# PR #48543 完整报告

vllm-project/vllm

[Frontend] Add diarized_json support for MOSS-Transcribe-Diarize

合并时间：2026-07-30 07:24

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48543

## 执行摘要

- 一句话：为 MOSS-Transcribe-Diarize 添加 diarized_json 响应格式
- 推荐动作：该 PR 代码质量高，测试覆盖全面，性能测量严谨，设计上遵循了 OpenAI 兼容性和扩展性原则。值得精读其解析器设计和测试策略，对实现类似模型特定输出解析有参考价值。

## 功能与动机

MOSS-Transcribe-Diarize 模型已通过 Whisper 风格 verbose_json 提供说话人分离转录，但缺乏 OpenAI 专用的 diarized_json 格式，限制了与 OpenAI SDK 的互操作性。用户通过 issue #48443 请求支持 diarized_json 以实现与 OpenAI API 的直接兼容，避免自定义响应转换。

## 实现拆解

1. 定义数据契约和协议模型：在 interfaces.py 中新增 DiarizedTranscriptionSegment 数据类和 SupportsTranscription.supports_diarized_transcription 标志，在 protocol.py 中定义 TranscriptionDiarizedSegment 和 TranscriptionResponseDiarizedPydantic 模型。
2. 实现 MOSS 特定解析器：在 moss_transcribe_diarize.py 中添加类方法 parse_diarized_transcript，使用正则表达式线性解析 MOSS 输出，验证每个段的开始、说话人、结束时间戳，并返回 DiarizedTranscriptionSegment 列表。将时间戳和说话人校验通用函数 parse_diarized_timestamp 和 parse_diarized_speaker 提取到 utils.py 以便复用。
3. 集成到服务层：在 serving.py 的 _create_speech_to_text 中处理 response_format=diarized_json，调用模型类解析器并序列化为 TranscriptionResponseDiarized；添加校验阻止不支持的模型或流式请求。
4. 扩展响应格式枚举：将 AudioResponseFormat 替换为 TranscriptionResponseFormat 并添加 'diarized_json'。
5. 文档更新：在 sphinx 文档中添加 diarized_json 示例和 MOSS 使用说明。
6. 测试覆盖：新增 108 行单元测试，覆盖正常段、重叠、噪声、不规则时间戳、空段和格式错误场景，确保 fail-closed 行为。

关键文件：

- tests/models/multimodal/processing/test_moss_transcribe_diarize.py（模块 MOSS 解析测试；类别 test；类型 test-coverage；符号 test_parse_diarized_transcript_preserves_moss_segments, test_parse_diarized_transcript_preserves_overlapping_segments, test_parse_diarized_transcript_preserves_numeric_text_markers, test_parse_diarized_transcript_ignores_whitespace_between_segments）：新增 108 行单元测试，全面覆盖 parse_diarized_transcript 的各种边界情况，验证 fail-closed 行为。
- vllm/model_executor/models/moss_transcribe_diarize.py（模块 模型解析；类别 source；类型 data-contract；符号 parse_diarized_transcript, _MOSS_DIARIZED_HEADER_RE, _MOSS_DIARIZED_END_RE）：核心变更：添加 MOSS 特定解析器 parse_diarized_transcript，定义正则表达式和类方法。
- vllm/model_executor/models/interfaces.py（模块 模型接口；类别 source；类型 data-contract；符号 DiarizedTranscriptionSegment, SupportsTranscription.supports_diarized_transcription, SupportsTranscription.parse_diarized_transcript）：定义 DiarizedTranscriptionSegment 数据类和在 SupportsTranscription 协议中添加 supports_diarized_transcription 标志及 parse_diarized_transcript 抽象方法。
- vllm/entrypoints/speech_to_text/transcription/protocol.py（模块 前端协议；类别 source；类型 core-logic；符号 TranscriptionDiarizedSegment, TranscriptionResponseDiarized, TranscriptionResponseVariant）：定义响应模型 TranscriptionDiarizedSegment 和 TranscriptionResponseDiarized，更新响应类型联合。
- vllm/entrypoints/speech_to_text/base/serving.py（模块 前端服务；类别 source；类型 core-logic；符号 _create_speech_to_text）：实现服务端处理 diarized_json 格式的完整分支，包括校验模型支持、调用解析器、错误处理。
- vllm/model_executor/models/utils.py（模块 工具函数；类别 source；类型 data-contract；符号 parse_diarized_timestamp, parse_diarized_speaker）：提取通用时间戳和说话人校验函数，供解析器复用。
- docs/serving/online_serving/speech_to_text.md（模块 文档；类别 docs；类型 documentation）：更新文档展示新 diarized_json 响应格式及使用示例。

关键符号：MossTranscribeDiarizeForConditionalGeneration.parse_diarized_transcript, parse_diarized_timestamp, parse_diarized_speaker, SupportsTranscription.parse_diarized_transcript, _create_speech_to_text

## 关键源码片段

### vllm/model_executor/models/moss_transcribe_diarize.py

核心变更：添加 MOSS 特定解析器 parse_diarized_transcript，定义正则表达式和类方法。

```
import regex as re

# 匹配段头：[start][Sxx]
_MOSS_DIARIZED_HEADER_RE = re.compile(
    r"\[(?P<start>[0-9.]{1,32})\]\s*\[(?P<speaker>S[0-9]{1,15})\]"
)
```

```
# 匹配段尾：[end] 必须位于末尾
_MOSS_DIARIZED_END_RE = re.compile(r"\[(?P<end>[0-9.]{1,32})\]\s*\Z")

@classmethod
def parse_diarized_transcript(cls, text: str) -> list[DiarizedTranscriptionSegment]:
    """
    解析 MOSS 原生格式 [start][Sxx]text[end] 的转录文本。
    策略：线性查找所有段头，每个段头后必须紧跟一个有效的段尾时间戳，
    否则整个解析失败（fail-closed）。
    """
    # 收集所有合法的段头
    headers: list[tuple[re.Match[str], float, str]] = []
    for match in _MOSS_DIARIZED_HEADER_RE.finditer(text):
        start = parse_diarized_timestamp(match["start"])
        speaker = parse_diarized_speaker(match["speaker"])
        if start is not None and speaker is not None:
            headers.append((match, start, speaker))

    if not headers:
        return [] # 没有段头返回空列表

    segments: list[DiarizedTranscriptionSegment] = []
    for index, (header, start, speaker) in enumerate(headers):
        # 段体范围：当前段头结束到下一个段头开始（或文本末尾）
        next_header_start = (
            headers[index + 1][0].start() if index + 1 < len(headers) else len(text)
        )
        body = text[header.end() : next_header_start]

        # 段体必须以格式 [end] 结尾
        end_match = _MOSS_DIARIZED_END_RE.search(body)
        if end_match is None:
            return [] # 缺少结束时间戳，解析失败

        end = parse_diarized_timestamp(end_match["end"])
        if end is None or end < start:
            return [] # 时间戳无效或顺序错误

        segment_text = body[: end_match.start()].strip()
        if segment_text:
            segments.append(
                DiarizedTranscriptionSegment(
                    start=start,
                    end=end,
                    speaker=speaker,
                    text=segment_text,
                )
            )
```

```
    return segments
```

### vllm/entrypoints/speech_to_text/base/serving.py

实现服务端处理 diarized_json 格式的完整分支，包括校验模型支持、调用解析器、错误处理。

```
if request.response_format == "diarized_json":
    # 调用模型类解析器，返回 DiarizedTranscriptionSegment 列表
    diarized_segments = self.model_cls.parse_diarized_transcript(text)
    if not diarized_segments:
        return self.create_error_response(
            "Model output did not contain a valid diarized transcript"
        )
    # 构造 OpenAI 兼容的响应
    final_response = cast(
        T,
        TranscriptionResponseDiarized(
            duration=duration_s,
            text=separator.join(
                segment.text for segment in diarized_segments
            ),
            segments=[
                {
                    "id": f"seg_{index}",
                    "start": segment.start,
                    "end": segment.end,
                    "text": segment.text,
                    "speaker": segment.speaker,
                }
                for index, segment in enumerate(diarized_segments)
            ],
            usage=usage,
        ),
    )
```

## 评论区精华

- 复用工具函数：Reviewer gcanlin 建议将时间戳和说话人解析函数从模型文件移到 utils.py 以促进复用，提交者采纳并实现了 parse_diarized_timestamp 和 parse_diarized_speaker。
- 解析器简化：gcanlin 提供了一种更简洁的正则表达式实现以减少嵌套深度，提交者接受并应用。
- 审批：Isotr0py 审核后批准该 PR。
  - 将解析函数移到 utils.py 以便复用 (design): 提交者 wskr00 接受建议并完成移动，创建了 parse_diarized_timestamp 和 parse_diarized_speaker。
  - 解析器实现简化 (design): 提交者 wskr00 采用建议，修改了解析器代码。
  - PR 审批 (other): PR 获得批准，等待合并。

## 风险与影响

- 风险：该 PR 风险较低，因为：所有解析操作都在现有模型输出之上进行，不影响核心推理路径；解析器采用 fail-closed 策略，无效输入返回空列表并错误响应，不会崩溃；新 diarized_json 路径不与现有 json/verbose_json 路径交织；提供了 10 个单元测试和一个端到端测试，覆盖边界情况；性能测试表明解析开销极小（~0.094ms）。唯一潜在风险是解析器取决于 MOSS 输出格式的稳定性，若未来模型输出格式变化，解析可能失败但仍然会优雅地报告错误。
- 影响：对用户：使用 MOSS-Transcribe-Diarize 模型的用户现在可以通过 OpenAI SDK 直接请求 diarized_json 响应，无需额外处理；对系统：仅新增代码路径，无性能影响；对团队：定义了一个可扩展的接口（supports_diarized_transcription 和 parse_diarized_transcript），未来其他模型可以复用。影响程度中等，仅限于 speech-to-text 服务的一小部分。
- 风险标记：fail-closed 解析，向后兼容，测试覆盖充分

# 关联脉络

- 暂无明显关联 PR