PR #48554 完整报告

vllm-project/vllm

[Rust Frontend] Integrate MM audio support

合并时间：2026-07-15 15:00

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48554

执行摘要

PR #48554 在 Rust 前端（vllm-rs）中集成了音频多模态支持，使用户能够通过 Rust API 处理音频输入（如 Qwen3-ASR）。变更涉及 Rust 多模态核心、渲染器及 Python 侧模型适配，核心是将 ResolvedMultimodalSpec 重构为 per-modality，并提取通用 item 构建逻辑。

功能与动机

音频支持已在 llm-multimodal 库中实现（PR#1905），此 PR 将其引入 vLLM 的 Rust 前端，填补了音频模态的空白，使 Rust 前端具备与 Python 前端等同的多模态能力（图像、视频、音频）。端到端测试通过 Qwen3-ASR 验证。

实现拆解

1. 新增音频预处理模块（audio.rs）：封装 AudioPreProcessor 调用，通过 tokio::spawn_blocking 执行 CPU 密集的音频特征提取。
2. 重构多模态数据结构（multimodal.rs）：将 ResolvedMultimodalSpec 从共享改为每个模态独立持有，包含 modality、field_layouts（模态特定）和 primary_key；新增 audio 字段和 AudioModalitySupport。
3. 提取通用 item 构建（item.rs）：将图像和音频共用的 build_batched_items 独立成模块，减少 image.rs 和 audio.rs 中的重复代码。
4. 扩展模板渲染（renderer/hf/mod.rs）：增加 audio_token 字段和 TemplateContentPart::Audio 变体，使 Jinja 模板能渲染音频占位符。
5. 适配请求模型（request.rs / convert.rs）：支持 InputAudio 和 AudioUrl 内容部分，在 server 路由中转换。
6. Python 侧适配（qwen3_asr.py, qwen2_5_omni_thinker.py）：接收 flat 化的 3D 音频特征，适配批处理。

rust/src/chat/src/multimodal/audio.rs

新增文件，实现音频预处理核心逻辑，包括 prepare_audios 和 preprocess_audios 方法

```
// SPDX-License-Identifier: Apache-2.0
// SPDX-FileCopyrightText: Copyright contributors to the vLLM project

//! Audio-modality preparation through `llm-multimodal`.

use std::sync::Arc;
```

```
use llm_multimodal::{AudioClip, Modality, PreprocessedEncoderInputs};
use vllm_engine_core_client::protocol::dtype::ModelDtype;
use super::{AudioModalitySupport, MultimodalModelInfo, PreparedMedia, item};
use crate::error::{Error, Result, bail_multimodal, multimodal};

/// Forward-kwargs name of the primary audio encoder input.
pub(super) const AUDIO_PRIMARY_KEY: &str = "input_audio_features";

impl MultimodalModelInfo {
    /// Preprocess fetched audio clips as one batch and build per-item features.
    pub(super) async fn prepare_audios(
        &self,
        clips: Vec<Arc<AudioClip>>,
        uuids: Vec<Option<String>>,
    ) -> Result<PreparedMedia> {
        let support = self.audio.as_ref().ok_or_else(|| Error::UnsupportedModality {
            modality: Modality::Audio.to_string(),
        })?;
        // 调用 preprocess_audios 在 blocking 线程中进行 CPU 密集预处理
        let preprocessed = self.preprocess_audios(support, &clips).await?;
        // 获取 prompt 替换（占位符展开）
        let replacements = support.spec.prompt_replacements_for(&self.context, &preprocessed)?;
        if replacements.len() != clips.len() {
            bail_multimodal!(
                "number of audio prompt replacements {} does not match number of audio clips {}",
                replacements.len(),
                clips.len()
            );
        }
        let hashes = clips.iter().map(|clip| clip.hash.clone()).collect();
        // 通过通用 item::build_batched_items 构建引擎侧多模态特征
        let items = item::build_batched_items(
            &support.spec,
            preprocessed,
            hashes,
            uuids,
            ModelDtype::Float32,
        )?;
        Ok(PreparedMedia {
            modality: Modality::Audio,
            placeholder: support.placeholder.clone(),
            replacements,
            items,
        })
    }

    /// Run CPU-heavy audio preprocessing in a blocking task.
    async fn preprocess_audios(
        &self,
```

```
        support: &AudioModalitySupport,
        clips: &[Arc<AudioClip>],
    ) -> Result<PreprocessedEncoderInputs> {
        let processor = Arc::clone(&support.processor);
        let clips = clips.to_vec();
        // 使用 spawn_blocking 避免阻塞异步运行时
        tokio::task::spawn_blocking(move || Ok(processor.preprocess(&clips)?))
            .await
            .map_err(|error| multimodal!("audio preprocessing task failed: {error}"))?
    }
}
```

rust/src/chat/src/multimodal.rs

核心重构文件：将 ResolvedMultimodalSpec 改为 per-modality，新增 audio 字段和 resolve_audio_processor 方法

```
// （省略头部注释和导入）

/// Resolved multimodal support for one loaded model.
#[derive(Clone)]
pub struct MultimodalModelInfo {
    context: MultimodalModelContext,
    image: Option<ModalitySupport>,
    video: Option<ModalitySupport>,
    // 新增 audio 字段，单独持有可选音频支持
    audio: Option<AudioModalitySupport>,
    media_connector: Arc<MediaConnector>,
}

impl MultimodalModelContext {
    /// Resolve an audio preprocessor for one loaded model.
    fn resolve_audio_processor(
        &self,
        model_spec: &'static dyn ModelProcessorSpec,
        preprocessor_config: &PreProcessorConfig,
    ) -> Option<Arc<dyn AudioPreProcessor>> {
        // 委托给模型 spec 的 audio_processor 方法
        model_spec.audio_processor(&self.config, preprocessor_config).map(Arc::from)
    }
}

/// Static model-specific tensor-layout behavior for one modality.
#[derive(Clone)]
struct ResolvedMultimodalSpec {
    raw: &'static dyn ModelProcessorSpec,
    modality: Modality, // 新增：指示该 spec 对应的模态
    field_layouts: EncoderFieldLayouts, // 模态特定的编码器字段布局
    keep_on_cpu_keys: HashSet<String>,
}
```

```
impl ResolvedMultimodalSpec {
    fn new(raw: &'static dyn ModelProcessorSpec, modality: Modality) -> Self {
        Self {
            raw,
            modality,
            field_layouts: raw.encoder_field_layouts_for(modality),
            keep_on_cpu_keys: raw.keep_on_cpu_keys_for(modality).into_iter().collect(),
        }
    }

    fn primary_key(&self) -> &'static str {
        match self.modality {
            Modality::Image => image::IMAGE_PRIMARY_KEY,
            Modality::Video => video::VIDEO_PRIMARY_KEY,
            Modality::Audio => audio::AUDIO_PRIMARY_KEY,
            Modality::ImageEmbeds => unreachable!("image embeds use no encoder preprocessor")
            ,
        }
    }

    fn field_layout_for(&self, key: &str) -> Option<&FieldLayout> {
        if key == self.primary_key() {
            Some(&self.field_layouts.encoder_input)
        } else {
            self.field_layouts.metadata.get(key)
        }
    }
}
```

评论区精华

> chatgpt-codex-connector[bot]: 当音频请求时长不同时，input_audio_features 被标记为 batched field 可能导致 Python 端处理失败，建议使用 flat_from_sizes。 BugenZhao: 已在 commit b738294 中修复，将 Qwen3-ASR 音频特征改为 flat field。

风险与影响

- Rust 端重构风险：ResolvedMultimodalSpec 从共享改为 per-modality，可能影响图像 / 视频路径的字段布局和 primary key 解析。但测试验证了基本路径。
- 新增依赖风险：依赖 llm-multimodal 的 AudioPreProcessor，其行为变化可能影响音频预处理链路。
- Python 模型适配风险：qwen3_asr.py 和 qwen2_5_omni_thinker.py 的修改涉及 forward 逻辑，可能影响现有 Python 用户。
- 兼容性：无 breaking change，旧有图像 / 视频功能不受影响。

关联脉络

此 PR 是 Rust 前端多模态能力扩展的第二阶段（第一阶段为 PR#47959 图像 / 视频支持），后续可基于当前架构支持更多模态（如 TOA、IMU）。与 llm-multimodal 库的演进紧密耦合。