# Transcribe Audio with multiple speakers

**Code**

```
import whisperx
import os
import tkinter as tk
from tkinter import filedialog, ttk, scrolledtext
from pydub import AudioSegment
import logging
import subprocess
import sys
import shutil
import warnings

# Suppress deprecation warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logger = logging.getLogger()

# Force WhisperX to use local VAD model to avoid redirect error
os.environ["WHISPERX_VAD_MODEL_PATH"] = r"D:\\PY\\models\\vad\\pytorch_model.bin"

# Redirect logging to GUI log window
class TextHandler(logging.Handler):
    def __init__(self, text_widget):
```

```
        super().__init__()
        self.text_widget = text_widget

    def emit(self, record):
        try:
            msg = self.format(record)
            if self.text_widget.winfo_exists():  # Check if widget still exists
                self.text_widget.insert(tk.END, msg + '\n')
                self.text_widget.see(tk.END)
                self.text_widget.update()
        except tk.TclError:
            pass  # Ignore errors if GUI is closed

# Supported audio formats
SUPPORTED_FORMATS = ['.wav', '.m4a', '.mp3', '.mp4', '.mkv']

# Function to convert audio to WAV if not already WAV

def convert_to_wav(input_file, output_dir, temp_dir):
    file_ext = os.path.splitext(input_file)[1].lower()
    if file_ext == '.wav':
        logger.info(f"Input file {input_file} is already WAV. No conversion
needed.")
        return input_file

    output_file = os.path.join(temp_dir,
os.path.splitext(os.path.basename(input_file))[0] + '.wav')
    try:
        format_param = 'matroska' if file_ext == '.mkv' else file_ext[1:]
        audio = AudioSegment.from_file(input_file, format=format_param)
        audio.export(output_file, format='wav')
        logger.info(f"Converted {input_file} to {output_file}")
        return output_file
    except Exception as e:
        logger.error(f"Error converting {input_file} to WAV: {str(e)}")
        try:
            result = subprocess.run(
```

```
                ['ffmpeg', '-i', input_file],
                capture_output=True, text=True, check=False
            )
            logger.error(f"FFmpeg output: {result.stderr}")
        except Exception as ffmpeg_e:
            logger.error(f"Could not run FFmpeg to diagnose file:
{str(ffmpeg_e)}")
        raise

# Main transcription function

def transcribe_audio(input_file, output_dir, temp_dir):
    wav_file = None
    try:
        # Convert to WAV if necessary
        wav_file = convert_to_wav(input_file, output_dir, temp_dir)

        # Load the model
        logger.info("Loading WhisperX model...")
        asr_options = {
            "max_new_tokens": 448,
            "clip_timestamps": False,
            "hallucination_silence_threshold": 0.6
        }
        model = whisperx.load_model("base", device="cpu", compute_type="float32",
asr_options=asr_options)

        # Transcribe
        logger.info("Transcribing audio...")
        result = model.transcribe(
            wav_file,
            batch_size=16,
            language=None,
        )

        # Align timestamps
        logger.info("Aligning timestamps...")
```

```
        model_a, metadata = whisperx.load_align_model(language_code="en",
device="cpu")
        result = whisperx.align(result["segments"], model_a, metadata, wav_file,
device="cpu")

        # Diarization
        logger.info("Performing diarization...")
        hf_token = "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"  # Replace with your
token
        try:
            diarize_model =
whisperx.diarize.DiarizationPipeline(use_auth_token=hf_token, device="cpu")
            diarize_segments = diarize_model(wav_file)
            result = whisperx.assign_word_speakers(diarize_segments, result)
        except AttributeError:
            logger.warning("DiarizationPipeline not available in this whisperx
version. Skipping diarization.")
            for segment in result["segments"]:
                segment["speaker"] = "Unknown"

        # Save output
        output_file = os.path.join(output_dir, "transcription_with_speakers.txt")
        with open(output_file, "w") as f:
            for segment in result["segments"]:
                start = segment["start"]
                end = segment["end"]
                text = segment["text"]
                speaker = segment.get("speaker", "Unknown")
                f.write(f"[{start:.2f}s - {end:.2f}s] Speaker {speaker}:
{text}\n")

        logger.info(f"Transcription complete. Output saved to {output_file}")

    except Exception as e:
        logger.error(f"Error during transcription: {str(e)}")
        raise
    finally:
```

```
        if wav_file and wav_file != input_file and os.path.exists(wav_file):
            try:
                os.remove(wav_file)
                logger.info(f"Removed temporary WAV file: {wav_file}")
            except Exception as e:
                logger.warning(f"Could not remove temporary WAV file {wav_file}:
{str(e)}")

# GUI Application

class TranscriptionApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Audio Transcription")
        self.root.geometry("600x600")

        tk.Label(root, text="Input Audio File:").pack(pady=5)
        self.input_entry = tk.Entry(root, width=50)
        self.input_entry.pack(pady=5)
        tk.Button(root, text="Browse", command=self.browse_input).pack(pady=5)

        tk.Label(root, text="Output Directory:").pack(pady=5)
        self.output_entry = tk.Entry(root, width=50)
        self.output_entry.pack(pady=5)
        tk.Button(root, text="Browse", command=self.browse_output).pack(pady=5)

        tk.Label(root, text="Temporary Directory (for WAV files):").pack(pady=5)
        self.temp_entry = tk.Entry(root, width=50)
        self.temp_entry.insert(0, "D:\\PY\\temp")
        self.temp_entry.pack(pady=5)
        tk.Button(root, text="Browse", command=self.browse_temp).pack(pady=5)

        tk.Button(root, text="Transcribe",
command=self.start_transcription).pack(pady=10)

        tk.Label(root, text="Log:").pack(pady=5)
        self.log_text = scrolledtext.ScrolledText(root, height=10, width=60,
```

```
wrap=tk.WORD)
        self.log_text.pack(pady=5)

        text_handler = TextHandler(self.log_text)
        text_handler.setFormatter(logging.Formatter('%(asctime)s - %(levelname)s -
%(message)s'))
        logger.addHandler(text_handler)

    def browse_input(self):
        file_path = filedialog.askopenfilename(filetypes=[("Audio/Video Files",
"*.wav *.m4a *.mp3 *.mp4 *.mkv")])
        if file_path:
            self.input_entry.delete(0, tk.END)
            self.input_entry.insert(0, file_path)

    def browse_output(self):
        dir_path = filedialog.askdirectory()
        if dir_path:
            self.output_entry.delete(0, tk.END)
            self.output_entry.insert(0, dir_path)

    def browse_temp(self):
        dir_path = filedialog.askdirectory()
        if dir_path:
            self.temp_entry.delete(0, tk.END)
            self.temp_entry.insert(0, dir_path)

    def start_transcription(self):
        input_file = self.input_entry.get()
        output_dir = self.output_entry.get()
        temp_dir = self.temp_entry.get()

        if not input_file or not output_dir or not temp_dir:
            logger.error("Please select input file, output directory, and
temporary directory.")
            return
```

```
        if not os.path.exists(input_file):
            logger.error(f"Input file {input_file} does not exist.")
            return

        if os.path.splitext(input_file)[1].lower() not in SUPPORTED_FORMATS:
            logger.error(f"Unsupported file format. Supported formats: {',
'.join(SUPPORTED_FORMATS)}")
            return

        if not os.path.exists(output_dir):
            try:
                os.makedirs(output_dir)
                logger.info(f"Created output directory: {output_dir}")
            except Exception as e:
                logger.error(f"Could not create output directory {output_dir}:
{str(e)}")
                return

        if not os.path.exists(temp_dir):
            try:
                os.makedirs(temp_dir)
                logger.info(f"Created temporary directory: {temp_dir}")
            except Exception as e:
                logger.error(f"Could not create temporary directory {temp_dir}:
{str(e)}")
                return

        import threading
        threading.Thread(target=transcribe_audio, args=(input_file, output_dir,
temp_dir), daemon=True).start()

# Main execution
if __name__ == "__main__":
    try:
        subprocess.run(['ffmpeg', '-version'], capture_output=True, check=True)
    except (subprocess.CalledProcessError, FileNotFoundError):
        logger.error("FFmpeg is not installed or not found. Please install FFmpeg
```

```
to proceed.")
        sys.exit(1)

    default_temp_dir = "D:\\PY\\temp"
    if not os.path.exists(default_temp_dir):
        try:
            os.makedirs(default_temp_dir)
        except Exception as e:
            logger.error(f"Could not create default temporary directory
{default_temp_dir}: {str(e)}")
            sys.exit(1)

    root = tk.Tk()
    app = TranscriptionApp(root)
    root.mainloop()

    try:
        if os.path.exists(default_temp_dir):
            shutil.rmtree(default_temp_dir)
            logger.info(f"Cleaned up default temporary directory:
{default_temp_dir}")
    except Exception as e:
        logger.warning(f"Could not clean up default temporary directory
{default_temp_dir}: {str(e)}")
```

# □□ WhisperX Offline Transcription Setup with GUI

## □□ Summary

This guide details how to set up and patch WhisperX to transcribe long audio files (MP3, MP4, etc.) **offline** using a GUI-based Python app, bypassing VAD model downloads and network dependencies.

---

# □□ Project Overview

- **Platform**: Python 3.12 with WhisperX + SpeechBrain
- **Goal**: Offline GUI app for audio transcription with speaker diarization
- **Input**: Audio/Video file
- **Output**: Timestamped transcript with speaker labels

---

# □ Key Features

- No network requirement for VAD
- GUI with file selection and logging
- Long file support (tested on 3hr+ MP3)
- Speaker diarization using `speechbrain`
- Chunk-based transcription (VAD manually bypassed)

---

# □□ Setup Instructions

## 1. □□ Python Environment

bash

```
python -m venv .venv
.venv\Scripts\activate
pip install whisperx torchaudio pydub tkinter speechbrain
```

## 2. □□ Folder Structure

bash

```
/transcriber/
├── transcribe.py            # GUI application
├── models/vad/pytorch_model.bin  # Downloaded manually
├── .venv/...
```

## 3. □□ Environment Variable (Set in `transcribe.py`)

python

```
os.environ["WHISPERX_VAD_MODEL_PATH"] = r"D:\\PY\\models\\vad\\pytorch_model.bin"
```

## 4. □□ GUI Usage

Run:

bash

`python transcribe.py`

Then:

- Select audio file
- Choose output and temp folders
- Click **Transcribe**

---

# □□ WhisperX Modifications

## □ `vad.py` Patch

- Replaced Hugging Face model download with local load
- Stubbed `merge_chunks()` for compatibility

python
```
def load_vad_model(...):
    model_fp = os.environ.get("WHISPERX_VAD_MODEL_PATH")
    if not model_fp or not os.path.exists(model_fp):
        raise FileNotFoundError("Local VAD model path invalid.")
    print(f"Using local VAD model at: {model_fp}")
    bundle = torchaudio.pipelines.HUBERT_BASE
    return bundle.get_model().to(device).eval()

def merge_chunks(chunks, *args, **kwargs):
    return chunks
```

## □ `asr.py` Patch

- Skipped internal VAD model logic
- Injected manual chunking (30s per segment)

**Modified `transcribe()` inside `FasterWhisperPipeline`:**

python
```
duration = audio.shape[0] / SAMPLE_RATE
chunk_duration = 30.0
vad_segments = []
start = 0.0
while start < duration:
    end = min(start + chunk_duration, duration)
    vad_segments.append({"start": start, "end": end})
    start = end
```

---

# □□ Issues Resolved

| Issue | Resolution |
| --- | --- |
| `TranscriptionOptions.__new__()` missing args | Manually passed `asr_options` with required fields |
| HTTP 301 for VAD model | Replaced remote load with offline `.bin` path |
| `'dict' has no attribute 'ndim'` | Dummy VAD model returned incompatible type → fully bypassed |
| `vad_segments` unexpected argument | Removed invalid param from `transcribe()` call |
| `input shape (1, 80, 782456)` too large | Manual chunking into 30s segments |

---

# □□ Final Notes

- Long audio files (2–3 hrs) may take 30–60+ minutes depending on CPU speed
- Recommended: run on GPU or chunk files into 1-hour batches
- Supports `.mp3`, `.wav`, `.mp4`, `.mkv`, `.m4a`

---

# □□ Files to Backup for Future Use

- `transcribe.py`
- Patched: `whisperx/vad.py`
- Patched: `whisperx/asr.py`
- `pytorch_model.bin` saved locally

---

# □□ Future Improvements

- Optional: add GUI dropdown for model size (base/medium/large)
- Optional: progress bar and chunk counters
- Optional: automatic chunked transcription and merge

---

Revision #3
Created 2025-05-29 03:07:56 UTC by Slitzer
Updated 2025-05-29 03:10:12 UTC by Slitzer