type,
                uuid,
                detail: None,
            },
            None => {
                return Err(Status::invalid_argument(format!(
                    "media[{index}].source is required"
                )));
            }
        };
        parts.push(part);
    }
    Ok(parts)
}

/// 仅校验 URI 是否可解析且 scheme 在白名单内。
///
/// 注意：没有解析 host 或阻断内网地址，存在 SSRF 面，
/// 见 review 中 depthfirst-app[bot] 的讨论。
fn validate_media_uri(
    index: usize,
    field: &str,
    value: &str,
    allowed_schemes: &[&str],
) -> Result<(), Status> {
    let uri = Url::parse(value).map_err(|_| {
        Status::invalid_argument(format!("media[{index}].{field} is not a valid URI"))
    })?;
```

```
        if !allowed_schemes.contains(&uri.scheme()) {
            return Err(Status::invalid_argument(format!(
                "media[{index}].{field} must use the {} scheme",
                allowed_schemes.join(" or ")
            )));
        }
        Ok(())
    }
```

评论区精华

- 委托位置：BugenZhao 建议“Shall we make it a method on self.processor and perhaps delegate it here?”，作者答复“Fixed.”，最终 ChatLlm::prepare_media 只做委托，实现落在 ChatRequestProcessor。
- 代码风格：BugenZhao 指出“We can directly call item.modality()”，以及日志中传 status.message() 拷贝“doesn't seem very necessary”，作者均接受并修复。
- 错误归类：BugenZhao 询问“Shall we also categorize the ChatError returned here?”，作者说明“This was intentionally deferred to a follow up PR to reduce the scope of this PR”，维护者接受并合并。
- 消息上限：BugenZhao 在 proto 行内提问是否提升 max_decoding_message_size 以避免典型请求被拒，作者想做可配置 flag；BugenZhao 最终直接 push 一个 commit：“just pushed a commit to use that hardcoded value for gRPC server as well”，复用 HTTP 侧 32 MB 常量。
- 安全告警：depthfirst-app[bot] 标记 MEDIUM 级 SSRF——validate_media_uri 只校验 scheme 不校验 host，MediaConnector 的 reqwest 客户端默认跟随重定向，可被诱导访问 169.254.169.254 云 metadata 或内网服务；未见维护者公开回应，属于未解决风险。

风险与影响

- SSRF（高）：convert.rs 仅做 scheme 白名单，聊天层的 MediaConnector 抓取 URL 时无 host/IP 限制且默认跟随重定向；若 gRPC 服务具备内网访问能力，恶意客户端可探测内网与云 metadata，错误信息可能泄露内部拓扑。建议后续补 host 校验与禁重定向。
- 核心路径重构（中）：prepare_request 统一了 unary/stream，行为变化点包括自动生成 request_id、arrival_time 赋值和日志结构，回归面覆盖所有既有 gRPC 文本请求；现有测试主要覆盖新功能，TLS、超时等旧边界未逐项回归。
- 行为兼容性：媒体请求必须提供 token_ids prompt，文本 prompt + media 会直接返回 INVALID_ARGUMENT，客户端需适配；proto 字段为纯增量，旧客户端不受影响。
- 资源开销：gRPC 消息上限由 tonic 默认 4 MB 提至 32 MB，raw bytes 大图会驻留内存；URL 抓取引入出网依赖，重定向放大可能被滥用。
- 团队影响：为 audio/video、媒体错误分类与可配置消息上限三个 follow-up 打好了契约与服务层基础，Modality 枚举已预留。

关联脉络

本 PR 是 Rust 前端 gRPC 服务能力补全的一环，与近期 Rust 侧 PR 存在明确演进关系：#50746 加固 gRPC stop 字符串校验（同样改动 grpc/tests.rs），#48048 在 GenerateRequest 中引入 session_id 字段（与本 PR 的 media = 14 同属请求契约演进），#50868 则演进 Rust benchmark 流处理。整体方向是让 Rust 前端与 Python 侧的多模态、会话能力逐步对齐。