```
import gc

import threading

from typing import List, Optional


import torch

import whisperx

import os


from dotenv import load_dotenv


load_dotenv()


hf_token = os.getenv("HF_TOKEN")



class WhisperTranscriber:

    """

    A class for transcribing audio using the Whisper ASR system.


    Args:

        device (str): The device to use for computation (default: "cuda").

        compute_type (str): The compute type to use (default: "float16").

        batch_size (int): The batch size for transcription (default: 16).

        hf_token (Optional[str]): The Hugging Face authentication token (default: None).

        audio_file (Optional[str]): The path to the audio file to transcribe (default: None).

        audio_files (Optional[List[str]]): A list of paths to audio files to transcribe (default: None).
```

```
"""

def __init__(
    self,
    device: str = "cuda",
    compute_type: str = "float16",
    batch_size: int = 16,
    hf_token: Optional[str] = hf_token,
    audio_file: Optional[str] = None,
    audio_files: Optional[List[str]] = None,
):
    self.device = device
    self.compute_type = compute_type
    self.batch_size = batch_size
    self.hf_token = hf_token
    self.lock = threading.Lock()
    self.audio_file = audio_file
    self.audio_files = audio_files

def load_and_transcribe(self, audio_file):
    """
    Load the Whisper ASR model and transcribe the audio file.

    Args:
        audio_file (str): The path to the audio file.
```

```
    Returns:
        dict: The transcription result.
    """
    with self.lock:
        model = whisperx.load_model(
            "large-v2", self.device, compute_type=self.compute_type
        )

    audio = whisperx.load_audio(audio_file)
    result = model.transcribe(audio, batch_size=self.batch_size)
    print(result["segments"])  # Before alignment

    with self.lock:
        del model
        gc.collect()
        torch.cuda.empty_cache()

    return result

def align(self, segments, language_code):
    """
    Align the transcribed segments with the audio using the Whisper alignment model.

    Args:
        segments (list): The transcribed segments.
        language_code (str): The language code.
```

```
Returns:
    dict: The alignment result.
"""
with self.lock:
    model_a, metadata = whisperx.load_align_model(
        language_code=language_code, device=self.device
    )

audio = whisperx.load_audio(self.audio_file)
result = whisperx.align(
    segments,
    model_a,
    metadata,
    audio,
    self.device,
    return_char_alignments=False,
)
print(result["segments"])  # After alignment

with self.lock:
    del model_a
    gc.collect()
    torch.cuda.empty_cache()

return result
```

```python
def diarize_and_assign(self, audio_file, segments):
    """
    Diarize the audio and assign speaker IDs to the segments.

    Args:
        audio_file (str): The path to the audio file.
        segments (list): The aligned segments.

    Returns:
        dict: The diarization and assignment result.
    """
    with self.lock:
        diarize_model = whisperx.DiarizationPipeline(
            use_auth_token=self.hf_token, device=self.device
        )

    diarize_segments = diarize_model(audio_file)
    result = whisperx.assign_word_speakers(diarize_segments, segments)
    print(diarize_segments)
    print(result["segments"])  # Segments now assigned speaker IDs

    return result

def process_audio(self, audio_file: str):
    """
```

```
        Process the audio file by transcribing, aligning, and diarizing it.

        Args:
            audio_file (str): The path to the audio file.

        Returns:
            dict: The final result.
        """
        transcription_result = self.load_and_transcribe(audio_file)
        aligned_result = self.align(
            transcription_result["segments"], transcription_result["language"]
        )
        final_result = self.diarize_and_assign(audio_file, aligned_result)
        return final_result

    def run(self, audio_file: str):
        """
        Run the audio processing pipeline.

        Args:
            audio_file (str): The path to the audio file.

        Returns:
            dict: The final result.
        """
        return self.process_audio(audio_file)
```

```
# Instantiate the WhisperTranscriber
model = WhisperTranscriber()

# Run the audio processing pipeline
result = model.run("song.mp3")
```