PR #47729 完整报告

vllm-project/vllm

[Model] Support MOSS-Transcribe-Diarize

合并时间：2026-07-08 19:05

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47729

执行摘要

- 一句话：支持 MOSS-Transcribe-Diarize 语音转写模型
- 推荐动作：值得精读。该 PR 展示了 vLLM 多模态模型中音频路线（转录）的完整集成模式，包括 TensorSchema 定义、MultiModalProcessor 实现、WeightsMapper 使用。review 中关于多音频并发正确性的发现具有普遍参考价值，建议关注类似模型中的 batch 处理模式。

功能与动机

MOSS-Transcribe-Diarize 是一个端到端语音转文本模型，支持长格式转写并生成带时间戳的说话人标签。PR body 指出："MOSS-Transcribe-Diarize is an end-to-end speech-to-text model for long-form transcription with timestamped speaker labels." 该模型在 vLLM 中注册后，用户可使用 OpenAI 兼容的 ASR 端点。

实现拆解

1. 配置类：在 vllm/transformers_utils/configs/moss_transcribe_diarize.py 中新增 MossTranscribeDiarizeConfig，继承 PretrainedConfig，整合 Qwen3Config 和 WhisperConfig，默认参数匹配 Whisper-medium + Qwen3-0.6B，并暴露 audio_merge_size、adaptor_input_dim 等超参。
2. 模型实现：在 vllm/model_executor/models/moss_transcribe_diarize.py 中定义 MossTranscribeDiarizeForConditionalGeneration，组合 WhisperEncoder、时域自适应器（time-merge）和 Qwen3 语言模型。实现 SupportsTranscription 接口，定义输入输出 TensorSchema（MossTranscribeDiarizeAudioInputs/MossTranscribeDiarizeEmbeddingInputs），并构建基于 ChatML 的提示模板。
3. 模型注册：在 vllm/model_executor/models/registry.py 中添加 MossTranscribeDiarizeForConditionalGeneration 的映射，使模型名称可被 vLLM 识别。
4. 配置映射：依次修改 vllm/transformers_utils/configs/__init__.py 和 vllm/transformers_utils/config.py，将 MossTranscribeDiarizeConfig 加入配置查找表，支持从 HuggingFace ID 自动加载。
5. 测试与文档：在 tests/models/registry.py 中添加模型条目（标记为不可在线获取），并在 docs/models/supported_models.md 中列出新模型。

6. 审查修复：根据 review 反馈删除了重复代码、使用 WeightsMapper 加载权重、简化提示构造、将硬编码合并因子改为从配置读取、修正 _process_audio_input 使其支持多音频批处理。

关键文件：

- vllm/model_executor/models/moss_transcribe_diarize.py（模块 核心模型；类别 source；类型 core-logic；符号 MossTranscribeDiarizeAudioInputs, MossTranscribeDiarizeEmbeddingInputs, _compute_total_audio_tokens, _get_max_audio_samples）：新增主模型实现，包含模型类、输入输出 TensorSchema、多模态处理逻辑、权重加载等全部核心代码。
- vllm/transformers_utils/configs/moss_transcribe_diarize.py（模块 模型配置；类别 source；类型 core-logic；符号 MossTranscribeDiarizeConfig, init）：新增配置类，继承 PretrainedConfig，管理 Qwen3 和 Whisper 的子配置，暴露关键超参。
- vllm/model_executor/models/registry.py（模块 模型注册；类别 source；类型 data-contract）：将新模型注册到模型字典，使 vLLM 能够识别和加载。
- vllm/transformers_utils/configs/__init__.py（模块 配置映射；类别 source；类型 configuration）：将 MossTranscribeDiarizeConfig 添加到配置映射字典，支持自动配置加载。
- vllm/transformers_utils/config.py（模块 配置工具；类别 source；类型 configuration）：添加 moss_transcribe_diarize 到配置查找表，支持从 HuggingFace ID 自动解析。
- tests/models/registry.py（模块 测试覆盖；类别 test；类型 test-coverage）：添加模型测试条目，验证模型注册逻辑。
- docs/models/supported_models.md（模块 文档；类别 docs；类型 documentation）：在支持模型文档中添加 MOSS-Transcribe-Diarize。

关键符号：MossTranscribeDiarizeConfig, MossTranscribeDiarizeForConditionalGeneration, MossTranscribeDiarizeAudioInputs, MossTranscribeDiarizeEmbeddingInputs, _compute_total_audio_tokens, _get_max_audio_samples, _as_audio_embedding_list, _get_required_token_id, _get_audios_from_mm_data, _add_vllm_audio_metadata, load_weights, forward, _process_audio_input

关键源码片段

vllm/transformers_utils/configs/moss_transcribe_diarize.py

新增配置类，继承 PretrainedConfig，管理 Qwen3 和 Whisper 的子配置，暴露关键超参。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
from typing import Any
from transformers import PretrainedConfig, Qwen3Config
from transformers.models.whisper.configuration_whisper import WhisperConfig

class MossTranscribeDiarizeConfig(PretrainedConfig):
    """Configuration for MOSS-Transcribe-Diarize."""
    model_type = "moss_transcribe_diarize"
```

```
# 子配置声明，使 transformers 能自动加载
sub_configs = {"text_config": Qwen3Config, "audio_config": WhisperConfig}
keys_to_ignore_at_inference = ["past_key_values"]

def __init__(
    self,
    text_config: dict[str, Any] | Qwen3Config | None = None,
    audio_config: dict[str, Any] | WhisperConfig | None = None,
    audio_token_id: int = 151671, # <|audio_start|> token id
    audio_merge_size: int = 4, # 时域合并倍数
    adaptor_input_dim: int | None = None, # 自适应器输入维度，默认由 audio_config.d_model *
    merge_size 计算
    tie_word_embeddings: bool = True,
    **kwargs: Any,
) -> None:
    # text_config：默认使用 Qwen3-0.6B 配置
    text_config_obj: Qwen3Config
    if text_config is None:
        text_config_obj = Qwen3Config(
            vocab_size=151936, hidden_size=1024, intermediate_size=3072,
            num_hidden_layers=28, num_attention_heads=16, num_key_value_heads=8,
            head_dim=128, max_position_embeddings=40960,
            tie_word_embeddings=tie_word_embeddings, rope_theta=1_000_000.0,
            layer_types=["full_attention"] * 28,
        )
    elif isinstance(text_config, dict):
        text_config_obj = Qwen3Config(**text_config)
    else:
        text_config_obj = text_config

    # audio_config：默认使用 Whisper-medium 配置
    audio_config_obj: WhisperConfig
    if audio_config is None:
        audio_config_obj = WhisperConfig(
            num_mel_bins=80, d_model=1024, encoder_layers=24,
            encoder_attention_heads=16, encoder_ffn_dim=4096,
            max_source_positions=1500, dropout=0.0, attention_dropout=0.0,
            activation_dropout=0.0, activation_function="gelu",
            encoder_layerdrop=0.0, scale_embedding=False,
        )
    elif isinstance(audio_config, dict):
        audio_config_obj = WhisperConfig(**audio_config)
    else:
        audio_config_obj = audio_config

    text_config_obj.tie_word_embeddings = tie_word_embeddings
    if not getattr(text_config_obj, "layer_types", None):
        text_config_obj.layer_types = ["full_attention"] * text_config_obj.num_hidden_layers
```

```
        super().__init__(tie_word_embeddings=tie_word_embeddings, **kwargs)

        self.text_config = text_config_obj
        self.audio_config = audio_config_obj
        self.audio_token_id = int(audio_token_id)
        self.audio_merge_size = int(audio_merge_size)
        self.adaptor_input_dim = (
            int(adaptor_input_dim) if adaptor_input_dim is not None
            else int(audio_config_obj.d_model) * int(audio_merge_size)
        )
        # 暴露常用属性，方便模型代码直接引用
        self.vocab_size = int(text_config_obj.vocab_size)
        self.hidden_size = int(text_config_obj.hidden_size)
        self.num_hidden_layers = int(text_config_obj.num_hidden_layers)
        self.num_attention_heads = int(text_config_obj.num_attention_heads)
        self.num_key_value_heads = int(text_config_obj.num_key_value_heads)
        self.head_dim = int(text_config_obj.head_dim)
        self.hidden_act = text_config_obj.hidden_act
        self.max_position_embeddings = int(text_config_obj.max_position_embeddings)
        self.rms_norm_eps = float(text_config_obj.rms_norm_eps)
        self.is_causal = True
```

评论区精华

1. 多音频并发正确性（linyueqian，高优先级）：_process_audio_input 始终返回单元素列表，但 v1 engine 可能将多个音频请求分在一批，触发断言失败。作者添加循环处理并修复。
2. 硬编码合并因子（linyueqian，中优先级）：编码器直接乘 4，而配置中有 audio_merge_size，若不一致则形状错误。作者改为从配置读取。
3. 重复 / 冗余代码（Isotr0py）：提示构建和 Qwen3Config 创建重复。作者简化并移除。
4. 权重映射（Isotr0py）：建议使用 WeightsMapper 兼容旧格式。作者采用。
5. Prompt 注入风险（bot，低优先级）：用户输入未转义拼接至 ChatML，可能注入控制标记；未修复。

- 多音频并发正确性 (correctness): 作者修复为返回与输入数量匹配的嵌入列表。
- 编码器合并因子硬编码 (design): 作者改为从配置读取 merge factor。
- 重复代码与冗余配置 (style): 作者移除重复，简化提示构造。
- 权重映射使用 WeightsMapper (design): 作者采纳建议，增加 WeightsMapper。
- Prompt 注入安全风险 (security): 未回复，风险未修复。建议后期参考 qwen3_asr.py 的 sanitize_chatml_input 函数处理。

风险与影响

- 风险：
  1. 多音频并发正确性（高）：原实现假设单音频请求，在并发场景会导致崩溃。虽已修复，但类似批量处理模式在其他模型可能潜伏，需审查相似代码。

2. 硬编码值（中）：合并因子硬编码曾导致配置不一致，已修复。但编码器内部还有其他硬编码值（如 WHISPER_ENCODER_STRIDE）未配置化。
3. Prompt 注入（低）：用户输入直接嵌入模板，未转义 ChatML 标记，存在理论安全风险。参考 qwen3_asr.py 的 sanitize_chatml_input 可缓解。
4. 新路径覆盖：语音转写流程与现有视觉多模态共享架构，但测试只覆盖注册，缺乏端到端推理测试，增加回归风险。 - 影响：用户：新增 MOSS-Transcribe-Diarize 模型支持，可通过 OpenAI 兼容 API 调用 ASR 带说话人分离功能。系统：无性能或资源影响，仅新模型文件，不影响现有模型。团队：维护负担轻微，新增约 900 行代码，配置映射已标准化。

- 风险标记：多音频并发正确性，硬编码合并因子，Prompt 注入

关联脉络

- 暂无明显关联 PR