Credits

My translation script was based on a script by porgate55555

https://www.akiba-online.com/threads/whisper-and-its-many-forms.2142559/post-4867905

The various versions of WhisperJAV are by mei2

https://github.com/meizhong986/WhisperJAV

The garbage list to remove repetitions and hallucinations from transcribed files was originally posted by ironfevers

I've incorporated suggestions and changes to my scripts and workflow based on many useful posts on Akiba-Online

Install required software

- Python (select the option to add Python to your PATH when installing)

  https://www.python.org/

- GIT

  https://github.com/git-guides/install-git

- Ffmpeg (you probably also need to add Ffmpeg to your PATH)

  https://www.ffmpeg.org/

- Subtitle Edit

  https://www.nikse.dk/subtitleedit

- You may need to reboot after installations to make sure the PATHs are applied.

Extract and (optionally) pre-process audio

Purpose: Create audio file(s) suitable for transcription.

- Clone https://github.com/TotoTheDog0/audioPreprocess
- Follow the README to install and run the audio_processor.py script.
- In the UI drag-and-drop or select the video file(s) to be processed. Select normalisation and slow-down options (if wanted) and output folder. I usually output audio files directly to my Google Drive so they're in the correct location for the transcription step.
  I almost always output at least one normalised file and one normalised and slowed-down file. I find dynamic normalisation generally provides the best transcription (almost always better than peak normalisation). For files that are speech-heavy or where the dialog is quiet, speech

normalisation can be better. There's nothing, apart from time, stopping you outputting files with multiple normalisation methods.

I find using both original-speed and slowed-down audio files means, between the files, fewer lines get skipped or miss-transcribed.

- Start processing and wait. It can take quite a long time to process the audio (especially speech normalisation) – maybe 10 minutes per audio file for long videos. You'll get a progress indication in the UI and terminal window.
- Files are suffixed to show how they were processed. E.g.,

Filename.mp3	No processing
Filename_sl0w.mp3	Slowed to 60% speed
Filename_snorm.mp3	Speech normalised. Original speed.
Filename_dnorm_sl0w.mp3	Dynamic normalised. Slowed to 60% speed

The '_sl0w' suffix has a zero not an 'O'. This is deliberate as it provides an uncommon pattern used by the translation script to recognise slowed-down files.

- There is an 'audio_processor.log' file saved to the output folder which can be useful for diagnostics if something goes wrong.

Transcribe audio to Japanese subtitle(s)

Purpose: Create Japanese srt subtitle files for subsequent translation.

I use a slightly modified version of v0.7b of the Google Colab notebook from https://github.com/meizhong986/WhisperJAV

The modifications allow batch-processing capabilities and shut down the session on completion. I won't provide that modified notebook as it doesn't belong to me. Also, the newer versions of the notebook, or command line or GUI versions, almost certainly do a better a job (I just haven't yet got around to learning how best to use the newer versions of WhisperJAV).

- If not already there, move the extracted audio files to the appropriate folder on your Google Drive (where the WhisperJAV notebook looks for the files).
- Process your audio files with the Whisper notebook. If you have a free Colab account, you may only be able to process one or two files before you reach your limit and get disconnected.
- I won't provide detailed settings for the Whisper transcription here – you're better off looking at the documentation on the repo.
- I never use Whisper for translation as the quality of translation is poor. If you have access to an external translation service within Whisper, it may be okay – I've never tried.

- Once processing is completed you should have a Japanese srt subtitle file for each audio file in the same Google Drive folder as your audio files. Of course, if you have slowed-down audio files those generated srt files will also be "slowed". The slowing will be corrected in the translation step. The transcribed files are likely to have many hallucinations. These can mostly be corrected in the translation step.

Clean and translate input subtitles to English (or another language)

Purpose: Create correct speed, cleaned, English (or other language) srt subtitle files for subsequent manual editing.

- Clone https://github.com/TotoTheDog0/subsTranslate
- Follow the README to install, edit the .env and run the subsTranslate.py script.
- In the UI select the input folder. I usually just leave the input subtitles in the Google Drive folder from the Whisper transcription step.
- In the UI select which model (or models, up to three) and provider you wish to use, and the order of priority. If API calls fail or time-out the script will move on, after three attempts, to the next model in the list.
- My current preferred model for translation is DeepSeek v3.2 via OpenRouter. Selecting the DeepSeek Chat model via DeepSeek API will use the latest DeepSeek model (currently v3.2) but I've found the DeepSeek API unreliable. You can also change the model temperature (works for DeepSeek models but maybe not others).
- In the UI also select whether you want the input subtitles cleaned. If cleaning is selected, the script looks up garbage_list.json and deletes matching garbage (hallucinations and repeated characters) before translating. I almost always select the option to clean. You can also select the option to search and replace text patterns, in subs_replace.json, after translation.
- Start processing. You'll get progress indications in the terminal window.
- First the script corrects the timing of any slowed-down files (identified by the '_sl0w' suffix) by multiplying all timecodes by 0.6.
- The script sends 'chunks' (100 subtitle lines) plus the translation prompt (you can see that within the script) to the model and reassembles translated lines into a new, translated subtitle file. If the script detects untranslated lines, it will resend them for translation in smaller chunks. If translation of lines fails three times the original lines are used in the translated subtitle (I very seldom have translation fail in three attempts). The model is also prompted to include brief contextual and cultural notes [enclosed in square brackets] in the translated subtitle. You can change the prompt within the script if you want a different style of translation.
- If you want to translate to a language other than English, change the prompt in the script.

- Once translation is completed, you should have translated files in a 'Translated' subfolder of your input folder. Translated files are prefixed with 'Translated_'
- There is a log file saved which can be useful for diagnostics if something goes wrong.
- One useful thing to note… the first thing the script does, before translation, is correct the timing of slowed srt files. So, if something goes wrong with the translation, the slowed srt files will no longer be slowed down (i.e., their timing will now be correct). So, if you have a problem during translation, rename any input subtitle files that have the '_sl0w' suffix so their timing won't be adjusted again when retrying translation. If this happens, I just change the '_sl0w' suffix to '_slow' so I can still see which subtitles were generated from slowed audio.

Manually edit subtitle(s) to correct, complete and clarify them

Purpose: Manually correct and refine automatically translated subtitles.

This is the part of the workflow that requires the most hands-on time.

I use Subtitle Edit to do this step as you can open two subtitle files side by side and move lines from one file to the other.

- Move all the translated srt files to the folder containing the original video.
- I usually delete the 'Translated_' prefix from a translated subtitle that was not generated from slowed audio and open it in Subtitle Edit. Then save it as a vtt subtitle file (because you can add formatting and positioning to vtt files). If the subtitle is named the same as the video, but with suffix(es) then you'll also get a video preview in Subtitle Edit.
- Then open another translated file beside the vtt file. This other file might be from a slowed audio transcription + translation and/or a file that was normalised using a different method to the vtt file.
- Then go through files line-by-line and make corrections or refinements or move lines from the second file to the vtt file. Having two files side-by-side gives two points of reference to choose the "best" transcribed and translated lines. This can be a quick or slow process – depending on how good/bad the automatic translations are and how fussy you are.
- If you're really fussy and have generated more than two versions of the subtitles (e.g., maybe you generated versions from both dynamically and speech normalised audio), you can repeat the manual comparison process with other files – using the vtt file as the "master" file each time.
- Once I'm happy (is anyone ever <u>really</u> happy?) with the quality of the master file I'll use the 'merge lines with the same text' and 'fix common errors' options in Subtitle Edit, then a spell check to do a final cleanup of the file.
- If I'm being really, really fussy I might manually add translations of on-screen captions to the vtt file. Usually, I'd just take screen snips of the captions and translate the snip using Google

Translate. Because I use vtt format, which can handle positioning and basic text formatting, I like to format on-screen captions in italics and maybe position them in a different part of the screen to the speech subtitles.