PR #33604 完整报告

sgl-project/sglang

Fix Whisper transcription for audio over 30 seconds

合并时间：2026-08-15 23:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33604

执行摘要

- 一句话：修复 Whisper 长音频静默截断，新增能量感知分块转写
- 推荐动作：值得精读。该 PR 是一个完整的 " 适配器扩展点 + 服务层通用编排 " 范例，核心看点：①能量感知切分与 vLLM 逐位对齐的兼容策略（切分点可复现）；②strip=False 保留模型边界空白以正确处理无空格脚本的拼接语义；③_build_chunk_request 每块重建 sampling_params 以适配多模态处理器 " 消费式 " 取键的约定；④reviewer 对失败路径、并发上限、语言选择的三轮纠偏展示了生产级审查应当关注的边界。对要扩展新 ASR 适配器的工程师，这是必读示例。

功能与动机

PR body 明确说明：Whisper 的 feature extractor 会把输入填充或截断到固定的 3000 个 mel 帧（30 秒）编码窗口，因此当服务层把长文件作为一次生成发送时，/v1/audio/transcriptions 会静默丢弃 30 秒之后的所有内容。切分点选择对齐旧 vLLM speech-to-text 行为（每个块最后 1 秒内最安静的 100 ms RMS 窗口），并保证 Qwen3-ASR 等其他适配器不受影响。

实现拆解

本 PR 按以下 5 步完成长音频分块转写能力：

1. 新增能量感知分块模块 audio_chunking.py：新增 find_split_point 与 split_audio_energy_aware 两个函数。前者在指定区间内按 100 ms（16 kHz 下 1600 个采样）步长计算 RMS 能量，返回最安静窗口的起始下标；后者先用 load_audio 解码并重采样到 16 kHz 单声道，再按 max_clip_s 步长行走，在每个步长末尾 1 秒搜索窗内寻找低能量切点，返回连续不重叠的 WAV 字节块及各块起始偏移。循环边界刻意与 vLLM 的 _find_split_point 逐位一致，保证切分点与旧 vLLM 转写端点完全一致。
2. 在 serving_transcription.py 的 create_transcription 入口接入分块：_get_audio_duration 增加 soundfile 解析失败时的全量解码 fallback，并通过 asyncio.to_thread 把时长计算与分块解码放入线程池，避免长文件阻塞事件循环。当 adapter.max_audio_clip_s 存在且音频超窗时调用 split_audio_energy_aware；分块失败或结果非法（空块、单块、偏移数不匹配）直接返回 400 错误，不再回退到明知会截断的原始音频。分块结果通过 request._audio_chunks / request._chunk_offsets_s 挂载。

3. 分块请求编排与拼接：_handle_non_streaming_request 检测到 _audio_chunks 后转入 _handle_chunked_non_streaming_request，顺序派发各块（测试断言 max_active_dispatches == 1，回应 reviewer 对并发扇出的担忧）。_build_chunk_request 为每块克隆 GenerateReqInput 并重建独立的 sampling_params dict——多模态处理器在构造 decoder prompt 时会 pop 掉 transcription 级键，复用同一 dict 会导致后续块参数被消费。_finalize_text 抽取原内联的 fused 解析逻辑并新增 strip 参数：strip=False 保留模型发出的边界空白，无空格脚本（zh/ja/th）不注入 ASCII 空格；语言仅在 visible.strip() 非空且未设置时记录（首个非空块胜出）。_abort_chunk_requests 按 rid 中止已派发请求，先处理已分配 rid，再在 dispatch 窗口后二次中止未分配 rid 的请求。
4. 适配器抽象扩展：transcription_adapters/base.py 新增默认 max_audio_clip_s（返回 None，禁用分块）与 build_verbose_response_chunked 默认实现（返回拼接文本、空段列表，适配器可覆写）；whisper.py 声明 max_audio_clip_s = 30.0，实现 build_verbose_response_chunked：直接使用 serving 层已拼接的文本（避免二次 ASCII join 破坏无空格脚本），仅从每块 output_ids 解析段，_parse_segments 新增 time_offset_s 与 seg_id_start 参数，对每块时间戳偏移、段 id 跨块连续编号。common.py 的 load_audio 入参类型从 str 扩展为 Union[str, bytes]。
5. 测试配套：新增 test_audio_chunking.py（安静窗口选择、均匀能量回退、连续性、不重叠、波形保持、无静音仍推进）；test_serving_transcription.py 新增 _MockChunkTokenizerManager（模拟 rid 分配、abort 记录、并发计数）与 TestLongAudioChunkedNonStreaming（顺序拼接、块失败中止、分割失败返回 400、短音频不分块、fused 自动检测首块语言胜出、无空格脚本拼接）；test_whisper_adapter.py 验证段偏移与单块零偏移等价的语义；GPU 端到端 test_serving_transcription.py 用 30 秒静音 + 10 秒语音的 40 秒 WAV 覆盖 JSON 转录、verbose_json 段时间戳超过 30 秒、流式三条路径。

关键文件：

- python/sglang/srt/entrypoints/openai/serving_transcription.py（模块 转录服务；类别 source；类型 core-logic；符号 create_transcription, _handle_chunked_non_streaming_request, _finalize_text, _build_chunk_request）：核心编排文件：接入分块入口、顺序派发、文本拼接、失败中止、流式处理全部在此完成，是本次变更的主路径。
- python/sglang/srt/entrypoints/openai/audio_chunking.py（模块 音频切分；类别 source；类型 core-logic；符号 find_split_point, split_audio_energy_aware）：新增的能量感知分块模块，定义切点搜索与整段切分算法，切分行为与 vLLM 旧转写端点逐位对齐。
- python/sglang/srt/entrypoints/openai/transcription_adapters/whisper.py（模块 转写适配；类别 source；类型 core-logic；符号 max_audio_clip_s, build_verbose_response_chunked, _parse_segments, parse_fused_output）：Whisper 适配器声明 30 秒窗口上限、实现分块 verbose_json 响应与带偏移的段解析，是分块能力的模型侧核心。
- python/sglang/srt/entrypoints/openai/transcription_adapters/base.py（模块 适配基类；类别 source；类型 data-contract；符号 max_audio_clip_s,

build_verbose_response_chunked, parse_fused_output）：定义适配器分块契约：max_audio_clip_s 默认 None 禁用分块，build_verbose_response_chunked 默认实现保证第三方适配器行为不回归。

- test/registered/unit/entrypoints/openai/test_serving_transcription.py（模块 转录服务；类别 test；类型 test-coverage；符号 _MockChunkTokenizerManager, generate_request, abort_request, _long_wav_bytes）：核心编排的 CPU 单测：mock TokenizerManager 模拟 rid 分配、abort 记录与并发计数，覆盖顺序拼接、失败中止、分割失败、fused 语言与无空格脚本场景。
- test/registered/unit/entrypoints/openai/test_audio_chunking.py（模块 音频切分；类别 test；类型 test-coverage；符号 TestFindSplitPoint, TestSplitAudioEnergyAware, _tone_with_silences）：分块算法本身的 CPU 单测：验证安静窗口选择、均匀能量回退、连续性、不重叠、波形保持与无静音推进。
- test/registered/openai_server/basic/test_serving_transcription.py（模块 端到端；类别 test；类型 test-coverage；符号 long_audio_wav_bytes, _assert_keywords, test_long_audio_transcribes_past_30s, test_long_audio_verbose_json_segment_offsets）：GPU 端到端验证：40 秒 WAV（30 秒静音 + 10 秒语音）保证语音完全落在 Whisper 窗口之外，覆盖 JSON、verbose_json 与流式三条路径。
- test/registered/unit/entrypoints/openai/test_whisper_adapter.py（模块 转写适配；类别 test；类型 test-coverage；符号 _FakeTokenizer, TestWhisperChunkedVerboseResponse）：验证分块 verbose_json 的段偏移与跨块 id 连续性，以及单块零偏移与未分块路径语义一致。
- python/sglang/srt/utils/common.py（模块 公共工具；类别 source；类型 dependency-wiring；符号 load_audio）：load_audio 入参类型扩展为支持 bytes，是分块解码与时长 fallback 的公共基础。

关键符号：find_split_point, split_audio_energy_aware, OpenAIServingTranscription.create_transcription, OpenAIServingTranscription._handle_chunked_non_streaming_request, OpenAIServingTranscription._finalize_text, OpenAIServingTranscription._build_chunk_request, OpenAIServingTranscription._abort_chunk_requests, OpenAIServingTranscription._generate_long_audio_stream, TranscriptionAdapter.max_audio_clip_s, TranscriptionAdapter.build_verbose_response_chunked, WhisperAdapter.max_audio_clip_s, WhisperAdapter.build_verbose_response_chunked, WhisperAdapter._parse_segments, WhisperAdapter.parse_fused_output, load_audio

关键源码片段

python/sglang/srt/entrypoints/openai/audio_chunking.py

新增的能量感知分块模块，定义切点搜索与整段切分算法，切分行为与 vLLM 旧转写端点逐位对齐。

```
"""Energy-aware audio chunking for ASR models with a bounded input window.
```

Whisper 的编码器只接受固定 30 秒窗口（3000 个 mel 帧），feature extractor
会把超长输入静默截断，所以超长音频必须先切成独立块。若盲目在窗口边界
下刀，可能切到单词中间，把缝两侧的转录都弄坏；因此在每个窗口尾部
1 秒的搜索区间里找 RMS 能量最低（最安静）的位置切。

行为刻意与 vLLM 的 ``OpenAISpeechToText._split_audio`` / ``_find_split_point``
对齐：块连续、不重叠，拼接回来就是完整波形，切分点与旧 vLLM 端点
逐位一致。
"""

# 每个 max-length 窗口尾部用于搜索低能量切点的区域长度（秒）。
SPLIT_SEARCH_WINDOW_S = 1.0

# RMS 能量按 100 ms（16 kHz 下 1600 个采样）为步长评估，
# 搜索区间内最安静的一个步长的起点即切点。
MIN_ENERGY_WINDOW_SIZE = 1600

def find_split_point(wav: np.ndarray, start_idx: int, end_idx: int) -> int:
    """返回 ``wav[start_idx:end_idx]`` 内最安静能量窗口的起始下标（绝对值）。

    循环边界故意不评估搜索区间的最后一个步长：这是照抄 vLLM 的
    ``_find_split_point``，从而保证切分点与旧 vLLM 转写端点完全一致。
    """
    segment = wav[start_idx:end_idx]
    min_energy = math.inf
    quietest_idx = start_idx
    for i in range(0, len(segment) - MIN_ENERGY_WINDOW_SIZE, MIN_ENERGY_WINDOW_SIZE):
        window = segment[i : i + MIN_ENERGY_WINDOW_SIZE]
        energy = (window**2).mean() ** 0.5
        if energy < min_energy:
            quietest_idx = i + start_idx
            min_energy = energy
    return quietest_idx

def split_audio_energy_aware(
    audio_data: bytes,
    max_clip_s: float,
    sample_rate: int = 16000,
) -> Tuple[List[bytes], List[float]]:
    """把音频切成不超过 ``max_clip_s`` 秒的连续 WAV 块。

    解码并重采样到 16 kHz 单声道后，按 ``max_clip_s`` 步长前进，在每个
    步长末尾的 ``SPLIT_SEARCH_WINDOW_S`` 内找最低能量点作为切点，保证
    切在停顿处而非单词中间。返回 ``(chunk_bytes, offsets_s)``，
    ``offsets_s[i]`` 是第 i 块在原始音频中的起始时间。
    """
    if not audio_data:

```
        raise ValueError("audio_data is empty")
    audio = load_audio(audio_data, sr=sample_rate, mono=True)
    chunk_size = int(sample_rate * max_clip_s)
    search_size = int(sample_rate * SPLIT_SEARCH_WINDOW_S)
    total = audio.shape[-1]

    raw_chunks: List[np.ndarray] = []
    offsets_s: List[float] = []
    i = 0
    while i < total:
        offsets_s.append(i / sample_rate)
        if i + chunk_size >= total:
            raw_chunks.append(audio[i:]) # 剩余不足一个窗口，整段收尾
            break
        search_start = i + chunk_size - search_size
        search_end = min(i + chunk_size, total)
        split_point = find_split_point(audio, search_start, search_end)
        raw_chunks.append(audio[i:split_point])
        i = split_point # 切点成为下一块起点，块与块无缝衔接

    chunks: List[bytes] = []
    for chunk in raw_chunks:
        buf = io.BytesIO()
        sf.write(buf, chunk, sample_rate, format="WAV")
        chunks.append(buf.getvalue())
    return chunks, offsets_s
```

python/sglang/srt/entrypoints/openai/transcription_adapters/whisper.py

Whisper 适配器声明 30 秒窗口上限、实现分块 verbose_json 响应与带偏移的段解析，是分块能力的模型侧核心。

```
    def build_verbose_response_chunked(
        self,
        request: TranscriptionRequest,
        text: str,
        rets: List[dict],
        chunk_offsets_s: List[float],
        tokenizer,
        usage: TranscriptionUsage,
    ) -> TranscriptionVerboseResponse:
        """由多个块的结果组装 verbose_json 响应。

        Whisper 的时间戳 token 相对各自块的 30 秒窗口，所以每个块的段
        都要按块在原始音频中的起始时间偏移；segment id 跨块连续编号。
        ``text`` 直接用 serving 层已拼接好的文本——若在这里用 output_ids
        重新解码再用 ASCII 空格 join，会破坏中文、日文、泰文等无空格
        脚本（模型本来就没输出空格）。
        """
        segments: list[TranscriptionSegment] = []
```

```
    for ret, offset_s in zip(rets, chunk_offsets_s):
        # _parse_segments 对每一块的 timestamp token 加上 time_offset_s，
        # 并把起始 seg_id 接在已累积段数之后。
        _, part_segments = self._parse_segments(
            ret.get("output_ids", []),
            tokenizer,
            time_offset_s=offset_s,
            seg_id_start=len(segments),
        )
        segments.extend(part_segments)
    return TranscriptionVerboseResponse(
        language=request.language,
        duration=round(request.audio_duration_s, 2),
        text=text,
        segments=segments,
        usage=usage,
    )
```

评论区精华

合并者 JustinTong0323 的审查贯穿了从实现到测试的多个关键决策点，且多数意见已在最终提交中落实：

1. 并发扇出风险（性能）："Serialize these chunk generations or enforce an explicit configurable bound; one task per user-controlled 30-second chunk lets a single long upload fan out into arbitrarily many tokenizer/GPU requests, and the route has no duration limit." 最终实现改为顺序派发（测试断言 `max_active_dispatches == 1`），缓解了并发放大，但未引入总时长 / 块数上限配置。
2. 分割失败回退（正确性）："Return an error when splitting fails instead of sending the original over-window audio; this fallback knowingly re-enters Whisper's 30-second truncation path and can return HTTP 200 with an incomplete transcript." 最终实现改为返回 400 错误，不再静默回退。
3. 无空格脚本拼接（正确性）："Use the already stitched text here and parse output_ids only for segments; joining chunk texts with an ASCII space corrupts spaceless scripts such as Chinese, Japanese, and Thai." 最终 `build_verbose_response_chunked` 直接用 serving 层已拼接文本，`_finalize_text` 通过 `strip=False` 保留模型边界空白。
4. 语言选择语义（正确性）："Select the reported language from the first chunk with non-empty visible text instead of the first parsed prefix; a leading-silence chunk can otherwise lock verbose_json.language to an arbitrary language." 最终 `_finalize_text` 用 `visible.strip()` 判断非空，首个非空文本块的语言胜出。
5. 测试 mock 缺陷（测试）："Set asr_max_concurrent_sessions to an integer on this mock; OpenAIServingTranscription passes it to asyncio.Semaphore, so the registered CPU test currently raises TypeError before any chunk orchestration runs." 已修复为 `asr_max_concurrent_sessions=32`。

6. 设计复核（设计）："This is a design choice, what's your opinion? @shenxiul"——reviewer 在语言选择策略上主动征询作者意见。最终批准意见："CI base green; extra-CI gptq failure is a pre-existing main regression unrelated to this PR. LGTM."

- 分块生成的并发扇出风险 (performance): 最终实现改为顺序派发（单测断言 max_active_dispatches == 1），并发放大被消除，但未引入总时长 / 块数上限配置。
- 分割失败时回退发送原始音频 (correctness): 最终实现改为返回 400 错误响应，不再回退到原始音频。
- 无空格脚本的拼接语义 (correctness): 最终 build_verbose_response_chunked 直接用已拼接文本，_finalize_text 以 strip=False 保留模型边界空白。
- 自动检测语言应取首个非空文本块 (correctness): 最终 _finalize_text 以 visible.strip() 判断非空，首个非空文本块的语言胜出。
- 测试 mock 的 Semaphore 类型错误 (testing): 已修复为 asr_max_concurrent_sessions=32。
- 语言选择策略的设计征询 (design): 采用首个非空可见文本块的语言；整体审查最终以 LGTM 批准。

风险与影响

- 风险:
  1. 资源放大无上限（性能 / 可用性）：虽然块改为顺序派发，但单个长上传仍会生成 ceil(时长 /30) 个独立 tokenizer/GPU 请求（如 1 小时音频产生 120 个块），总延迟与计算成本随时长线性放大，路由层无总时长或块数上限。reviewer 在讨论中明确指出这一点，属于部分缓解。
  2. 核心路径变更（回归）：create_transcription 与 _handle_non_streaming_request 是 /v1/audio/transcriptions 的主路径；_finalize_text 抽取后 strip=True 时必须保持与原内联逻辑完全一致，否则非分块请求的文本 / 语言行为会漂移。_parse_segments 新增参数为默认值，非分块路径不受影响。
  3. 适配器兼容性（接口）：TranscriptionAdapter 新增 build_verbose_response_chunked 默认实现（空段列表）。第三方自定义适配器若未覆写，在超窗音频且 max_audio_clip_s 非 None 时会丢失段时间戳；默认 None 禁用了该路径，风险可控。
  4. 音频解码稳定性：_get_audio_duration 的 fallback 与分块都会全量解码音频，asyncio.to_thread 已规避事件循环阻塞，但线程池饱和可能在高并发下排队；sf.write 重编码为 PCM16 WAV 存在量化误差（测试以 atol=2/32768 接受）。
  5. 测试稳定性（CI）：GPU 端到端测试依赖远程下载 sgl-test-files 的 mp3 并拼接 40 秒 WAV，网络抖动可能导致 flaky；est_time 从 60 提升到 90 秒。
  6. 安全：无新攻击面，但长音频上传可放大 GPU 资源消耗（reviewer 已关注），路由层缺少时长限制的现状仍在。 - 影响：对用户：Whisper 模型下超过 30 秒的音频从 " HTTP 200 但内容被静默截断 " 变为完整转录；verbose_json 段时间戳正确反映原始音频时间轴（可超过 30 秒）；流式路径同步可用；分割失败时获得显式 400 错误而非静

默丢失。对系统：转录路径新增 CPU 侧解码 / 分块 / 重编码开销，长音频按块多次进入多模态编码与解码，计算成本与时长线性相关，是此前 " 一次编码但丢内容 " 之外的增量成本。对团队：TranscriptionAdapter 抽象形成 " 适配器声明窗口上限 → serving 层通用分块 → 适配器负责段偏移与文本拼接 " 的扩展模式，Qwen3-ASR 等其他适配器只需声明 max_audio_clip_s 即可复用分块能力。 - 风险标记：核心路径变更，长音频资源放大无上限，多适配器接口扩展，测试依赖外部音频资源

关联脉络

- PR #34892 feat: add safeguards for remote media URLs: 同属 multimodal 输入侧防御 / 审计方向，且都触及 python/sglang/srt/utils/common.py 的公共工具面；本 PR 扩展 load_audio 支持 bytes 输入，为分块音频的 CPU 侧处理提供基础。