PONY PRESERVATION PROJECT

- VOICE -

A Guide

written by **twiggles** !!NPm5BKKNxx4 and many wonderful anons

maintained by Anon

RIP **twiggles** !!NPm5BKKNxx4

current thread

The Pony Preservation Project was started for the preservation of FiM. Please keep non-FiM content such as EQG and G5 in the appropriate threads.

Overview

The Pony Voice Preservation Project is a collaborative effort by /mlp/ to create quality AI text to speech generators for as many My Little Pony: Friendship is Magic characters as possible.

Nine years into the ride and not long before it is scheduled to end, we have reached an age of technology where a trained neural network can make [Trump and Obama recite Steamed Hams](), or make [Jordan Peterson perform Lose Yourself]() using nothing but recordings of their voices. We have nine seasons, five movies, and a leak's worth of voice material available to us in high quality. There is more public research into machine learning than ever before giving us a strong foundation to build upon. By the end of the year the show will have ended leaving the future of the community unclear. Now is the perfect time for this project.

The MLP community has always been extremely creative, making an immense amount of fan content over the span of almost nine years. Having the ability to generate voice lines for a character without needing their voice actor has incredible implications for fan works, and could ensure that new content is still made well beyond the end of the show.

This guide will explain how we plan to accomplish our goal, where we are now, and how you — the person reading this right now — can join in and contribute.

Current Project Goals

- Improve Text-to-Speech
    - Anons (mainly Vul) are looking into ways of improving voice quality.
    - Anons (mainly Cookie) are looking into ways of improving emotion prediction from out-of-domain text.
    - If you have knowledge of the subject, share it in the thread.
- Create a [dataset of show scenes](#) as plain text
    - so we can generate original MLP episodes using AI.
    - 1. Convert the show into a script that LLM's [ChatGPT] can generate
    - 2. Feed the script into Text-to-Speech, Text-to-SFX and Text-to-Video models to create episodes.
- Encourage archiving
    - If you like what the project has done so far, no better way to back things up than to do it yourself. Download a copy of the many resources today!
- Encourage use of the tools
    - Much effort has gone into the creation of this, let's see some results ^:)

How to Contribute

General guide to working independently

This section explains how to contribute INDEPENDENTLY to the Pony Preservation Project. You can do these things without interacting at all with people already working on the Preservation Project.

How to contribute to the Pony Preservation Project:

- Anyone with godlike patience - Collect, organize, label, and maintain pony data in datasets
- AI and data people - Use the data, help fix errors, find good papers
- Anyone artistically motivated - Make original content, ponifications, artisanal shitposts
- Anyone that can keep up with what's going on - Organize content, make sure people know how to help, write tutorials
- Developers - Make everyone's lives easier, make collaboration easier
- Anyone able to reach an audience - Spread the good stuff, and make people love ponies and AI

This project is driven by the love of ponies and AI, and the contributions of many anons. We can use all the help we can get. The more people we have working on this, the better our wAIfus will be!

See the panel to get up-to-speed on the project:

- Slides: Google Docs
- Video: YouTube Mega (stream only) Mega (chat) Mega (chat, low res) PonyTube (Chat)
- Contains an overview of the Pony Preservation Project and information on contributing
- Presented on Jun 13, 2020

Keep up with AI blogs to stay knowledgeable:

- DeepMind's blog: science & futuristic capabilities possible today
- Google's AI blog: what will be accessible in a few years
- Facebook's AI blog: practical suggestions for developers
- Microsoft's AI research blog: practical research (usually)
- OpenAI's blog: great new ways of thinking about AI
- Uber's AI blog: getting AI to handle greater complexity

Learn to use AI developer tools:

- PyTorch, Julia, NumPy: These are currently the best frameworks for creating neural networks. Neural networks are hard to use properly! Follow this >>33987023 wise anon's guide. (https://desuarchive.org/mlp/thread/33963949/#q33987023)
- scikit-learn, Apache Lucene, Apache MLlib: Tools for creating decision trees and information retrieval systems.
- C4.5, C5.0, Classification and Regression Trees (CART), Random Forest: Algorithms for creating decision trees.
- Collaborative filtering, Latent Dirichlet Allocation, TF-IDF: Algorithms for information retrieval. Read up on TF-IDF before implementing your own!
- SpaCy AllenNLP, HuggingFace, scikit-learn: Tools for natural language (text) processing.
- Kaldi, Parselmouth, librosa: Tools for working with audio files.
- OpenCV, scikit-learn: Tools for working with image files.
- PredictionIO: A tool for creating recommendation engines.

Organize any of this pony data and add it to this doc + post it on the thread:

- Greentext dumps, fanfic dumps, thread dumps, image dumps, show BGM, fan music, YouTube playlists, comment histories, compilations, "Best of" voting results, blog posts
- Show, community, fanfics, videos, images, games, plushies

Dev & audio anons: figure out what's wrong with our clips:

Help us figure out what's going wrong with the clips! We need a better understanding of precisely what kinds of errors the AI is making when trying to generate voice clips.

1. Get a clean clip of a male voice (since these have the most problems).
2. Get a waveglow-synthesized version of the same clip.
3. Try out a bunch of standard audio manipulations of the clean clip.
4. Repeat until we find audio manipulations that sound very close to the waveglow-synthesized version.
5. Post the final set of audio manipulations.

Colab-capable anons: create ngrok links

Video demo - https://u.smutty.horse/lxlnfvvwoac.mp4

These steps explain how to get an ngrok link to Cookie's Multispeaker Colab Notebook. This lets anyone use a Colab server to create audio clips voiced by any of a few hundred characters.

You don't need a fast connection or powerful computer to do this. This uses Google's resources to host a server. Ngrok lets you expose Google's server to the public internet so anyone can access it.

1. Open Cookie's scripts in Colab.
   a. https://colab.research.google.com/drive/1UjSg4tDcubbkax781fE0pNeAFdht_MZ0?usp=sharing
2. Clip the "Copy to Drive" button. This button is tiny and gray, so it's hard to see. Ctrl+F for the text.
3. Follow the instructions in step "1 - Mount Google Drive and add model shortcut"
   a. You may need to click the folder icon on the left. You'll see an option for "Mount Drive" once you do.
4. Run all of the cells one at a time in sequence until you reach step 3. Once you run the cell that ends with "!python3 app.py", you'll get a link to the server. It will take about a minute before the link is active.
5. In the same window, open the Dev Tools Console
   a. In Chrome, the hotkey to open Dev Tools is Ctrl + Shift + J
   b. In the window pane that opens up, select the "Console" tab
6. Copy/paste the following and hit Enter. This will get the Colab instance to stay running for longer.
```
function ClickConnect(){
    document.querySelector("paper-button#ok").click()
}
setInterval(ClickConnect,60000)
```
7. Post the link in the thread so other anons can use it.

For a more detailed guide, see the Inference Server guide.

AI anons: test out audio super-resolution with WaveGlow

We're using WaveGlow to turn Tacotron output (mel spectrograms) into audio samples (PCM). The publicly-available WaveGlow is trained on 22khz audio, and it sounds good for that sampling rate. The problem is that 22khz audio doesn't sound good.

Cookie has tried training a 48khz WaveGlow, but WaveGlow is too difficult to train. One alternative would be to use the public 22khz WaveGlow model and run the results through an audio super-resolution network to increase the sampling rate. We need some anon to try this out to see if it produces good results.

If possible, use a pre-trained audio super-resolution network on WaveGlow's output directly. If that produces passable results, post that to the thread because we want to know about that immediately. Afterward, and if you have time, try to fine-tune the super-resolution network on our

data. Feel free to post in the thread if you need help using our data. If you don't have a good GPU to fine-tune the network, post in the thread and try using Google Colab.

Patient anons: transcribe the dialogue in comics

Comic Anon is trying to extend our transcript dialogue with comic dialogue. The goal is to create a dataset for a future AI chatbot that can speak like your waifu. In conjunction with speech recognition software and our Waifu TTS, this would let you chat with your waifu anywhere.

You can see an example comic transcription here:
- https://drive.google.com/file/d/1b2YnpmqSlCKPNBSMDjrFj51-XmZtxFOl/view

Comic Anon has found this OCR tool helpful:
- http://capture2text.sourceforge.net/

Clipper prefers to use speech to text on his phone to dictate the comics aloud.
- Whichever method you use, always double check the output to make sure it's correct. Automated methods like these have a tendency to introduce strange errors every now and then.

Post in the thread to coordinate your efforts with Comic Anon and make sure you're not doing redundant work.

Devs in training: learn to do machine learning

An anon recommended this online course:
- https://developers.google.com/machine-learning/crash-course/ml-intro

Post your progress in the thread as well as any questions if you have trouble understanding anything. We can use you questions to put together a machine learning FAQ.

Musical data anons: create a chord progression dataset

A bunch of anons are creating singing pony clips. Putting a simple chord progression on top of the clips seems to add a lot to the clips. If we had a dataset of chord progressions played at many tempos, in many styles, with many different instruments, anons could just copy/paste a chord progression over a clip with a little searching.

We don't know how to create such a dataset. If you do, post your ideas in the thread.

Web dev anons: create a *booru for audio data

Clipper has created a dataset of sound effects and background music with his own tags. The intention is to have a searchable repository for sounds that anons can add to their generated clips. The additional background music and sound effects add a lot more depth to the clips.

Derpibooru's tagging system is very good for getting high-quality, searchable tags for images and animation clips. It would be nice if we could have a *booru for audio clips with a similar tagging system.

Note that Danbooru has source code available.
- https://github.com/danbooru/danbooru

If you decide to work on this, please post in the thread.

Patient anons: check phoneme pronunciations

Phoneme pronunciations are good for a number of things.
- We can use it with a phoneme alignment tool to get information about how the speaking rate varies throughout a clip. This gives us speed information that we can feed to our AI models.
- It helps catch transcription errors, like spelling errors or text normalization issues.

We have pronunciations for most of our pony data, thanks in large part to Pronunciation Anon. In some cases and for newer data, we used auto-generated pronunciations. Most, if not all, of our pronunciations were written based on existing dictionary entries without any listening tests. We need to verify our pronunciations to make sure they line up with how ponies actually say these words.

You can find our list of missing words, filled-in pronunciations, and corresponding audio samples here:
- Auto-generated pronunciations only:
https://drive.google.com/file/d/1LpPWxUXQ0rrW_GgTdq-77gnTV0CJN30s/view?usp=sharing
- Auto-generated & manually-entered pronunciations:
https://drive.google.com/file/d/17v7T9aANMAjtroFxLbIyxqwJImkhX1m7/view?usp=sharing

For each of these, you would need to enter the provided pronunciation into ngrok, or any other TTS tool that accepts Arpabet, and listen to the result to make sure the pronunciation lines up with the show audio. Please report any errors in the thread.

Note that we do NOT want to include typos in our pronunciation dictionaries. If you see a typo, please report it in the thread so we can fix it in our dataset. If a word has a stutter, please make sure the phoneme pronunciation reflects the stutter. For example, "I-I-I'M" should have the pronunciation "AY0 AY0 AY1 M" (which becomes {AY0}-{AY0}-{AY1 M} in ngrok) to reflect the fact that the "I" is repeated when spoken.

Anons motivated by the prospect of animation AI

Take a look at our [Animation AI section](). Ask questions, post suggestions, and help out wherever you can.

Animation Anon is working on tools to load available Flash assets. Clipper is going through Derpibooru tags to find ones that would be useful as animation commands. Synthbot is doing whatever task isn't claimed. This is a difficult task and it requires a lot of work, but it seems to be possible for at least some characters.

We need to do a lot of work to get to the point where we can apply animation research to ponies. We just need to get to that point so we're basically guaranteed to get animation AI as researchers continue making progress.

Suggest Sources for Text Data

To make a text generator AI, we need good sources of pony text data. This would primarily consist of transcripts of episodes, Equestria lore from various wiki pages, and generic slice of life type fanfics. Once we have a good pile of text data to work with, we can refine it down to discard any elements we don't want. Since this will start as a baseline model, the most useful elements would be descriptions of the facts of Equestria and natural interactions of the main characters.

Below is a list of suggested sources. If you have any other ideas to suggest, please put them in the thread. Note that just because a link is suggested here, does not necessarily mean that all of it will be used. Each source will need at least some manual vetting to remove unnecessary or undesirable elements.

MLP fandom wiki

Episode transcripts - https://mlp.fandom.com/wiki/Special:BlankPage?blankspecial=transcripts

https://u.smutty.horse/mbsoykdnonq.7z

Characters - https://mlp.fandom.com/wiki/Characters

Locations & Settings - https://mlp.fandom.com/wiki/My_Little_Pony_Friendship_is_Magic

Lore - https://mlp.fandom.com/wiki/Category:Society

Misc - https://mlp.fandom.com/wiki/Cutie_marks

https://mlp.fandom.com/wiki/Friendship_lessons

Wikipedia

https://en.wikipedia.org/wiki/Equestria

https://en.wikipedia.org/wiki/List_of_My_Little_Pony:_Friendship_Is_Magic_characters

TVTropes

https://tvtropes.org/pmwiki/pmwiki.php/Analysis/MyLittlePonyFriendshipIsMagic

https://tvtropes.org/pmwiki/pmwiki.php/JustForFun/Equestria

https://tvtropes.org/pmwiki/pmwiki.php/Recap/MyLittlePonyFriendshipIsMagic

https://tvtropes.org/pmwiki/pmwiki.php/Characters/MyLittlePonyFriendshipIsMagic

Fimfiction

In regards to fanfics, we'll obviously want to keep it as FiM related as possible and keep extreme and niche settings/fetishes etc to a minimum (at least for a "base model"), so the "Slice of Life" Fimfic group is probably the best place to start since that should have fairly generic settings and more emphasis on everyday character interactions. Suggest taking the top hundred or so rated fics from the "Main", "Normal" and "Comedy" segments (watch out for duplicates) of the Slice of Life group for a first pass. Once we have that we can just discard any stories with tags we don't want and refine it further from there.

https://www.fimfiction.net/group/225/folder/11206/main?order=rating
https://www.fimfiction.net/group/225/folder/896/normal?order=rating
https://www.fimfiction.net/group/225/folder/894/comedy?order=rating

The G4 story bible

We only have this image version for now, which could be converted to text with some effort though a plain text version would obviously be more helpful here.

https://u.smutty.horse/mbvvgxeqmoi.7z

Original show bible PDF:

https://u.smutty.horse/lvocxbhiftu.pdf

Voice

What can I do with the AI?

The various AI voice generation tools that have been created by this project are incredibly versatile and have many potential applications. If you're looking for something fun to do with the voices but are currently lacking inspiration, here's a list of ideas:

Skits
- These are essentially verbal oneshot greentext stories, generally short comedic performances where the silly ponies say silly things. Examples include various works by [Snoopy Anon](), [BGM](), [Clipper]() and [other]() [anons](). Here are some [animated]() [examples]().
- If you're an absolute madlad, you could try [re-dubbing a whole episode]() or a [Peanuts Christmas special.]()
- See Clipper's [Master file 2]() for a folder containing various sound effects and background music that can be used to enhance your creations.

Some skit suggestions:
- Trixie writes a book she wrote about herself and her life called "The Hardships of Being Unequivocally Superior".
- Your pony of choice lists off the undesirable traits of other ponies to woo Anon by showing that they are logically the superior mate.
- Trixie gets sick of bills and tries to get Twilight to marry her for the sweet princess money.
- AJ speculates about starting a family with you.
- Ponies swooning over Anon, listing off traits like gamer, lives in a basement, watches anime, doesn't bathe, is overweight etc. like they're traits of an ideal man.
- Anon is being chased by an angry Dash, could be interesting to play with 360 audio positioning.
- Twilight builds a robotic assistant modeled on herself to help with problem solving, but robo-twi inherits her neuroticism, which feeds mutual freakouts.

Comics
- If you want to do a skit, but are finding it hard to write original material, you can try dubbing a comic instead. Grab a comic from one of the boorus, run the lines through the AI, add sound effects and music if you want, and then use a video editor like [Shortcut]() to piece it all together.
- [The Pony Hackers]() is a good example of this.

Voiced fanfics/greentext

- If you want to go for something bigger than comics, there is a near infinite selection of classic greentext stories and fanfics that would be wonderful to listen to as audiobooks.
- These are generally a larger commitment, but are also easy to produce since everything's already transcribed for you. All you need to do is feed it into the AI and edit it together in Audacity or similar. The level of complexity with these projects is usually defined by how autistic you are about finding the best sound effects and music.
- Clipper's dub of [Insults Are Magic](#) is a good example of what's possible with AI voices combined with sound effects and background music.
- While you're at it, you could also label the story with [Synthbot's fanfic labeller](#) to help improve the fanfic dataset.

Singing/rap/poetry

- Previous attempts at "singing" were mostly just anons trying to force the AI to speak lines in time with a beat, but with the recent addition of [SortAnon's TalkNet](#), we can now create singing and rapping lines based on reference clips fed to the AI as well as the input text.
- [Here](#) [are](#) [some](#) [examples](#) [of](#) [what](#) anons have made with this new capability. [WoodenToaster - Rainbow Factory (BGM Remix) (ft. TalkNet)](#)

Professions of love

- Self explanatory. Do it for her.
- [Here for you](#), [Twilight's voicemail](#)

Suggested workflows:

These propositions may be equally daunting as they are inspiring, and it may be difficult to work out where to start. Here are some example workflows written by those who've created various AI voice projects in the past to help you get started.

- Clipper, [Insults Are Magic](#).
    - Find original greentext in desuarchive, and from there find a screencap and pastebin.
    - (Optional) Label text from pastebin with [Synthbot's fanfic labeller.](#)
    - Crop screencap into smaller manageable pieces that would easily fit on a typical screen.
    - Generate lines using the various voice generation tools available, usually three times per line to give multiple options. Lines are typically organised by character and named 1a, 1b, 1c, 2a, 2b, 2c, 3a etc. The numbers refer to the spoken lines in chronological order and a, b and c are the duplicates of each line that were generated to allow for multiple options per line.

- (Optional) Record voiceover for narration and lines spoken by Anon. I chose to use my own voice, but you could use another AI if you prefer.
- Go through each line and pick the best variant for each one, can also combine elements of multiple takes if some options perform better in some areas but worse in others.
- Arrange refined lines in chronological order in Audacity, no particular attention paid to timing at this stage.
- (Optional) Add sound effects and background music. I used a separate audio track for dialogue, music and sfx, and also a few extra tracks for when I wanted to layer multiple sound effects at once.
- Adjust the timing of each line and sound effect, also re-generate any lines if needed.
- Export finished audio file, and import into chosen video editor.
- Rig up a simple slideshow with the screencap of the greentext, and add end credits.
- (Optional) Edit the credits.
- Export video file and upload.

- Anon, Pony Hackers and various other comics
    - Sample projects (Mega)
    - Find a suitable comic
        - If you already have a comic in mind, then you're already good to go.
        - You'll probably want to take into account the amount of time it will take to complete a comic, make sure it's not going to take more time than you're willing to give it.
        - Comics with more pages will obviously take more time, but also take into account the amount of text per page/panel.
        - Dialog with a lot of emotion and/or non-standard pronunciation will take more time to fine tune.
        - Look to see what characters are featured in the comic and that suitable models are available for them. For background ponies/OCs, try to have a plan ahead of time. Like applying sound effects to another character's voice.
        - If you want to do a video, look at the formatting and consider how you'd want to present it. Speech bubbles are probably easiest to deal with.
    - Extract lines from comic
        - Go through the comic and type out every speech segment into a new line on a text editor.
        - It can also be helpful to make notes on emotion, pronunciation, or other aspects of line deliveries.
        - Try to include enough information so that you don't have to reference back to the comic. In the middle of generating lines it can be easy to forget little details.

- Generate lines
    - Have a plan in place for file organization. My organization is typically p[panel].[line].wav with each page having it's own folder. For example, under this scheme page 1 panel 3 line 5 would look like: "page 1\p3.5.wav".
    - Reference "Making the Most of the AI" section for tips on improving your lines.
    - Context at the beginning of a line will have a greater effect on the delivery than context that comes after. I typically use context at the beginning for big adjustments and context after for fine tuning.
    - Punctuation has an effect and can sometimes result in strange things. For example, for a shouting line it may work better using context and a period instead of an exclamation mark.
    - If you want to keep the tone of the character constant, using the same context in front of various lines will go a long way to help with this.
    - Generate several versions of the same input and select which one sounds best. If it isn't exactly what you want, mess around with different inputs to see what other results can be achieved.
- Assembling lines
    - Decide whether you want to assemble audio in an audio editor or in a video editor. If you intend on making a video out of the comic later, assembling the lines directly in the video editor will give you more flexibility later. I will say that assembling the audio in a dedicated audio editor (like Audacity) first is a bit easier.
    - I'd recommend first assembling the lines on their own and focus on trying to get things sounding natural.
    - Consider where you'll apply sound effects later and give some breaks in the audio where these will happen. Don't worry too much about the duration of breaks at this point, mainly use them as a marker for later when applying sound effects.
    - Use separate tracks for different characters. This makes it easy to keep track of which audio belongs to who.
    - You can also consider panning the audio tracks to give an effect like characters being in different locations.
- Finding sound effects and background noise
    - The YouTube audio library is a good resource for professional sound effects and music. All you need is a Google account to access. Take note if you need to credit anyone when you use audio from here.
    - Most of my background audio comes from sound effects uploaded on YouTube itself. Note that this will not result in the highest quality of audio, but this is honestly the best way to find what you're looking for.
    - Keep background noise and music on a separate track from sound effects.

- Most sound editors have the ability to generate different types of noise. For example, Audacity can generate white, pink, and brownian noise. This can help make it seem like audio was recorded on an actual microphone. Just keep it subtle if you do use it.
- Creating video
  - I usually use an image editor to prepare all of my frames ahead of time before starting in a video editor.
  - For each panel you'll want to crop the image down to just the panel itself (if applicable).
  - Make a second layer on top of the panel and use a brush tool to paint over the dialog. You can then erase from the second layer bit by bit to reveal the speech as needed. Save an image for each line. Be sure to include a blank with no dialog as it can help make video transitions easier.
  - I would recommend using a similar naming scheme to the audio files. Whatever you do use, make sure that everything works out to be in alphanumerical order so when you go to import into the video editor everything is in order.
  - Make sure to save your project files for each panel, typically just the panel with the fully blanked second layer. This makes it easier if you need to go back and make any changes.
  - Once you've finished making your frames, import all the audio and images into your video editor.
  - If your video editor supports showing the audio levels on the timeline, you can use this to help position when the frames appear. I usually make the next frame appear slightly before the character says the line.
  - For video transitions I usually keep it simple. Each frame, the video just cuts to the next frame. I generally use a fade between panels and a dip to black between pages. This is just personal preference and only really suitable to comics that are structured as panels.
- Tips on posting
  - For posting on 4chan, smutty.horse is a personal favorite of mine to use as host.
  - Formatting as webm makes it a lot easier to upload to various places.
  - [This](#) is a simple tool to make webms. Be sure to check "Enable Audio" at the bottom. Note that this tool is designed to produce webms for 4chan and will target a 4mb file size. Encode options are limited.
  - As of late i've been using ffmpeg directly to try and get the most out of a limited file size. I was targeting a file size under 25mb. The following is the options/command I use. Some notes: vp9 is the video codec used, opus is the audio codec used, audio is limited to 16 bit, audio bitrate is 120k, video is set to use a CRF of 35, video will be converted to 30Hz, video is scaled to 720p, threads is set to 16 (set this to the number of threads your CPU has, 16 is just what I have).

```
ffmpeg -i input.mp4 -c:v libvpx-vp9 -c:a libopus -sample_fmt s16 -b:a 120k -vbr on -crf 35 -b:v 0
-vsync 2 -r 30 -vf scale=1280:720 -thread 16 output.webm
```

Tutorials

Creating an audio dataset

This is a guide on how to create a dataset of raw voice clips and text transcripts for any given source of audio. The output of this process can then be used to create a dataset for training in Google Colab and/or submitted to 15 (fifteenai15@gmail.com) for potential use at https://fifteen.ai/

Note that our system uses a file naming system different to that specified for contributions on 15.ai, however 15 is able to use the pony dataset without issue so it should be fine to submit your datasets to him using the same format we do. Just make sure you follow the format closely and note in your submission that you used the same system as us. Alternatively, you could adapt the PPP method to use a different naming system if you prefer. Please also share any datasets you create in the thread, more data is always helpful.

Before you start any work, please post in the current PPP thread about what you plan to do so other Anons know what's being worked on.

See also the section on cleaning audio if your audio source contains undesirable elements such as sound effects that obstruct the voice.

To submit your contributions, please follow the guidelines for submitting your content when you are done.

See also the "Automatic Clipping and Transcribing" section for an alternate method.

If you have any questions about the process, ask Clipper in the thread.

Overview

The full process is demonstrated by Clipper in this YouTube video.

The goal of this process is to take the full cut of an audio source and slice it into its individual lines, which will all be tagged with data such as the character speaking, the emotion with which the line is spoken and a full transcript of all the words spoken in each line.

Here is an overview of the core steps of the dataset creation process:

1. Obtain audio for your chosen character
2. Obtain subtitle .srt file and transcript, if possible.
3. Download Audacity, Notepad++, and Python.
4. Use the subtitle to Audacity app, if you were able to get a subtitle file.
5. Run the character tagger program, if you were able to get a compatible transcript.
6. Import the audio files into Audacity, and then create and edit labels as needed.
7. Export labels from Audacity and run them through the checking script, correcting any errors it finds.
8. Re-import the corrected labels into Audacity and export the audio clips.
9. Create a text transcript of all the newly created audio clips.
10. Upload all the files you've created.

We will now explain the process of each of these steps in more detail.

Obtaining Audio

There are several potential sources of audio, and some sources will be better than others. Here are some suggested sources, in order of most ideal to least ideal:

- Raw studio recordings
- PC game files
- Audiobooks and podcasts
- Netflix and iTunes videos
- YouTube videos

Raw studio recordings are often the hardest to obtain but will offer perfectly clean studio quality voices which are the best source of any audio. We were lucky enough to get some studio recordings for various MLP episodes from BigHckintosh as part of the Hasbro studio leaks.

The sound files in PC game files are often similar in quality to raw studio recordings and are easier to obtain. If you're training for a character or voice actor that appears in a video game, go digging through the files and see what you can find. You can also consult forums dedicated to the particular game if finding the specific files proves difficult.

Audiobooks and podcasts are also usually recorded in proper sound studios, but will also tend to have undesirable elements such as music and sound effects mixed in. These are often easily accessible on their respective host websites, and you can even download podcasts directly from iTunes.

Shows hosted on Netflix and iTunes are usually in high quality, but will almost certainly contain sound effects and background music which will obstruct the voices. This can be mitigated to an

extent, but it's never going to be as good as the raw recordings. For iTunes, you can simply download videos directly. To download videos from Netflix, you will need to use [Flixgrab](). For both Netflix and iTunes, make sure that the video you download has 5.1 audio, a format which should allow you to isolate and remove background music. You can usually find audio information like this in the detailed video descriptions. Note that obtaining 5.1 rips of audio in multiple languages will also be useful for removing background noise.

A YouTube video can be used if there is no material available on Netflix or iTunes, but is a much less ideal source as compression will compromise audio quality. The audio will also only be available in stereo, which means you won't be able to remove any music. Only use YouTube if there is literally no other viable option. Search for "youtube downloader" in Google for various online solutions to directly download videos from YouTube.

As a general rule, you should aim to obtain at least thirty minutes of audio for your chosen character before attempting training. This is the minimum amount of data you need to guarantee a reasonable chance of getting a decent output from your model. As with all applications of artificial intelligence, more good quality data will result in a more accurate model, so you should always aim to gather as much data for your chosen character as you can.

Obtaining Subtitles and Transcripts

Subtitles contain information on the content of speech and the time it occurs, which we can use to create Audacity labels to give us a head start. It may be possible to rip these files from a Blu-Ray disk, or they may be found online [here](). You can also use [this programme]() to download subtitles from YouTube videos, make sure you download in .srt format.

We have also created a tool to automatically tag characters to their lines with the help of a transcript. For this tool to work, you will need a transcript in the same format [as shown here.]() You may be able to find a transcript like this for your show at [https://www.fandom.com/](). Even if you can't find a transcript in this exact format, it can still be used as a source to copy-paste from to help with transcribing lines later on.

If you can't find .srt files and/or a transcript for your audio, don't worry too much. This step is optional, though having these files will make your life much easier later on so it is strongly recommended to make every reasonable effort to find them.

Acquiring Software

Audacity is a free open-source audio editor, we will use it to do the work of slicing the audio into its individual lines. Download Audacity [here.]()

Notepad++ is an enhanced version of notepad for editing text files. We will demonstrate some of its useful features later. Download Notepad++ here.

Python will allow us to run scripts to automate some steps for editing Audacity labels later. Don't worry if you don't have any expertise in coding, we won't be doing anything complicated and all the steps are fully demonstrated in the video. Download Python here.

Simply download and install these in the same way you would for any other programme.

If you are using a Netflix/iTunes/YouTube video as your audio source, you will need to isolate the audio from the video. The easiest way to do this is to install the FFMPeg plugin for Audacity, which will allow you to import video files just like you would for an audio file. Instructions for the Audacity plugin can be found here.

Subtitle to Audacity Tool

If you weren't able to get a subtitle .srt file, you can skip this step and also the Character Tagger step. Click here to skip this section.

Open the .srt to Audacity app here.
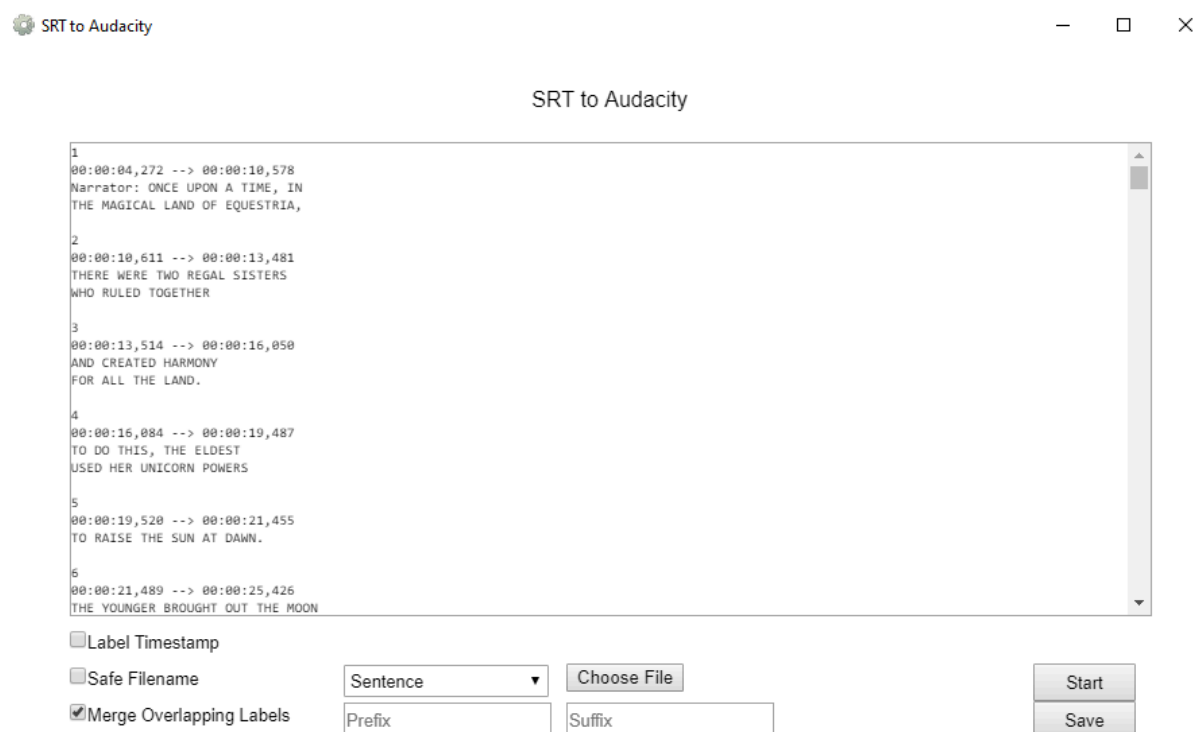
This is the subtitle to Audacity converter. It will generate Audacity labels with the information in the subtitle file, which will give us a head start. To use it, simply open the subtitle file in the app,

then click merge overlapping labels. This will merge any subtitles with overlapping timestamps into a single label. I recommend you leave the other two options blank. We will automatically generate timestamps later, and the safe filename option will remove question marks, which you will have to retype later.

The drop-down option will allow you to choose a formatting option for the output. You can use whichever formatting you want. Run the app, and save the output in a useful place.

If you were able to find a suitable transcript for your audio for use with the character tagger, go to the next section now. If not, keep reading.

Open the output of the subtitle to Audacity app in Notepad++ and use a macro to add three underscores to the start of every label. The use of macros in Notepad++ is demonstrated in the video here. Once done, follow this demonstration in the video to make some minor edits to the labels before importing them into Audacity.

Character Tagger

If you weren't able to get a transcript in the format shown earlier, you can skip this step. Click here to skip this section.

This is the Character Tagger. It will attempt to match every spoken line to a character by comparing the lines to a transcript. Open the output from the subtitle to audacity app from the

previous step in the middle box, and then copy and paste the transcript in the left box. In the suffix box, type in three underscores, and then click label unknown characters. Exactly why we do this will become clear in the next step. Run the programme and save the output in a useful place.

Open the output of the Character Tagger in Notepad++ and follow this demonstration in the video to make some minor edits to the labels before importing them into Audacity.

Clipping Audio in Audacity

This process is demonstrated in full in the video. I suggest you use the video as your primary source of information here as I feel that a live demo explains the process better than can be done with simple text and screenshots.

Open the dialogue audio file in Audacity. Once the audio has finished loading select "File" ->" Import" -> "Labels" in the menu. Choose the label file that we just created in the previous step. If you skipped the subtitle to Audacity step, you won't have any labels to import and will instead need to create them manually. Create a label track by selecting "Tracks" -> "Add new" -> "Label Track". See this section of the demo video for a guide on creating labels manually.
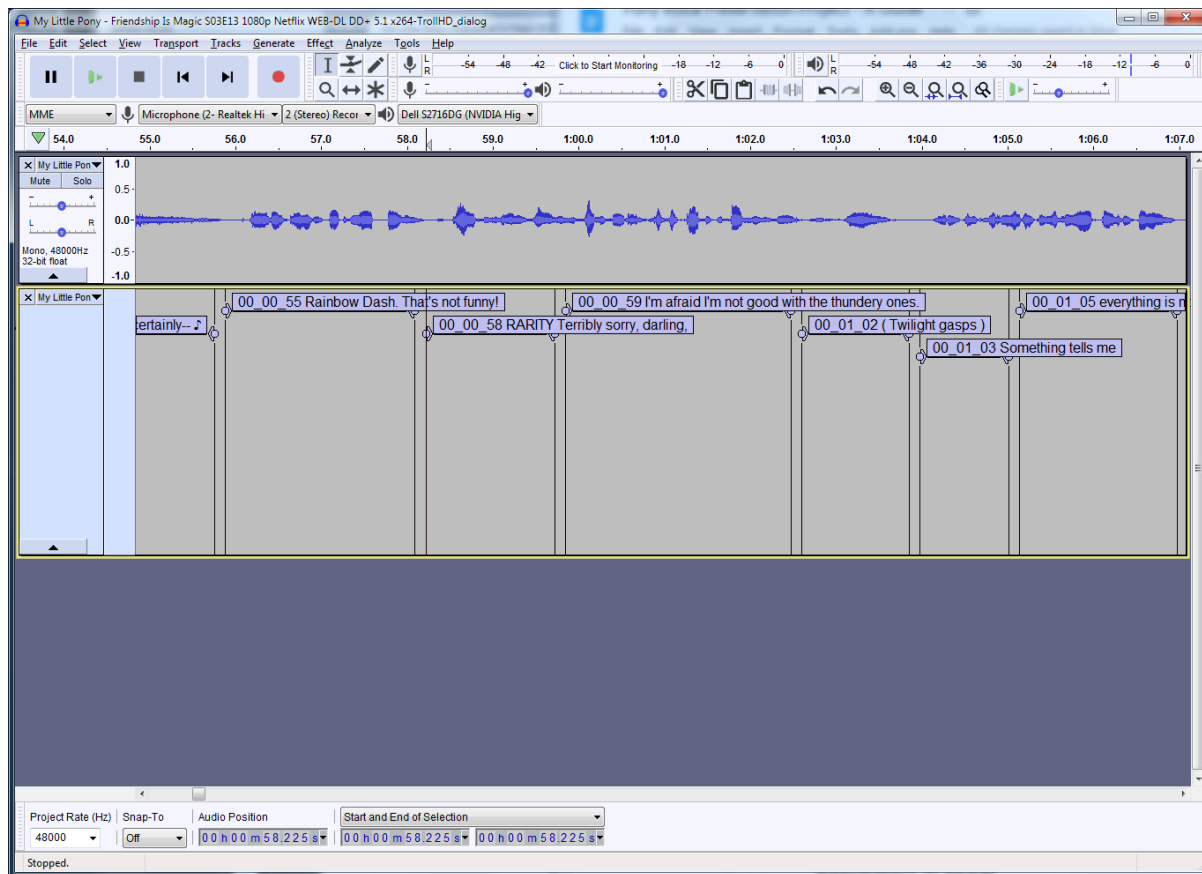
The idea here is to draw labels around all the individual lines of dialogue, and then fill those labels with the information we need. The start and end points of each label will mark the start and end point of each clip. The black vertical bars in the label track represent label boundaries, beginning and end. Clicking and dragging on either will allow you to start a selection from that point. The circular buttons on either end of the label text allow you to move the entire label in time, and the triangles attached to the circles allow you to move the beginning or end individually. You can create new labels if needed by highlighting a section of audio and pressing "Ctrl+B".

Use this to create and adjust labels as needed. You may combine labels as needed if the audio doesn't split well on the existing bounds between two labels, though there's no automatic way of doing this. Simply right click on and delete the second label and extend the first one to compensate.

Audio clips should contain about a sentence worth of dialogue, but it is most important that your clips are split in a way that sounds natural. Do not let clips start part way into a word or let them end in the middle of one. Try to clip such that the emotion conveyed by the tone of voice is consistent throughout. Audio clips should also contain entirely noisy or entirely clean audio, so take this into account as well when deciding where to put your clip boundaries. When filling in

the labels, you must make sure to include all the necessary information in the format shown below:

PONY PRESERVATION PROJECT

– VOICE –

So, you want to help? Great.

Phase 2 is using the high quality audio track from the episodes and films to extract all the chucks of dialogue and speech and link it to its exact transcript. For each episode / film, the end product is a shitload of audio flac files (the dialogue / speech itself) and an audacity label file (which contains all the exact transcript and very precise start and end time of said sample in the original audio).

By using a precise notation, each audio chunk can then be used for machine learning (or everything else if you will).

Tl ; Dr : Name your files like this fags :

HH_MM_SS_Character_Emotion_Noisy_Dialogue

Timestamp
In hour, minutes, seconds.
Duplicate timestamps don't matter, but should be avoided if possible.
Valid chars: [0-9]

Character name
Shorten the name if reasonable to do so, e.g. Twilight Sparkle -> **Twilight**. No pet names or nicknames. 'Multiple' can be used if appropriate.
Valid chars: [a-z], [A-Z], space

Emotion
Modifier on how the sentence is said, may not always be strictly emotion, such as 'whispering'.
If multiple emotion tags are appropriate, separate them with a space. Use **what you want**, but see the **suggestion list** below.
Valid chars: [a-z], [A-Z], space

Noise level
Indicate if the clip is noisy
Noisy clips are tagged with '**Noisy**' if it's easily recoverable, e.g. hoof steps.
Very noisy clips are tagged with '**Very noisy**' if the clip requires significant processing to recover, e.g. loud sound effects like train sounds or stampedes.
Clean clips are tagged with **nothing**, make sure underscores that would have separated the tag are still included.
Valid chars: [a-z], [A-Z], space

Dialogue
Dialogue that is spoken in each voice clip. Must be a word for word transcription of **EVERY** word that is spoken, **exactly as spoken**, with correct spelling and punctuation. This is necessary for the script that auto-generates the transcript text files to work correctly. Shortened versions of words should be **typed out** into **full** versions (e.g. Mr. -> Mister, 2 -> Two).
Valid characters: [a-z], [A-Z], space, ['.,!?-], ASCII accent marks.
['] is used as an apostrophe, not inline quotes
[,] indicates all pauses, including ones more accurately indicated with other symbols such as [: ; ...]

>Everything is case insensitive, upper case is used for human readability only.
>Global allowed characters are the following ASCII: [a-z], [A-Z], [0-9], [ ], [_-,.!?'], accent marks.
>Disallowed characters: "#$%&()*+/:;<+>@[]\^{}|~, ASCII control characters, DEL, All extended ASCII codes except accent marks. Underscores only for separation (and question mark, see below)
>[?] is replaced in file names with _, this is done automatically by Audacity in the export phase. This is unambiguous as we know that every underscore after the sixth one should be a question mark.
>The text file containing the raw Audacity labels must be included in all submissions as '**labels.txt**'.
>Each episode will have its clips in its own '**SXXEXX**' folder, so we know where each line came from. This will allow easy addition of extra identifiers in file names later by script if desired.

Timestamp - Ignore the timestamp while filling in the labels in Audacity, we will automatically generate it later.

Character - You can use abbreviations for tagging characters while clipping your audio to save time, for example "twi" for "Twilight". You can use whatever abbreviations you want, just make sure you keep a note of what you have used for future reference, you will need to enter the abbreviations you've used into the checking script in the next section.

Emotion - The suggested list to use is: Neutral (n), Happy (h), Amused (am), Sad (s), Annoyed (a), Angry (ag), Disgust (d), Sarcastic (sa), Smug (sm), Fear (f), Anxious (ax), Confused (c), Surprised (su), Tired (t), Whispering (w), Shouting (sh), Whining (wh) and Crazy (cr). It is suggested to use the one or two letter abbreviations given in the brackets for quicker and easier tagging, these abbreviations will be expanded into their full versions with the use of the checking script in the next section. If appropriate, you can use multiple emotion tags in a clip, such as "Happy Shouting" or "Angry Whispering". In these cases, make sure that the tags are separated by a space. You can also invent and use other emotion tags if you feel that none of the emotions listed above fit your audio. If you do this, make sure to keep a note of what you have used for future reference, you will also need to enter the abbreviations you've used into the checking script in the next section.

Noise - This is a difficult concept to explain as it's always going to come down to a judgement call. It is important to get this right, so make sure you put on a decent pair of headphones and listen carefully. If a clip is clean, that is free from all but the most trivial of noise, then leave the noise tag blank. Make sure that the underscores are still included. If there is significant noise in a clip, then you will need to make a judgement call. Marking it as noisy is effectively you saying that you would be happy to use that clip for training, despite the small amount of noise it still has. Marking the clip as very noisy is effectively you saying that the clip is unsuitable for training due to excessive noise. Generally speaking, for a clip to qualify as noisy, you should be able to clearly make out all syllables of every spoken word, and the noise itself should be quieter than the speech. It's usually best to be strict for the sake of preserving quality, so If you find yourself in doubt between clean and noisy, tag it as noisy. Similarly, if you find yourself in doubt between noisy and very noisy, tag it as very noisy. Use the shorthand "q" for noisy and "qq" for very noisy.

Dialogue - The transcript must contain every word that is spoken in each clip, exactly as spoken, with correct spelling and punctuation. Remember that all pauses should be represented by a comma or full stop, whichever fits best. All sentences should end with either a full stop, a question mark, or an exclamation mark. A sentence should not end with a comma. Also be careful not to make the clip too long, as everything you enter into the label will later become the filename for that clip. Windows imposes a hard cap of 260 characters for a filename, including the directory. As a general rule, each label should contain about one sentence of speech. The length should be at least one second, and no more than ten seconds. You can combine and split labels however you feel is best.

Keep in mind to not make the label of the clips too long, as what you enter into the label will later become the filename of that clip. Unfortunately Windows, being a shit OS, has a hard file name limit of 260 characters. Note that the character limit includes the file directory, so if you find that a filename has become too long to edit, try moving it out of folders/subfolders and into just the "Documents" section in the file explorer. This will shorten the directory path and allow you to edit filenames that would otherwise exceed the character limit.

Example:
C:\Users\Anon\Documents\PVPP\Sliced Dialogue\FiM\S1\s1e1\00_00_05_Celestia_Neutral__Once upon a time, in the magical land of Equestria..flac (135 characters)

C:\Users\Anon\Documents\00_00_05_Celestia_Neutral__Once upon a time, in the magical land of Equestria..flac (102 characters)

After editing the filename, you can then move it back into the original folder/subfolder, even if it exceeds the character limit.

I'll recommend again that you watch the section in the demo video that covers this section, as a live demo will communicate the process better.

Checking Script

We have now made the labels in Audacity with the character, emotion and noise tags and transcript. The next step is to make sure that there are no mistakes and add the timestamp to the start of each label. Refer back to the file naming system for details on the format of the timestamp.

Before following any of the instructions below, make sure you have exported the Audacity labels. File -> Export -> Export labels. Name the file "labelsraw.txt".

The checking script is available here. It will do the following:

- Replace shorthand character, emotion and noise tags with their full versions.
- Add a timestamp to the start of each label in the correct format.
- Check for some common abbreviations in the dialogue that should be expanded into their full versions, such as Mr. for mister.
- Check for any numbers in the dialogue that should be replaced with the word version.
- Adds a full stop to the end of any label that does not end with any sort of punctuation.

I will write out the full process for using the script, but I highly recommend watching the demonstration in the video. I feel it does a much better job of explaining than can be done with just text and screenshots.

To use the script, copy and paste it into a new Notepad++ file, and save it with a .py extension. This will save it as a Python executable. You can name the script whatever you want, the extension is the only thing that matters here. Create a new folder for the script and save it there, making sure there is nothing else in that folder.

From lines 158 to 398, you can see the stored information for shorthand character codes. You can add, remove and change any entries as needed, just make sure it's all in the same format as seen in the script, this is demonstrated in the video. The script will check for any abbreviations in the character field, and replace any it finds with its corresponding full version. It can also check for longer character tags and replace them with a shortened version.

From lines 400 to 423, you will see the entries for emotion. From lines 425 and 428, you will see the entries for noise. From lines 430 to 433, you will see the entries for abbreviations. These all work in the exact same way as for the characters, and you can also add to and edit these however you like, just make sure the formatting is kept consistent.

We now need to do just one more thing before we're ready to run the script. We need to tell the system to run the script in the console window so we can see its output as it goes through the labels. Create a new file in notepad++, and type the following:

@echo off
python filename.py
pause

Replace "filename" with whatever you named the checking script. Save this file as "Run.bat" in the same folder as the checking script. This process is also demonstrated in the video.

We are now ready to run the checking script. Copy the Audacity label file into the folder containing the checking script, and name it "labelsraw.txt". Double click run.

You will see in the output "labels.txt" folder that the script has added all the timestamps to the start of each label, and expanded all shorthand tags to their full versions. The demo video contains examples of how the checking script alerts you to any errors.

Once you have fixed all the errors the script has found, run it again on the corrected labelsraw.txt file to make sure nothing was missed. Once done, you can discard the labelsraw.txt file. We now need to do one last check on the output label file. Use find and replace to correct any cases of double spacing, and make sure there are no spaces before or after any of the underscores. Then copy and paste the labels into a word processor, such as

Microsoft Word or a Google document and run a spell-check. Fix any spelling errors it finds and copy the corrected labels back into the Notepad++ file. Save the file containing the completed labels.

Exporting Audio Clips

Import the new labels into your Audacity project and delete the old ones. We are now ready to export the audio to make the individual clips. It is recommended to disable "Show Metadata Tags editor before export" in the "Edit" -> "Preferences" -> "Import/Export" menu to prevent a window popping up for every single label in your project.
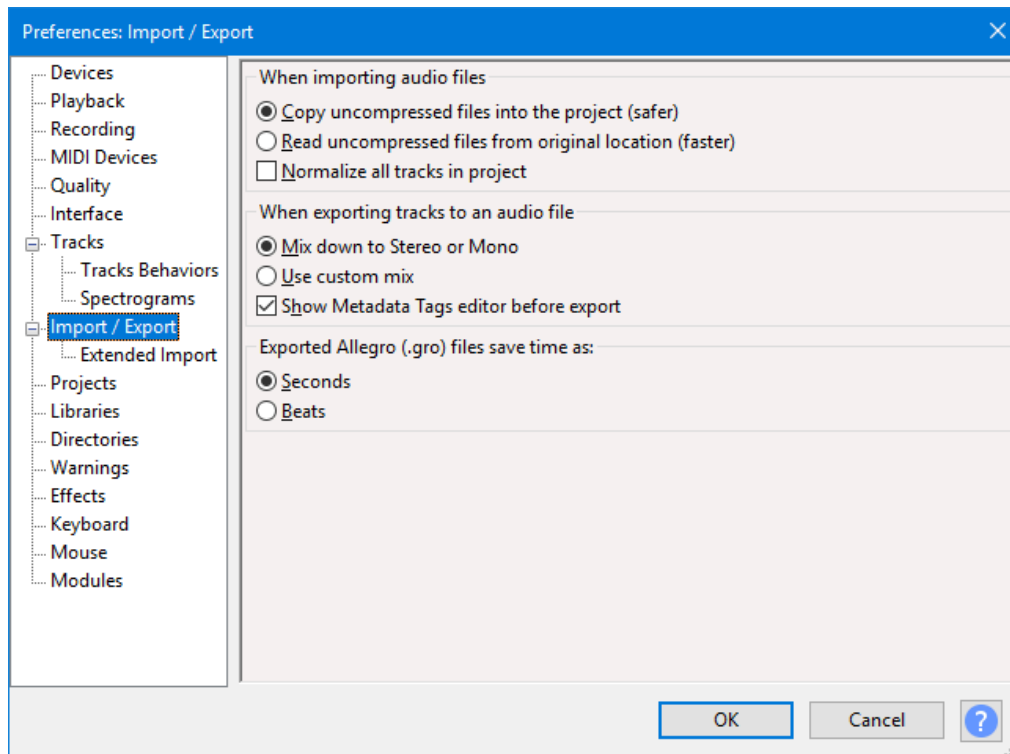
Now go to "Export -> Export Multiple". Make sure the settings match what you see in the screenshot below, and then select the folder you want to export the clips to. Once done, click export.
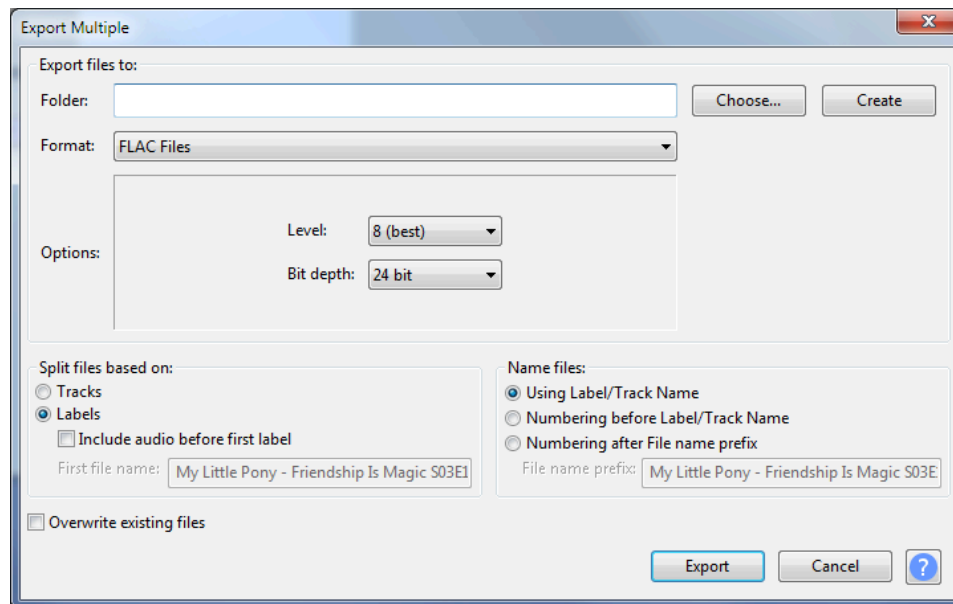
Question marks are not allowed in filenames, so Audacity will automatically ask you if you want to change them to an underscore. Press the enter key to confirm the change, along with every other label this applies to. Do not spam or hold down the enter key, as this occasionally creates duplicate files.

You may run into an error while exporting that interrupts the process. The most common cause of this is that the filename is too long, remember the 260 character limit for a filename, including the directory. If this happens, simply go to the label that caused the problem, and split it into two shorter ones. If you make any edits to the labels like this, make sure you re-export the updated label file.

The output should look something like this, with all the clips created with a filename according to the label we created in Audacity.

Name	Date modified	Type ^	Size
00_00_05_Celestia_Neutral__Once upon a ...	02/01/2020 14:41	FLAC File	120 KB
00_00_06_Celestia_Neutral__In the magic...	02/01/2020 14:41	FLAC File	120 KB
00_00_08_Celestia_Neutral__Of Equestria.	02/01/2020 14:41	FLAC File	85 KB
00_00_11_Celestia_Neutral__There were t...	02/01/2020 14:41	FLAC File	241 KB
00_00_14_Celestia_Neutral__And created ...	02/01/2020 14:41	FLAC File	183 KB
00_00_16_Celestia_Neutral__To do this.	02/01/2020 14:41	FLAC File	68 KB
00_00_17_Celestia_Neutral__The eldest.	02/01/2020 14:41	FLAC File	63 KB
00_00_18_Celestia_Neutral__Used her uni...	02/01/2020 14:41	FLAC File	276 KB
00_00_22_Celestia_Neutral__The younger.	02/01/2020 14:41	FLAC File	58 KB
00_00_23_Celestia_Neutral__Brought out t...	02/01/2020 14:41	FLAC File	174 KB
00_00_25_Celestia_Neutral__Thus, the two...	02/01/2020 14:41	FLAC File	370 KB
00_00_30_Celestia_Neutral__All the differe...	02/01/2020 14:41	FLAC File	159 KB
00_00_32_Celestia_Neutral__But as time w...	02/01/2020 14:41	FLAC File	98 KB
00_00_33_Celestia_Neutral_Noisy_The yo...	02/01/2020 14:41	FLAC File	180 KB
00_00_36_Celestia_Happy__The ponies rel...	02/01/2020 14:41	FLAC File	329 KB
00_00_40_Celestia_Neutral__But shunned ...	02/01/2020 14:41	FLAC File	229 KB
00_00_44_Celestia_Neutral__One fateful d...	02/01/2020 14:41	FLAC File	89 KB
00_00_45_Celestia_Neutral__The younger ...	02/01/2020 14:41	FLAC File	310 KB

Text Transcriptions

We now need to create a text file transcription for all of these audio clips. We need a text file for every single audio file that contains all the words spoken in each clip, and also has the exact same filename as the sound clip it is associated with. You can imagine that doing this manually for several hundred or even thousand clips would be a very long and tedious task, so we have created a script that will do this automatically. The use of this script is very similar to what we did with the checking script earlier and is demonstrated in the video.

The script is available here. This script will read the dialogue section of Audacity labels and create a text transcript containing that dialogue. This is why it's important that the dialogue section of every Audacity label is accurate to the spoken dialogue, as any errors in the labels will be replicated in the generated transcript. The script will then save the transcript as a text file with the exact same name as the sound clip it's associated with.

To use the script, copy and paste it into a new Notepad++ file. Just like we did with the checking script, save this new file with a .py extension to save it as a Python executable. You can name the script whatever you want, just make sure you remember the .py extension. Save the script in its own folder, making sure there is nothing else in the folder. Now create a "Run.bat" file in the same way we did with the checking script, and save it in the same folder.

Copy and paste the text file containing the Audacity labels into the folder with the script, and make sure it is named "labels.txt". Double click run.

The script will have created a text file transcription of every sound clip and saved it with the exact same name as the clip it's associated with.

Name

LabelGen
00_00_05_Celestia_Neutral__Once upon a time
00_00_06_Celestia_Neutral__In the magical land
00_00_08_Celestia_Neutral__Of Equestria
00_00_11_Celestia_Neutral__There were two regal sisters who ruled together
00_00_14_Celestia_Neutral__And created harmony for all the land
00_00_16_Celestia_Neutral__To do this
00_00_17_Celestia_Neutral__The eldest
00_00_18_Celestia_Neutral__Used her unicorn powers to raise the sun at dawn
00_00_22_Celestia_Neutral__The younger
00_00_23_Celestia_Neutral__Brought out the moon to begin the night
00_00_25_Celestia_Neutral__Thus, the two sisters maintained balance for their kingdom and their subjects
00_00_30_Celestia_Neutral__All the different types of ponies
00_00_32_Celestia_Neutral__But as time went on
00_00_33_Celestia_Neutral_Noisy_The younger sister became resentful
00_00_36_Celestia_Happy__The ponies relished and played in the day her elder sister brought forth!

Now cut and paste the text file transcripts into the same folder that contains the sound clips. The final result should look something like this:

Name	Date modified	Type	Size
00_00_05_Celestia_Neutral__Once upon a ...	02/01/2020 14:41	FLAC File	120 KB
00_00_05_Celestia_Neutral__Once upon a ...	02/01/2020 14:47	Text Document	1 KB
00_00_06_Celestia_Neutral__In the magic...	02/01/2020 14:41	FLAC File	120 KB
00_00_06_Celestia_Neutral__In the magic...	02/01/2020 14:47	Text Document	1 KB
00_00_08_Celestia_Neutral__Of Equestria.	02/01/2020 14:41	FLAC File	85 KB
00_00_08_Celestia_Neutral__Of Equestria	02/01/2020 14:47	Text Document	1 KB
00_00_11_Celestia_Neutral__There were t...	02/01/2020 14:41	FLAC File	241 KB
00_00_11_Celestia_Neutral__There were t...	02/01/2020 14:47	Text Document	1 KB
00_00_14_Celestia_Neutral__And created ...	02/01/2020 14:41	FLAC File	183 KB
00_00_14_Celestia_Neutral__And created ...	02/01/2020 14:47	Text Document	1 KB
00_00_16_Celestia_Neutral__To do this.	02/01/2020 14:41	FLAC File	68 KB
00_00_16_Celestia_Neutral__To do this	02/01/2020 14:47	Text Document	1 KB
00_00_17_Celestia_Neutral__The eldest.	02/01/2020 14:41	FLAC File	63 KB
00_00_17_Celestia_Neutral__The eldest	02/01/2020 14:47	Text Document	1 KB
00_00_18_Celestia_Neutral__Used her uni...	02/01/2020 14:41	FLAC File	276 KB
00_00_18_Celestia_Neutral__Used her uni...	02/01/2020 14:47	Text Document	1 KB
00_00_22_Celestia_Neutral__The younger.	02/01/2020 14:41	FLAC File	58 KB
00_00_22_Celestia_Neutral__The younger	02/01/2020 14:47	Text Document	1 KB

Submitting Work

The hard work is now done, all that's left is to submit your content. As shown earlier, there are several file host options available. It doesn't really matter where you upload your clips, so long as they are easily accessible.

Compress all the sound clips, text transcripts **and the text file containing the Audacity labels** into a zip file, upload to your file host of choice and then post a download link in the current PPP thread. Some recommended file hosts are Smutty.horse, Mega.nz, and catbox.moe.

Please do share any datasets you create with us over on /mlp/, we're always looking for more data to work with to improve our models and processes, especially with characters that have unique speech quirks and a wide range of emotions. You can also try emailing the download link to 15 (fifteenai15@gmail.com) for use on 15.ai, but he works separately to the PPP so unfortunately no guarantees can be made.

And that's it, we're done! This is all we need to do to create a basic dataset of sound clips and transcripts for any given source of audio. If you have any questions or points of clarification on any of the instructions written here or shown in the video, feel free to post them in the current

PPP thread. In the thread, you will find me under the name Clipper. You can also email me at clipper.anon01@gmail.com.   I always do my best to respond to all questions about creating datasets.

Automatic Clipping and Transcribing

This section is intended to assist with using the Automatic Super Speaker-Filtered Audio Processing (ASSFAP™) Colab notebook. The notebook allows you to use IBM's Cloud Speech tools to automatically clip and transcribe lines. While automated clipping may not be as accurate as manual clipping, it is a heck of a lot faster. These instructions will largely mirror what is in the notebook itself.

Open the notebook itself here:

Colab

The first thing you'll need to do is open the notebook in the playground. Click the button in the upper left to do so.
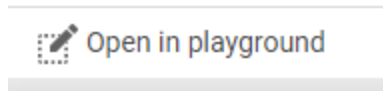
On the first code block you run, you will be prompted by Google whether or not you wish to run the notebook. Select "run anyway".

Before you upload your files to be clipped, you will first need to prepare them. You'll want to create a folder called "files" and put the audio files you want clipped inside. Make sure that none of the files are over 100mb else IBM's Cloud Speech tools will error out and the file will be skipped. I recommend using Audacity to cut your files into smaller pieces if needed. Make sure to cut where there is no speech.

Next you will need to create a small sample file of the speaker you wish to clip. It should be between 2 and 6 seconds long. Again, I recommend Audacity for this. Name the file "sample.wav" and place it outside of the file directory. When done it should appear as follows:
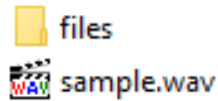
Add the files to a .zip file without any other subdirectories. When you open the zip, you should be able to see the files folder and the sample file.

Next, upload your .zip file to your Google Drive. Once done, you will need to publically share it and get the Drive ID. You will want to store this for later. For example:

Another thing you will need to set up is an IBM Cloud account. Do it **here**. Once you have set up your account, you will need to get an API key and URL for the Speech to Text tool. Be aware that the free plan only allows a limited amount of transcriptions per month.

From the dashboard click on "Create Resource" in the upper right hand part of the screen.
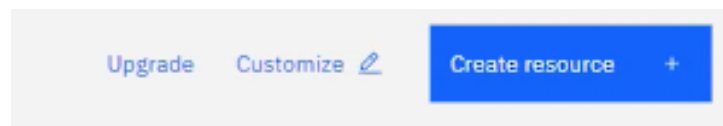
Click on "Services".

Click on "Speech to Text".
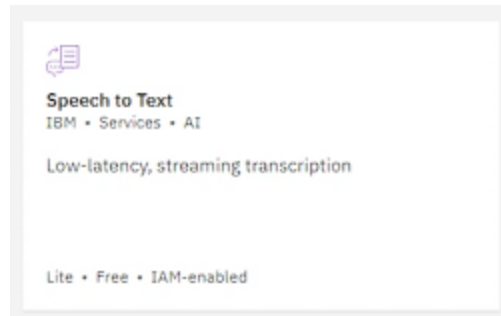
You will be given some options, but you can leave them as is. Click on "Create" in the lower right hand corner.
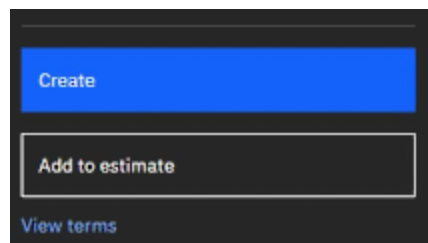
With that created you can go back to the dashboard (click the IBM logo at the top). Under resource summary, you should see "services" listed. Click on it.
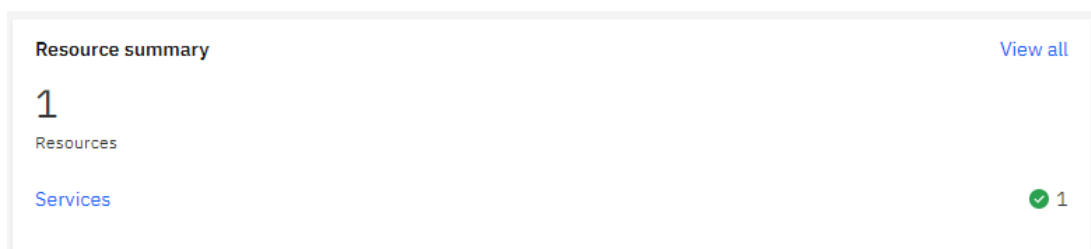
You'll then see something similar to below. Click on "Speech to Text-cy".

You will then have access to your API key and URL. Note these down for later.

Credentials

Download ⬇   Show credentials 👁

API key:

```
Your API key will be here                                      ⎘
```

URL:

```
Your URL will be here                                          ⎘
```

On to the notebook itself now. At the top of the first box you will need to set the sample rate and root path you want to use. If you want to use 22KHz data, leave sarate alone. If you want to do 48KHz, set it to 48000. Change the root path to where in your drive you want the output files saved.

```
[ ]  sarate = 22050 # Sample rate of your files
     root_path = 'gdrive/My Drive/Your/file/path/here/'  #change dir to your project folder
```

Once done, run the first cell. Note that you will need to authorize the notebook to access the contents of your drive when you run the cell (paste the authentication code into the box).

In the second cell, take your .zip file's Drive ID and paste it where "YOURIDHERE" is.

```
[ ]  !gdown --id YOURIDHERE -O dat.zip #Your data
```

In cell three you will need to enter your IBM API key and URL.

```
[ ]  ak = ""   #IBM Cloud speech to text API key.
     ibu = "" #IBM cloud speech to text API url
```

In cell four, you can change the output filename. You can set it to anything you want, all it changes is the filenames of the .tar.gz files the notebook produces.

```
[ ]  basename = "myCharacterName" # CHANGE THIS, the name of your output tar

     tarnm = basename + ".tar.gz"
     jstarnm = basename + "_json" + ".tar.gz"
```

At the top of cell five, you can adjust the audio volume for all clips if overall they are too loud.

```
[ ]  from tqdm.notebook import tqdm # Modern Notebook TQDM
     import shutil
     vomul = 0.8 # Some audio files are very loud in their current ite
```

The next section (cell 6) can be ran as is if you want to produce a 22KHz dataset. For 44KHz, you will need to modify it with the ARPA function from the 48KHz Synthesis notebook. Replace the existing ARPA function with:

```
!git clone https://github.com/jeroenmeulenaar/python3-mega.git
!(cd python3-mega; pip install urlobject pycrypto)

import os
os.chdir('python3-mega')
from mega import Mega
os.chdir('../')
m = Mega.from_ephemeral()
print("Downloading Dictionary...")
m.download_from_url('https://mega.nz/#!yAMyFYCI!o_UmixbiIzosyYk-6O5xRZZDGpFRik_eMrZ
um-iQuhQ')

def ARPA(text):
    out = ''
    for word_ in text.split(" "):
        word=word_; end_chars = ''
        while any(elem in word for elem in r"!?,.;") and len(word) > 1:
            if word[-1] == '!': end_chars = '!' + end_chars; word = word[:-1]
            if word[-1] == '?': end_chars = '?' + end_chars; word = word[:-1]
            if word[-1] == ',': end_chars = ',' + end_chars; word = word[:-1]
            if word[-1] == '.': end_chars = '.' + end_chars; word = word[:-1]
            if word[-1] == ';': end_chars = ';' + end_chars; word = word[:-1]
            else: break
        try: word_arpa = thisdict[word.upper()]
        except: word_arpa = ''
        if len(word_arpa)!=0: word = "{" + str(word_arpa) + "}"
        out = (out + " " + word + end_chars).strip()
    if out[-1] != "␤": out = out + "␤"
    return out

thisdict = {}   # And load it
for line in reversed((open('merged.dict_1.1.txt', "r").read()).splitlines()):
    thisdict[(line.split(" ",1))[0]] = (line.split(" ",1))[1].strip()
```

When done, it should look like this:

```
#This is the ARPA function from the 22KHz notebook. If you want to create a 48KHz MMI dataset,
#then you should paste the ARPA function from there.
!git clone https://github.com/jeroenmeulenaar/python3-mega.git
!(cd python3-mega; pip install urlobject pycrypto)

import os
os.chdir('python3-mega')
from mega import Mega
os.chdir('../')
m = Mega.from_ephemeral()
print("Downloading Dictionary...")
m.download_from_url('https://mega.nz/#!yAMyFYCI!o_UmixbiIzosyYk-6O5xRZZDGpFRik_eMrZum-iQuhQ')

def ARPA(text):
    out = ''
    for word_ in text.split(" "):
        word=word_; end_chars = ''
        while any(elem in word for elem in r"!?,.;") and len(word) > 1:
            if word[-1] == '!': end_chars = '!' + end_chars; word = word[:-1]
            if word[-1] == '?': end_chars = '?' + end_chars; word = word[:-1]
            if word[-1] == ',': end_chars = ',' + end_chars; word = word[:-1]
            if word[-1] == '.': end_chars = '.' + end_chars; word = word[:-1]
            if word[-1] == ';': end_chars = ';' + end_chars; word = word[:-1]
            else: break
        try: word_arpa = thisdict[word.upper()]
        except: word_arpa = ''
        if len(word_arpa)!=0: word = "{" + str(word_arpa) + "}"
        out = (out + " " + word + end_chars).strip()
    if out[-1] != "NL": out = out + "NL"
    return out

thisdict = {}    # And load it
for line in reversed((open('merged.dict_1.1.txt', "r").read()).splitlines()):
    thisdict[(line.split(" ",1))[0]] = (line.split(" ",1))[1].strip()

#Recognize an audio file using the IBM API in American English
```

The next block of code is where the actual clipping and transcription take place. At the top you will see some parameters to be tuned. The functions of the parameters are as marked.

```
[ ]  import librosa
     import soundfile as sf
     import random
     import shutil
     import math
     # ====== PARAMETERS ===========

     valper = 10 # How much % of the data to be used for validation. Default is 10%

     maxflen = 12.0 # Max length of audio file
     maxlen = 10.5 # Max length of recognition before it forcibly stops

     minsecs = 1.5 # Minimum amount of seconds
     #Do note that setting this to too high may make files contain junk audio detrimental to training
     mxdiff = 0.3 # Max difference in seconds betweeen words
     thres = 0.7 # How confident Resemblyzer must be that the speaker is there. Values below 0.65 are not recommend
     # --
     addcommas = True # Add a comma if a  certain percentage of the difference threshold is reached
     permxd = 0.6 # As percentage of the mxdiff value
     addperiods = True # Add a period if a certain percentage of the diff thres is reached
     perdota = 0.9

     maxtranslen = 450 #If the transcription becomes longer in length (in characters) than this value,
     # the clip is prematurely ended. Thanks fast-talking 3kliksphilip for forcing me to add this.

     # ======= PARAMETERS ==========
```

Once clipping has begun, you should see something similar to below:

```
Starting 16/17 : 5.flac
Requesting from IBM Cloud. This will take a while.
Acquired valid data of 799 words, starting clipping
100% [████████████████████████]  798/798 [03:08<00:00, 4.23it/s]

Extracted secs: 226 Total: 313
Extracted: 72.3%
Clips: 52 - Average clip length: 4.36
```

After all has been clipped and transcribed, you are ready to package the data. Note that if using a 48KHz dataset, change the indicated text to 48000.

```
!rm -rf data/wavs/*.wav

print("Normalizing, please wait")
!normalize-audio -q -m prewav/*.wav
print("finished normalizing")

files = os.listdir("prewav")
safemkdir("data/wavs")
for fi in tqdm(range(0,len(files))):
  f = files[fi]
  fpath = "prewav/" + f
  nupath = "data/wavs/" + f
  retval = subprocess.run("ffmpeg -i " + fpath + " -acodec pcm_s16le -ac 1 -ar 22050 " + nupath,shell=True)
```

```
[ ]  import shutil
     print("Tarring")

     !tar czf {tarnm} data
     shutil.copy(tarnm,root_path)
```

```
[ ]  print("Saving JSONs")
     !tar czf {jstarnm} json
     shutil.copy(jstarnm,root_path)
```

When the last of the cells are run, you will find two .tar.gz files in your Google Drive at the path
you set.

Character_json.tar.gz   Character.tar.gz

The _json .tar.gz contains metadata for if you want to reuse the same transcriptions later. The
main file you will be interested in is the regular .tar.gz file.

Note that this is NOT in the same format as the standard pony datasets. To use with the training
script you have two options. You can treat the extracted files as a standard custom dataset or
you can replace the data import cell with the following:

```
import shutil, os

data_path = '/wavs'
!rm -rf wavs

print("down flv")
!gdown --id YOURIDHERE -O dat.tar.gz
!tar -xzf dat.tar.gz

print("move")
shutil.move("data/filelist.txt","filelists/flist.txt")
shutil.move("data/valist.txt","filelists/vallist.txt")
shutil.move("data/wavs","wavs")

# On the training files and model name
hparams.training_files = "filelists/flist.txt"
hparams.validation_files = "filelists/vallist.txt"
```

Just create a new cell in section 3 of the notebook, paste the code in and run it when you are ready to import your data. Make sure to replace YOURIDHERE with the Drive ID of your exported dataset.

Cleaning Audio

Simple Audacity Edits

Audacity has a noise reduction tool that is primarily designed to remove background "hiss" that is commonly found with low-quality microphones. This tool can be adapted for use on any background noise that is constant and consistent, such as fan noise.

We will demonstrate the noise reduction process on this clip from the "Dead Air" Dr. Who Audiobook. Put on your headphones and listen carefully, you should be able to hear a "hiss" in the background.

Here is a video demonstration of the process. To use the noise reduction tool, you will first need to find an isolated sample of the noise you want to remove. The minimum sample size required is about half a second, but always try to find the largest sample you can. Here is the sample of the "hiss" that we will use. Once you have found a suitable sample, use the select tool to click and drag from the start to end of the sample, this will highlight the selection. Once done, go to "Effect" -> "Noise Reduction". In the window that pops up, click "Get Noise Profile". This is effectively you giving Audacity an example of what you want to be removed.

Now highlight the section of audio you want to remove the noise from, and again go to "Effect" -> "Noise Reduction". The default settings should work fine, so just click "Ok". Listen back to the sample to verify that the noise has been removed. If there's still some left, you can repeat the noise reduction process. Note that running the noise reductions several times will degrade the quality of the audio, so at some point you may need to make a compromise.

Here is the noisy clip that's been processed by the method described above. You should be able to hear that the "hiss" is almost completely removed.

For any noise or undesirable sound effects that are completely isolated, there is a much simpler solution. Just highlight the noise you want to remove, and then go to "Generate" -> "Silence" and press "Ok". This will replace the isolated noise with silence.

iZo method

The process we've come up with to clean the audio in the least destructive way takes advantage of the identical background noises present in forign dubs of the show. The short of it is that we align two dubs and use a center channel extraction tool to remove the similarities between the two tracks. Below are two quick demos of the process, do note that these are slightly out of date and is best to use this document as the definitive guide for this process.

Quick Video: YouTube
Narrated Tutorial: YouTube

To begin, you will need to locate two dubs of the show that are as close to each other as possible. The best source we've found for this is Netflix rips of the show. Do note that 5.1 rips are necessary for both the English and forign dub that is used. Due to this, the later seasons of the show will not be able to be processed in this manner as only stereo forign dubs exist at this time. See Resources.

Once you have procured the audio files you will be working with, both will need to be processed according to either this guide or using the mlp_dialog_rip.sh script from tools (It uses the same process). While the 20db reduction mentioned in the guide is correct for English dubs, it may not be correct for all forign dubs. One Anon suggests 14db is more often correct in these instances. In all cases, use your ears to determine the proper value of this reduction.

With that done you should be left with two mono audio tracks, one in English and the other forign.
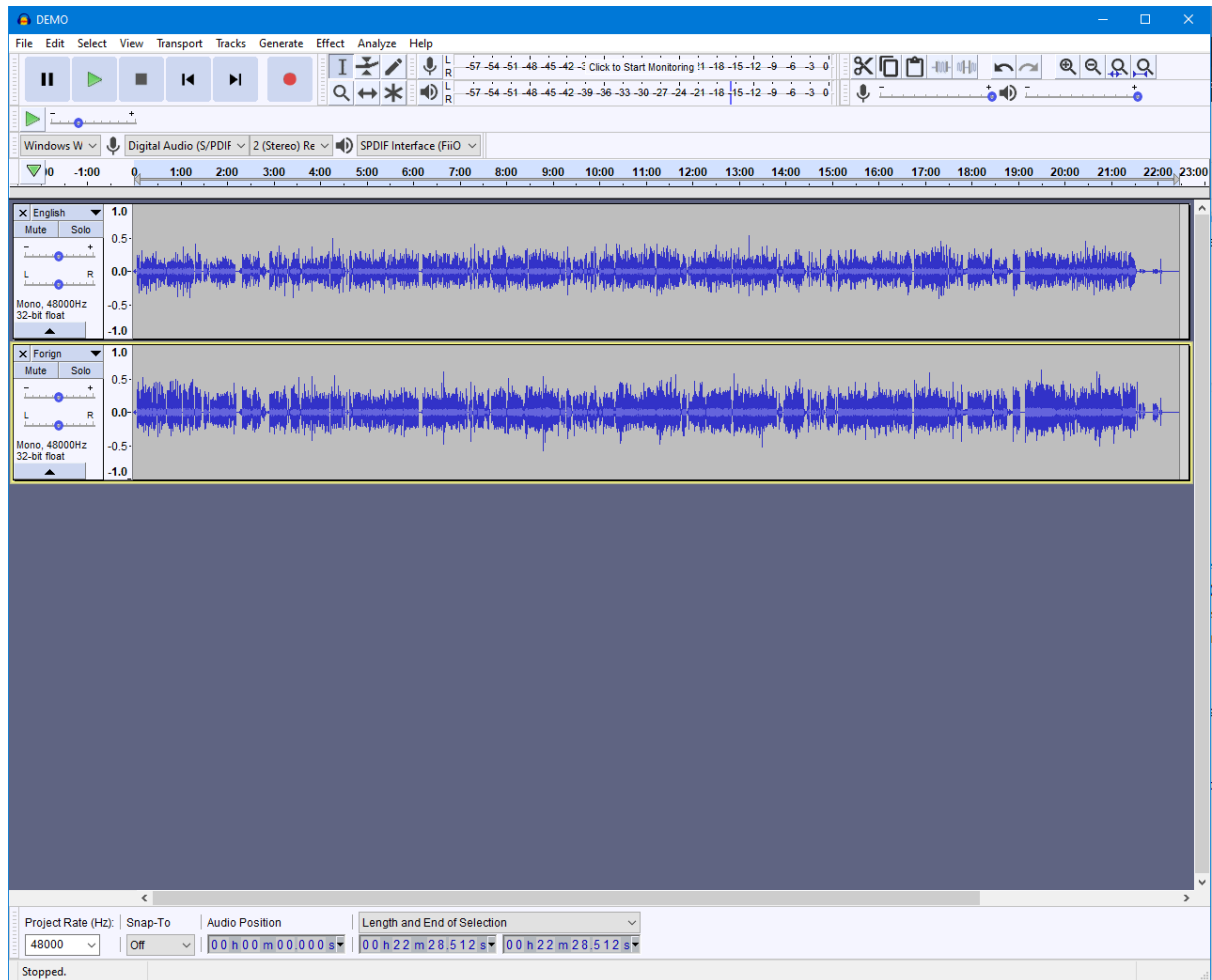
Your task here is to ensure that the two dubs are perfectly lined up. This is extremely important, so much so that the alignment needs to be sample perfect. That is to say that they must line up even when you zoom all the way into each individual sample (Use ctrl+alt+scroll to zoom in and out).

The best way to go about this is to find a location in the audio where both have a distinctive peak that can be used as a landmark. Take this hoofstep as an example, it may not be a large peak, but it is alone and isolated in each track.

Use the time-shift tool to adjust the **forign** dub to match up with the English. It is important that the English track stays in place so as to maintain alignment with the rest of the project.

Once you've got it close you will want to zoom in even closer. However you may notice that the graph flattens out and makes it hard to do alignment. The solution to this is to use the vertical zoom feature of Audacity. If you click on the time scale to the left of the graphs (Notice the red box), it will zoom in vertically. Use right click to zoom back out.
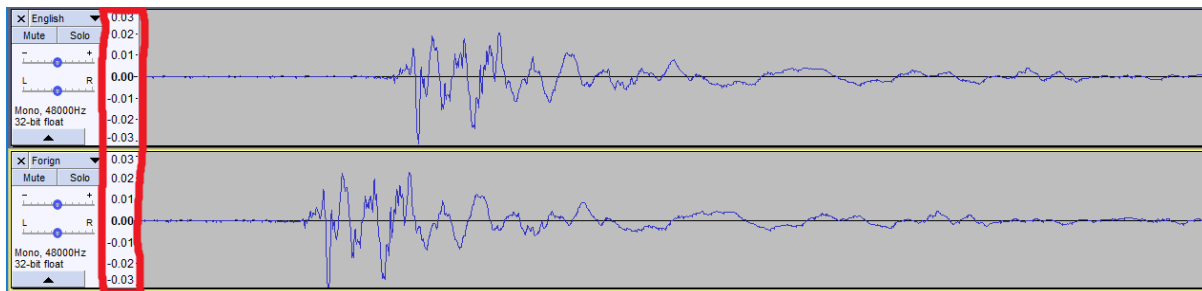

As you can see it is now much easier to align the tracks. You will want to continue to align the tracks as you zoom in until you have the tracks aligned to the sample. Aligned tracks should look like the following at this scale.
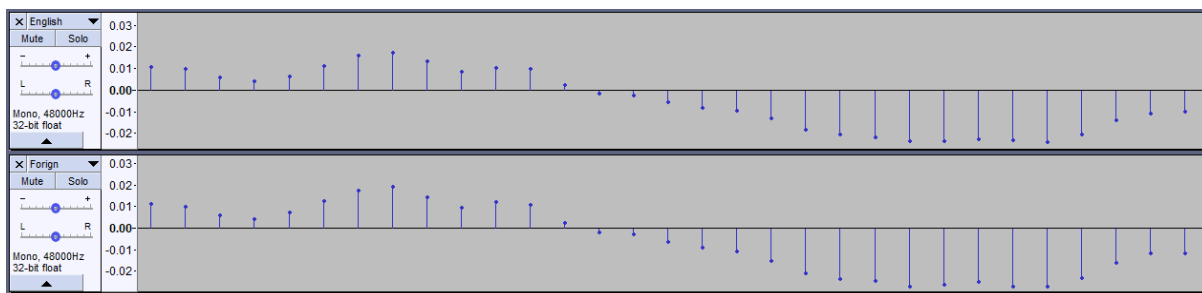
After you have aligned one spot, it is important to check alignment at other parts in the episode. It is not uncommon for the tracks to go out of alignment where commercial breaks would be. If this does occur, split the forign dub by selecting a point with the selection tool and pressing ctrl+I. Make sure that you are not splitting the track when any dialog is taking place. Once you've done this you will be able to adjust the two pieces of the track separately.
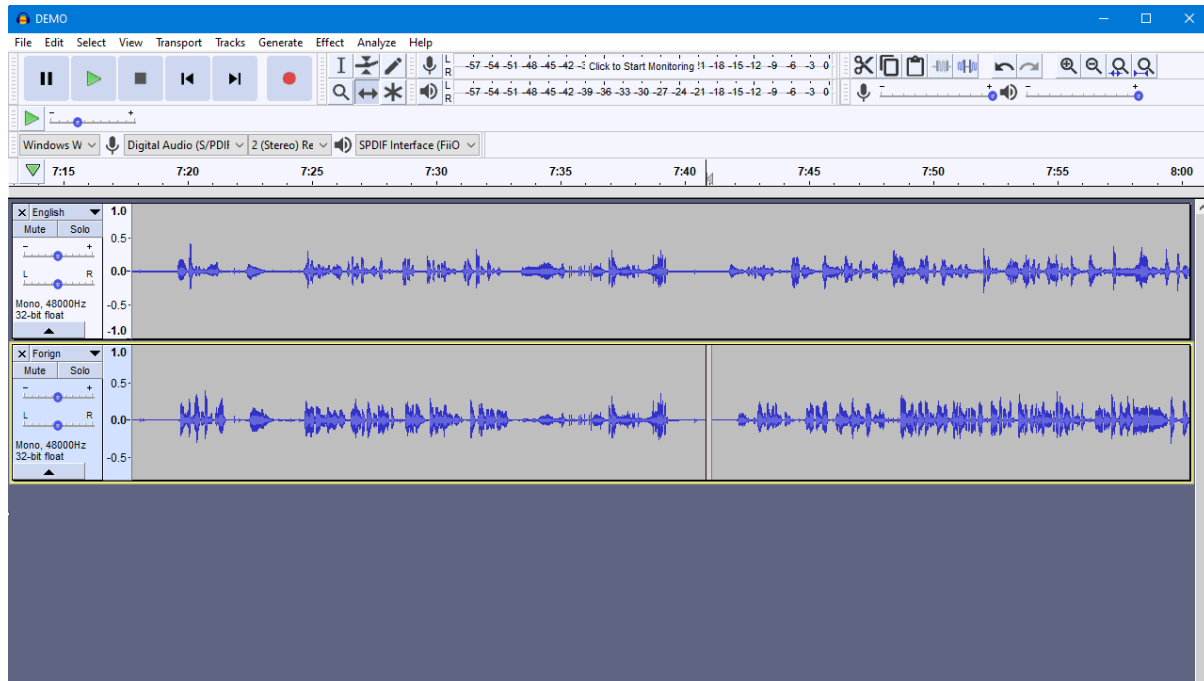
Once you are confident that the two tracks are aligned you will need to balance the tracks. This means ensuring that both tracks are at exactly the same volume. To do this it is recommended to find a section of the audio with a distinct sound effect, free of dialog (such as a hoofstep). You will want to make a selection of that noise on the English track and click on Effect->Amplify.
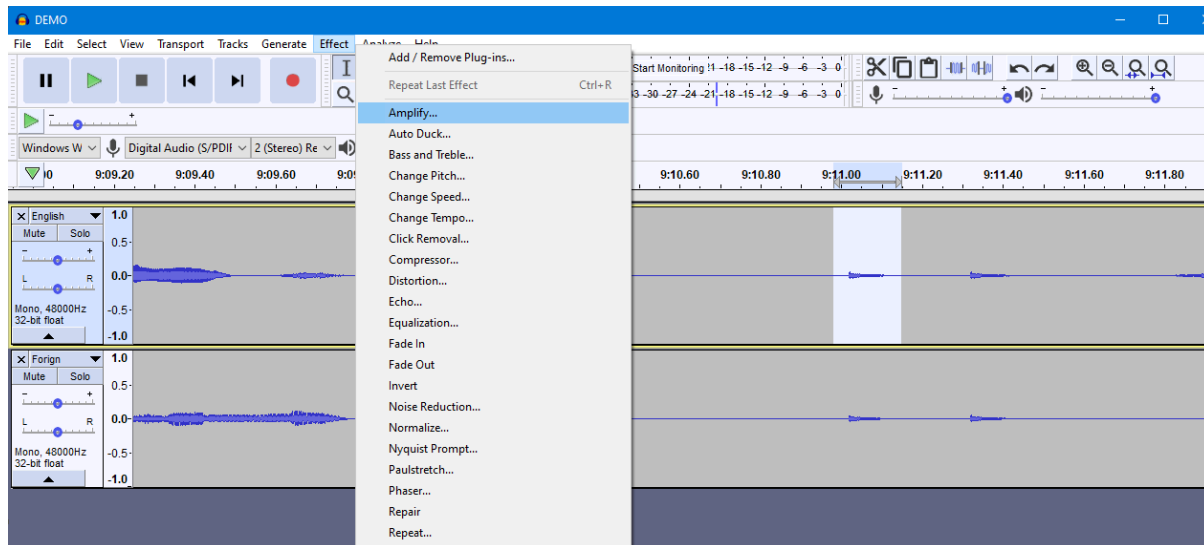
Enter 0 for the amplification and take note of the new peak amplitude. You will want to copy this into your clipboard and close the window.
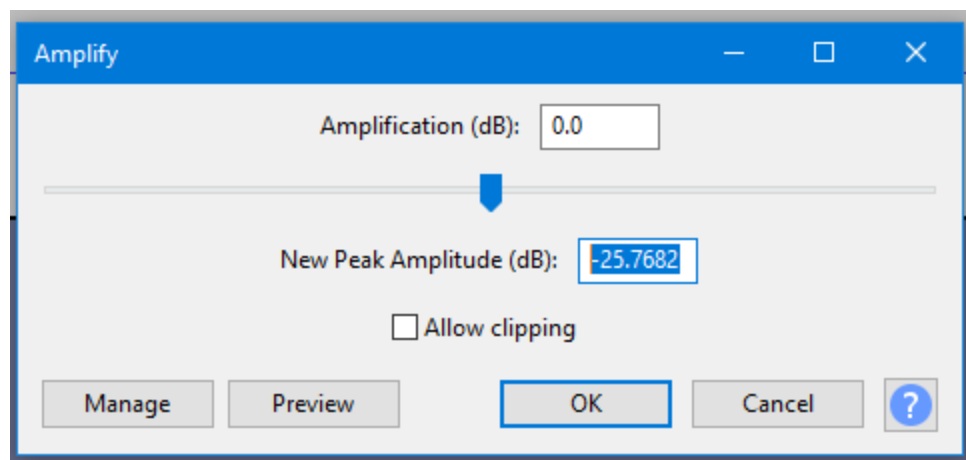
You will then want to make the same audio selection on the other track and click on Effect->Amplify again. Paste the new peak audio from before into the new peak audio here.
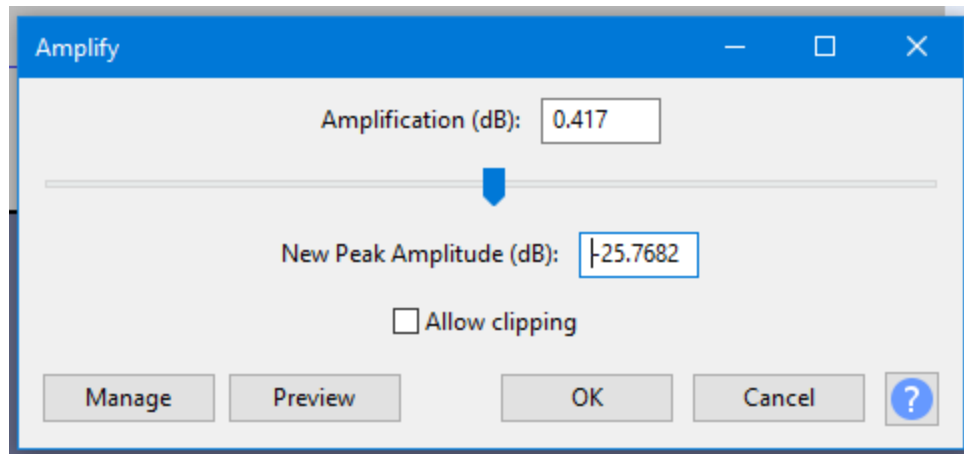
Take note of the amplification value that this gives. Copy this into your clipboard. Now select the entire forign dub (You can click anywhere that is empty in the box to the left of the track to select that track) and use Effect->Amplify to amplify the entire track by this value.
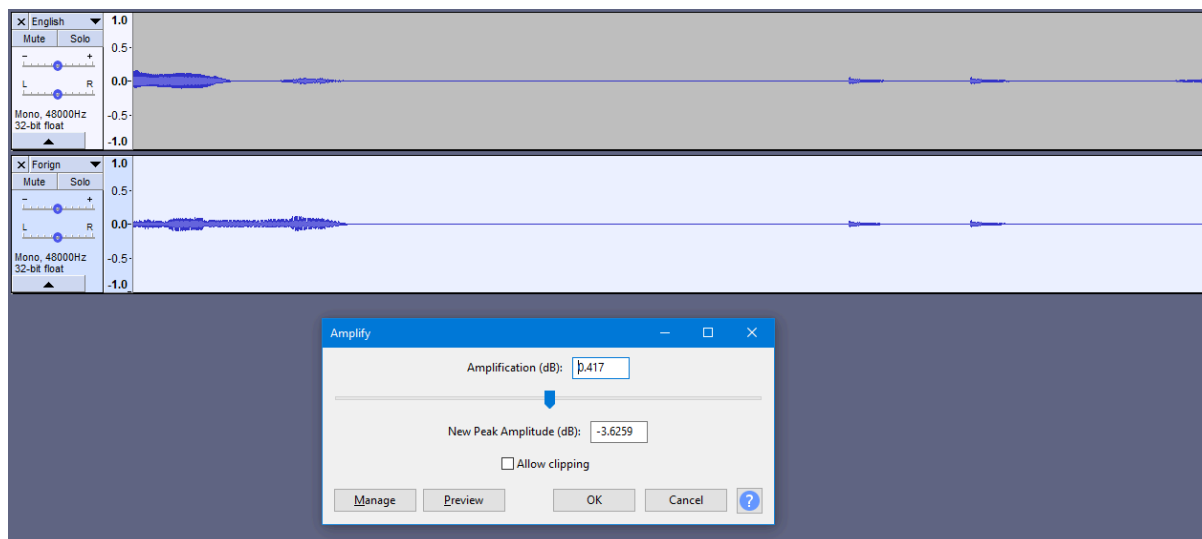
At this point you can check how well aligned and balanced the tracks are. If you invert one of the tracks, all background sound effects should be gone (You will hear both sets of dialog when you do this).
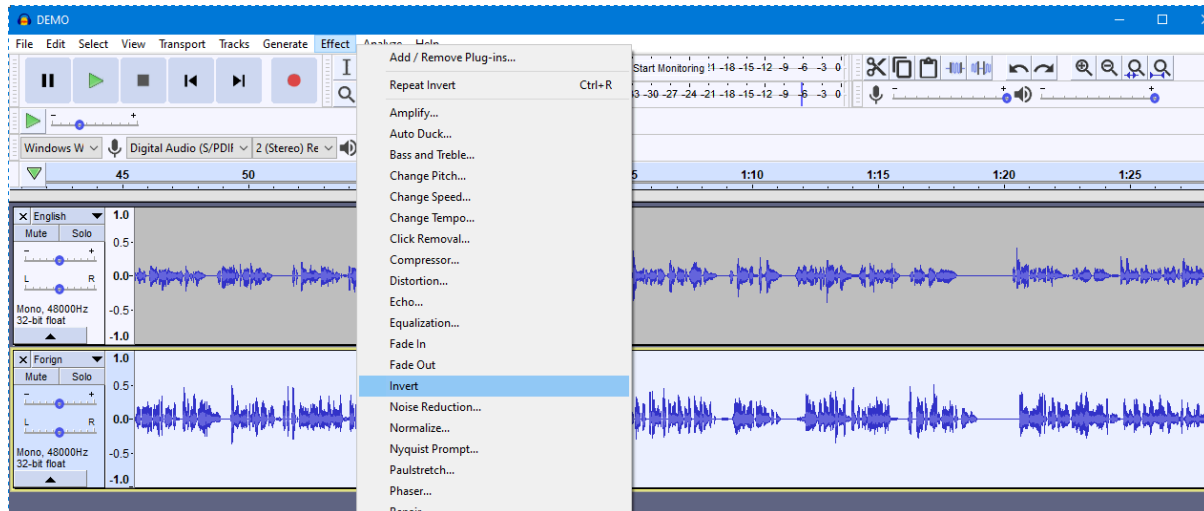
Ensure that neither track is inverted for the following steps. You will want to select both tracks (ctrl+A), click the dropdown menu to the left, and select "Make Stereo Track".
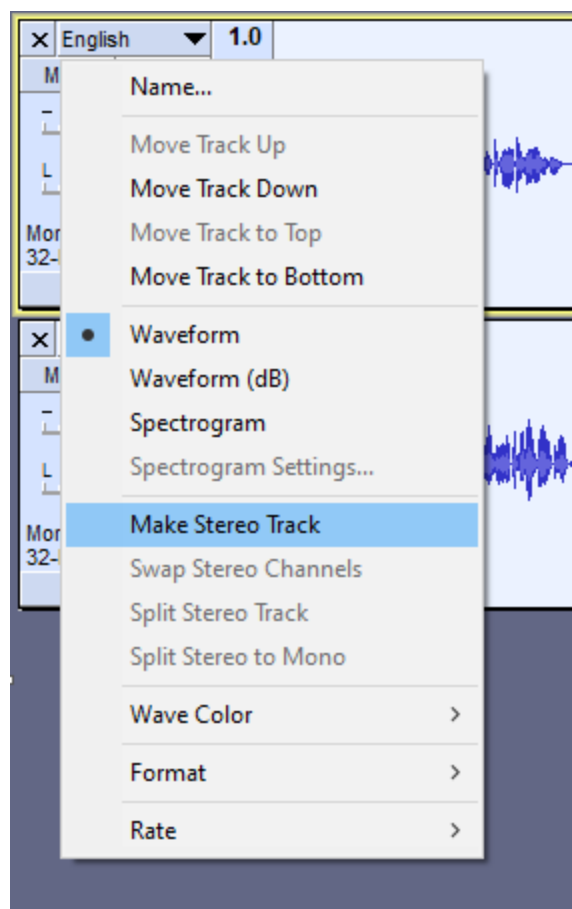
The two tracks should now appear as a single stereo track, with the English dub panned entirely to one side and the forgin dub entirely to the other.
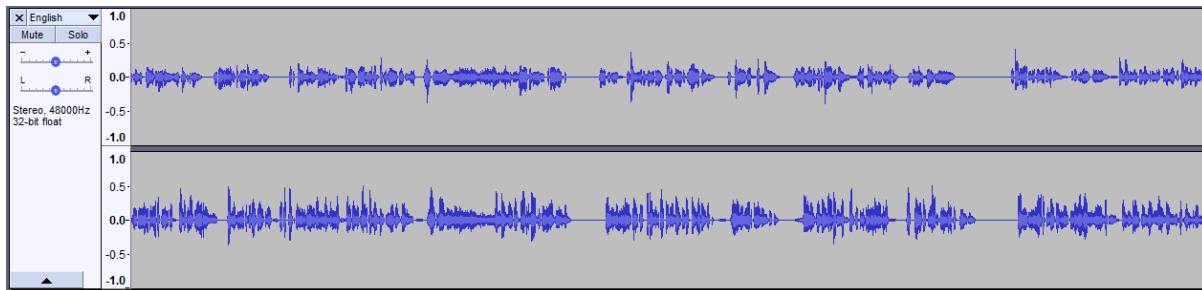
Now you will want to export your track to a lossless format (such as FLAC) using File->Export->Export Audio....
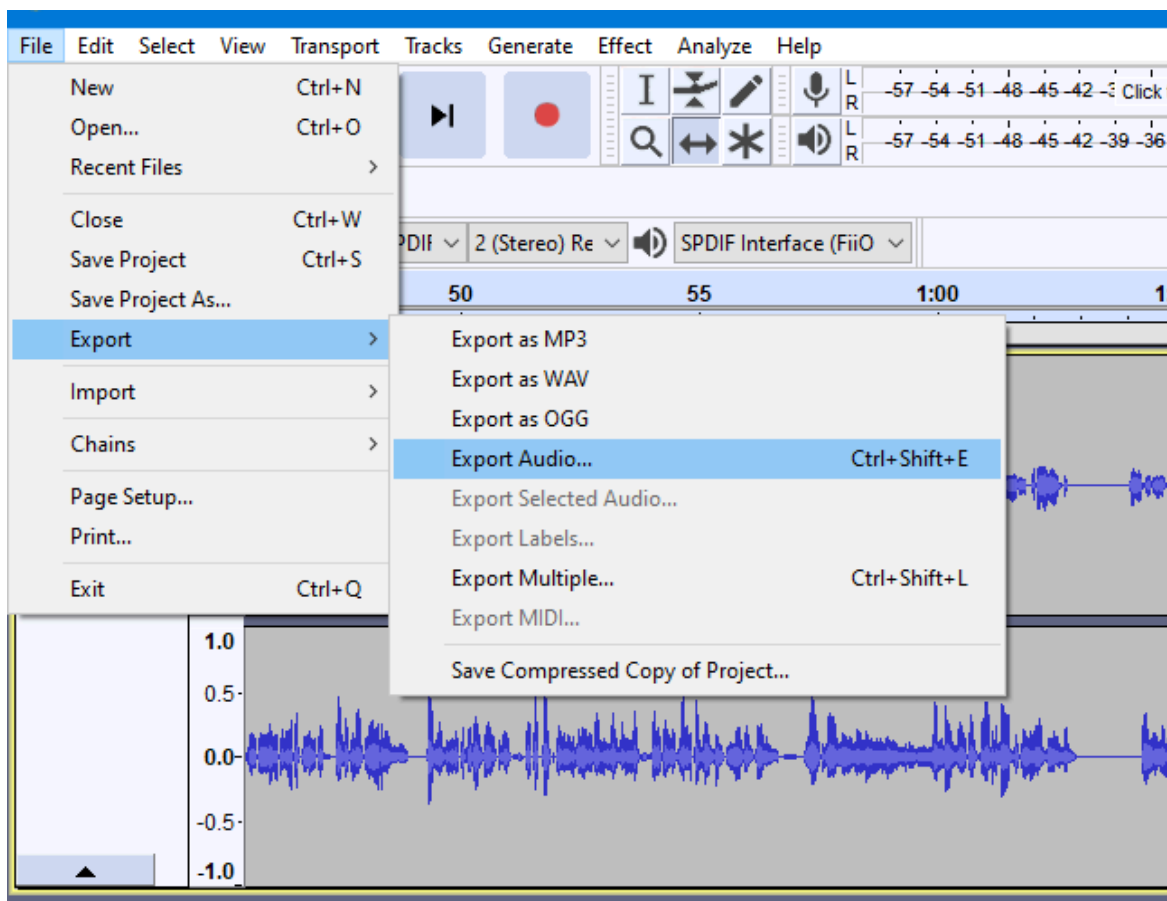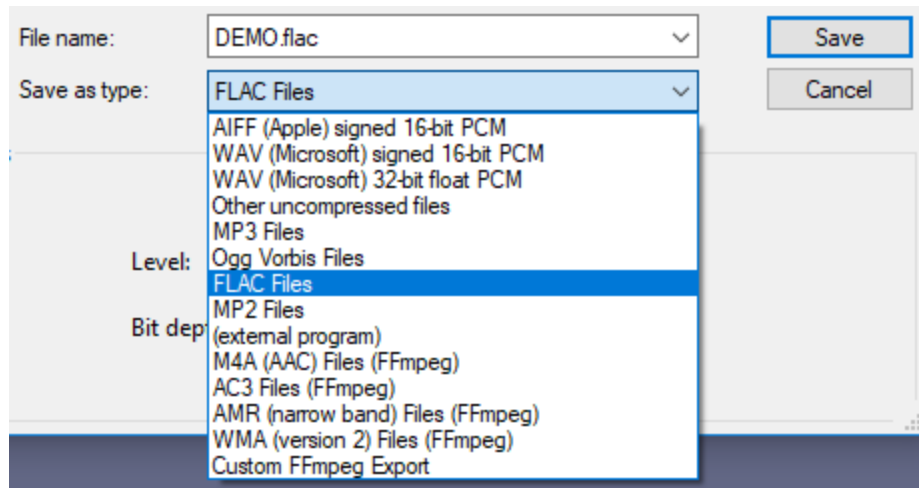
For the next step we will need iZotope RX7 ([See Resources](#)). Open your exported track.
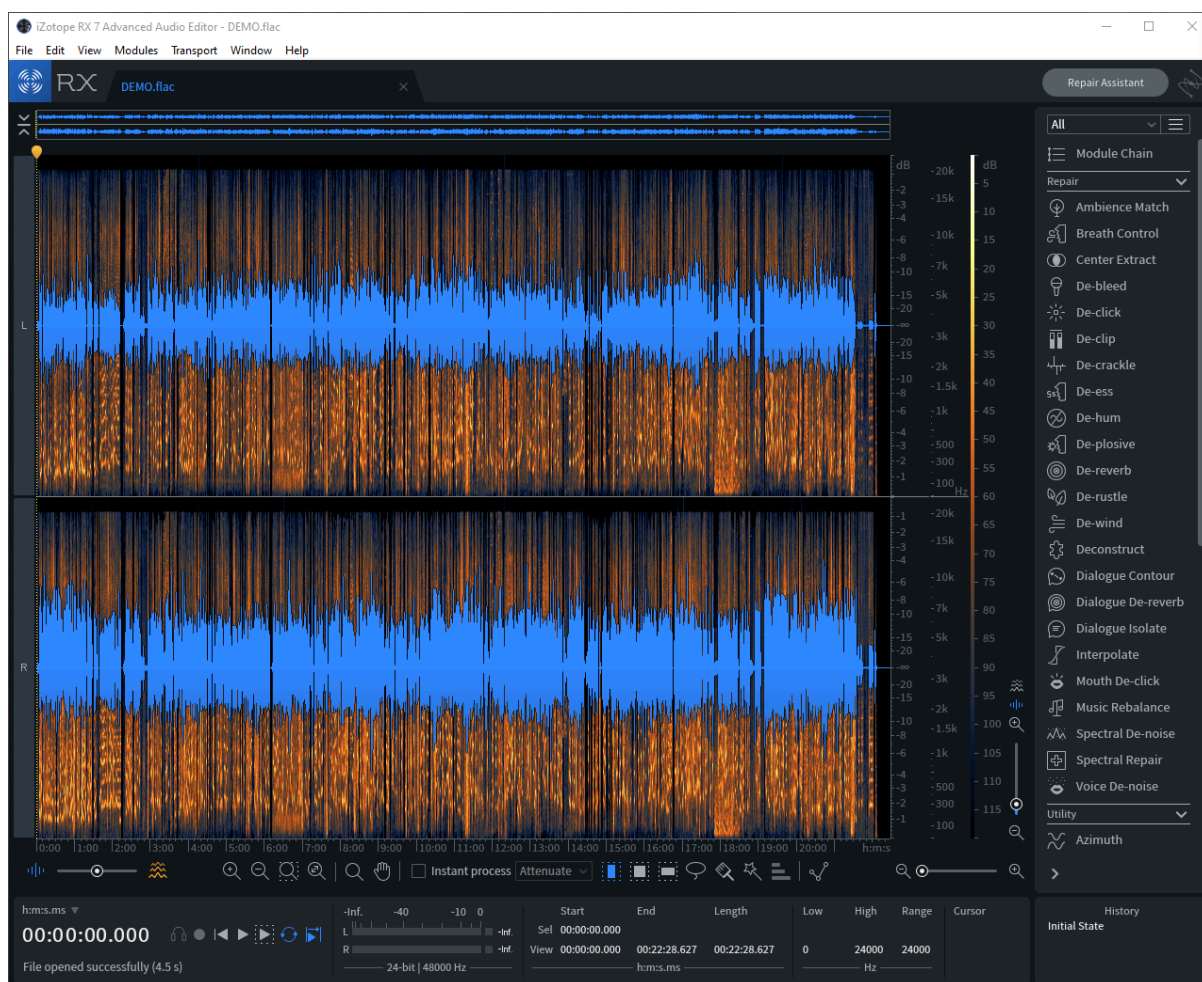
You will want to use the "Center Extract" tool on the right. Select "Keep Sides" and "True Phase". Set reduction strength to 1.5, and artifact smoothing and dry mix to zero. These are the settings we've had the best luck with.
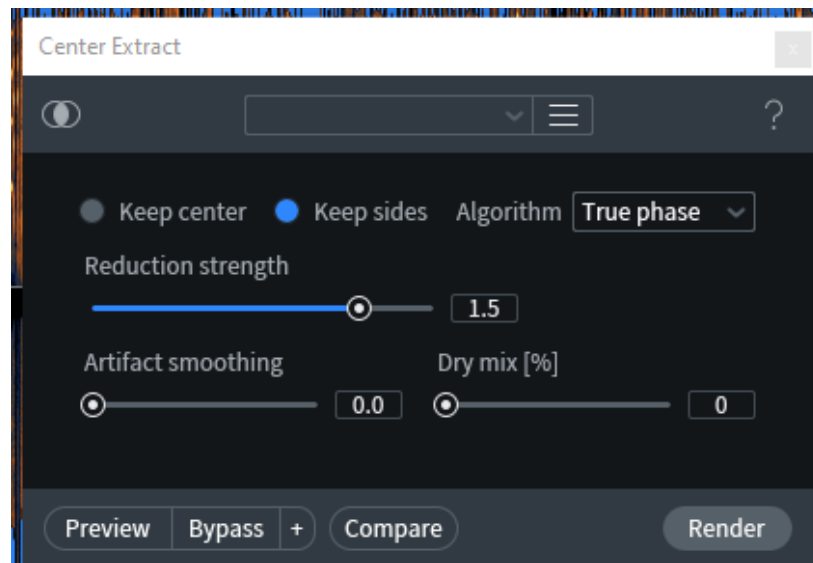
Once done configuring the settings, select render. This may take a few minutes for a whole episode.

Once done, export your track in a lossless format (FLAC) and import into a clean Audacity window.
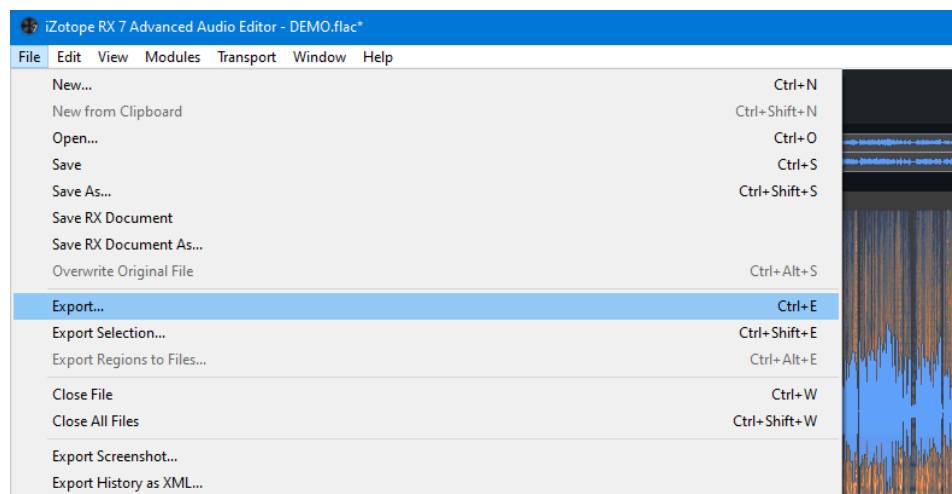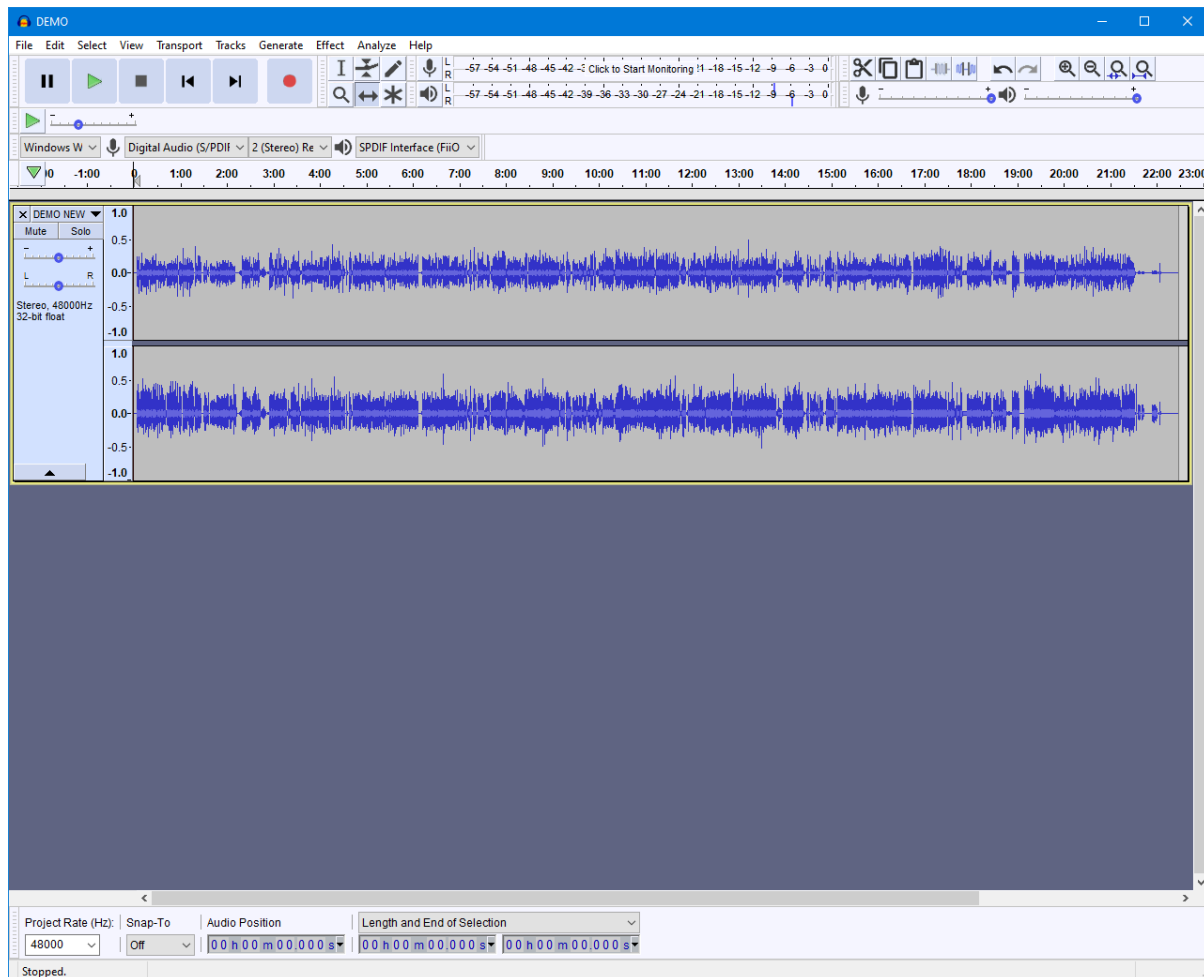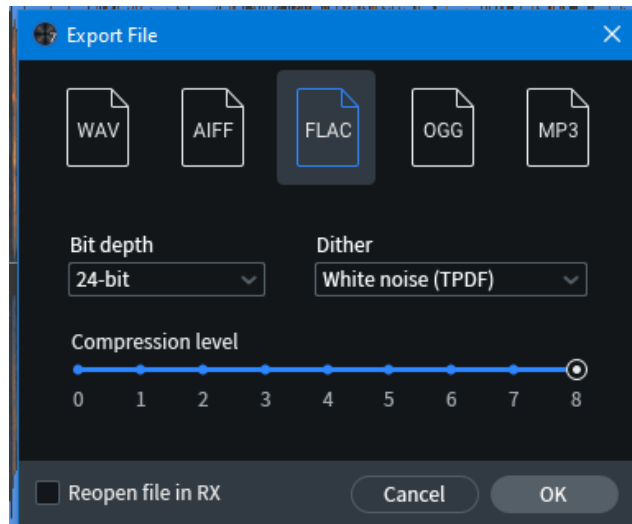
Click on the dropdown menu to the left of the track and select "Split Stereo to Mono".
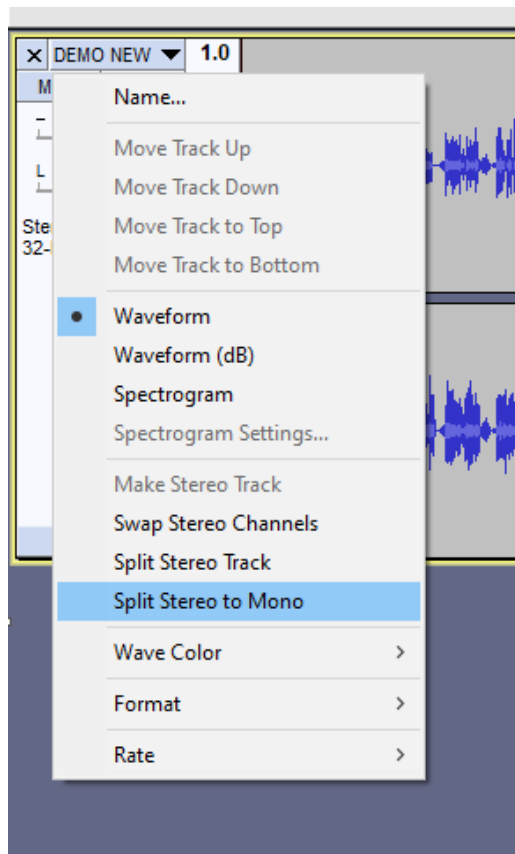
It should now look something like this.
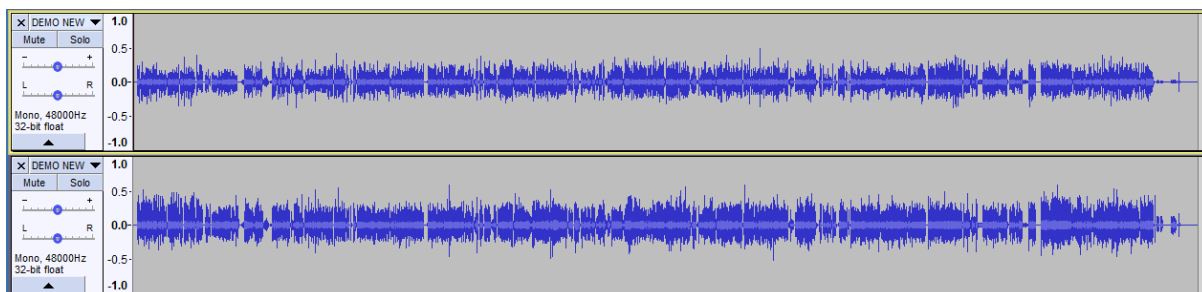
Click on the x to the left to remove the forign track. You should be left with just the cleaned English dub.
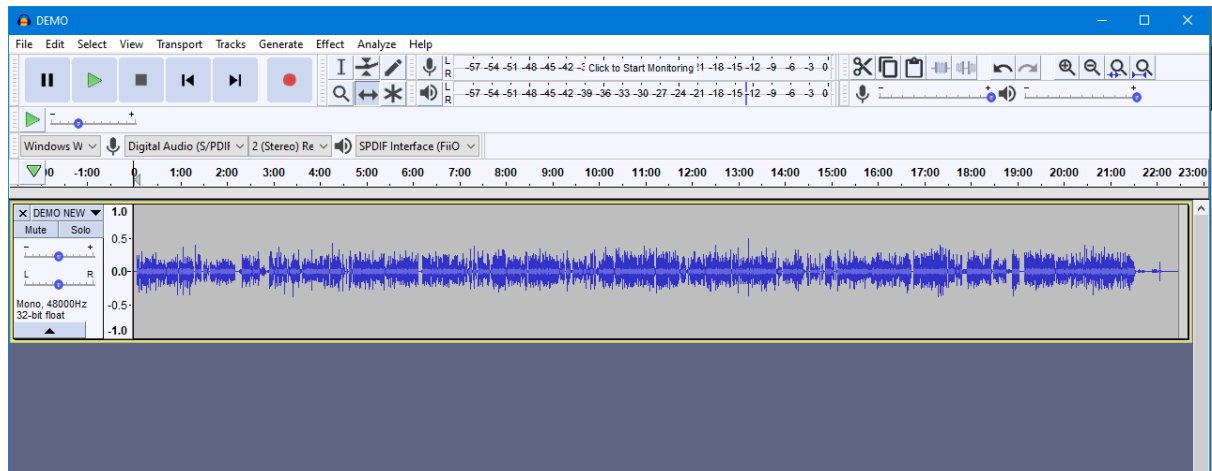
At this point you are essentially done. Give it a listen at a few points to ensure the process worked successfully. Export as FLAC, upload to your favorite file host, and submit in thread.

Using RTX Voice

RTX voice is nVidia's fancy new AI microphone audio cleaner. However, it doesn't just have to be for live audio. You can use it for prerecorded stuff as well.

Requirements: An nVidia GPU.

If you have an RTX card, you can just run the installer from nVidia.

RTX Voice Page

If you have a GTX card, you can follow **this guide** to get RTX voice up and running.

Once you've got it installed, you can capture the output audio via Audacity. Make sure you have the API set to WASAPI (it's the only one I found to work) and the microphone set to the loopback of your default audio output device. Once done, you're basically good to go.

If you do not want RTX voice to run at startup, you can disable it via the task scheduler. Note that the RTX voice application does not show up in the task manager startup list. You can see an example of the task scheduler entry below. Right click and disable the task to stop it starting at login.

RTXVoice_{B2FE1952-...   Disabled   At log on of any user
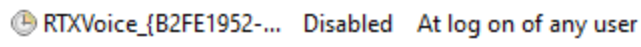
Some samples:
CNC machine
Jet engine
Hoofsteps and music
Various noises

As you can hear, the RTX voice cleaning works best with constant sounds.

Open Unmix

*I performed the following in Debian. As per the GitHub, Windows support has not been tested. I used a virtual machine.

Open Unmix is open source software that makes use of neural networks for music separation. We have used it to help us further clean audio of background noises.

To use Open Unmix, you will need to have Python installed as well as the dependencies for Open Unmix. You can install them with the following command.

pip3 install torch musdb norbert librosa

Clone the Github repository with:

git clone https://github.com/sigsep/open-unmix-pytorch.git

The full list of options for Open Unmix can be found at **THIS** page. The ones we will be interested in are --model and --targets. We will set --models to use the default models and --targets to only run the vocals model. Inside the open-unmix-pytorch folder you can run the following command to process audio. Replace track.wav with the path to your sound file.

python3 test.py track.wav --model umxhq --targets vocals

When complete, the processed files will be placed into the track_unmxhq subfolder. There will be a vocals.wav and accompaniment.wav. Vocals will be the extracted voice data and accompaniment will be everything left over.

Creating ngrok links

Video demo - https://u.smutty.horse/lwcswhmdfbf.mp4

These steps explain how to get an ngrok link to Cookie's Multispeaker Colab Notebook. This lets anyone use a Colab server to create audio clips voiced by any of a few hundred characters.

You don't need a fast connection or powerful computer to do this. This uses Google's resources to host a server. Ngrok lets you expose Google's server to the public internet so anyone can access it.

1. Open Cookie's scripts in Colab.
   a. https://colab.research.google.com/drive/1UjSg4tDcubbkax781fE0pNeAFdht_MZ0?usp=sharing
2. Click the "Copy to Drive" button. This button is tiny and gray, so it's hard to see. Ctrl+F for the text.
3. Follow the instructions in step "1 - Mount Google Drive and add model shortcut"
   a. You may need to click the folder icon on the left. You'll see an option for "Mount Drive" once you do.
4. Run all of the cells one at a time in sequence until you reach step 3. Once you run the cell that ends with "!python3 app.py", you'll get a link to the server. It will take about a minute before the link is active.
5. In the same window, open the Dev Tools Console
   a. In Chrome, the hotkey to open Dev Tools is Ctrl + Shift + J
   b. In the window pane that opens up, select the "Console" tab
6. Copy/paste the following and hit Enter. This will get the Colab instance to stay running for longer.
   ```
   function ClickConnect(){
       document.querySelector("paper-button#ok").click()
   }
   setInterval(ClickConnect,60000)
   ```
7. Post the link in the thread so other anons can use it.

For a more detailed guide, see the Inference Server guide.

Using the AI scripts

Thanks to Cookie and Synthbot, anybody can now begin training a TacoTron2 model. This guide will take largely from their instructions and posts in thread. All that will be needed is a Google account in order to use Colab.

[disclaimer]The AI side of things is still being worked on. Always be sure to check the threads for the most up to date information on the process and resources.[/disclaimer]

Preparations

Using Preprocessed Data

Before anything, a copy of the dataset must be present in your Google drive. Copy Synthbot's "Soundtools" into your drive.

https://drive.google.com/drive/folders/1SWIeZWjIYXvtktnHuztV916dTtNyIrpD
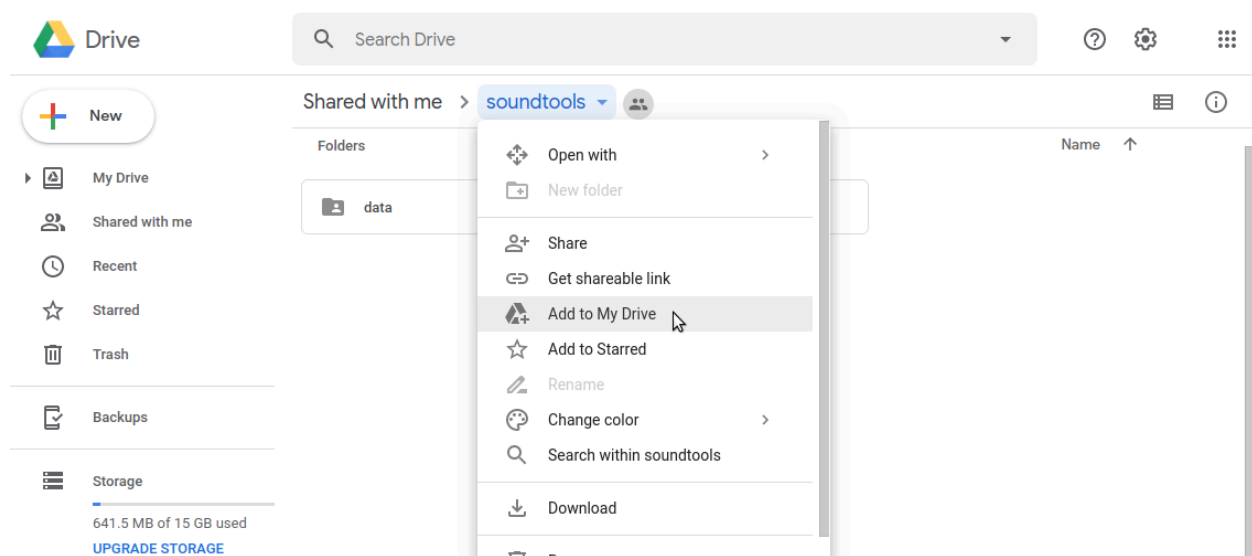
Making Your Own

Option 1: Using Synthbot's tools

Don't do this unless you're a developer.

Video Demo: YouTube

This will allow you to create tar files just like the ones available from the soundtools folder. Just upload your tar files into your soundtools folder, and point the training notebook towards it.

Find the directions on using Synthbot's tools on his Github here.

If you don't have a linux machine, you can use Virtualbox. Just set up a linux install and follow the instructions on Github.

Option 2: Using audio and text files

You will need to have your audio files in .wav format (48KHz 16bit mono) and a properly formatted filelist.

The filelist should contain a list of all file paths and the accompanying transcription. For example:

> /wavs/out/filename1.wav|Transcription text 1
> /wavs/out/filename2.wav|Transcription text 2
> /wavs/out/filename3.wav|Transcription text 3

And so on.

Once you've run through the first part of the notebook up through the block that creates /tacotron2/wavs/out/, you will need to upload your .wav files here.
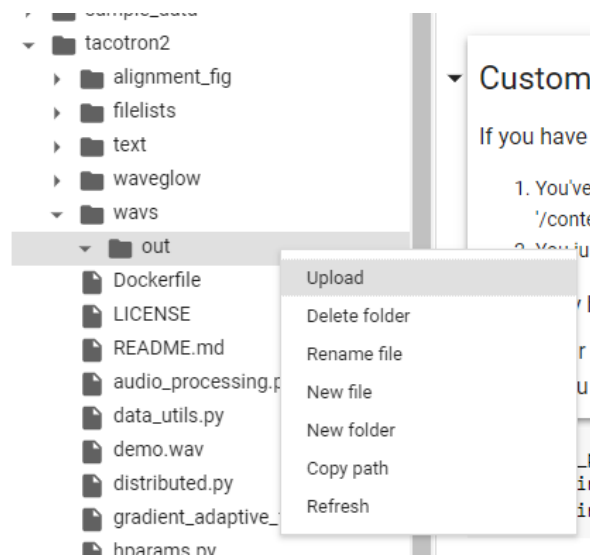
You will then need to upload your filelist into /tacotron2/filelists/.
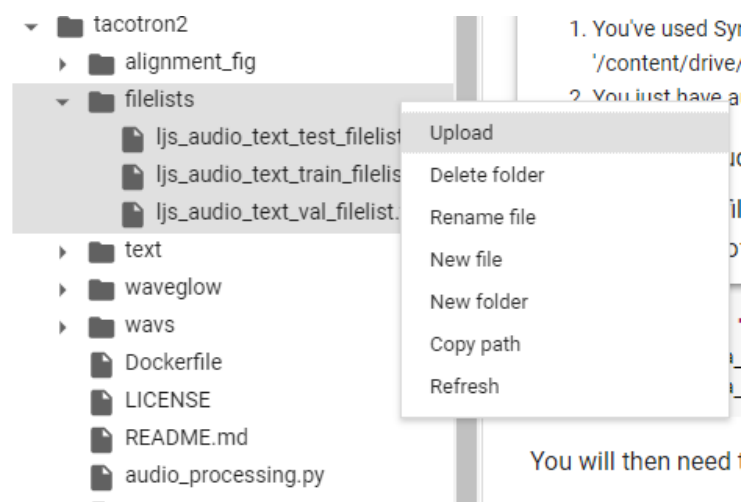
Refer to the appropriate training tutorial for further instruction.

Running Google Colab Scripts Locally

*General disclaimer, am not a Linux guru. These are just the steps that I took.

YouTube demonstration: YouTube
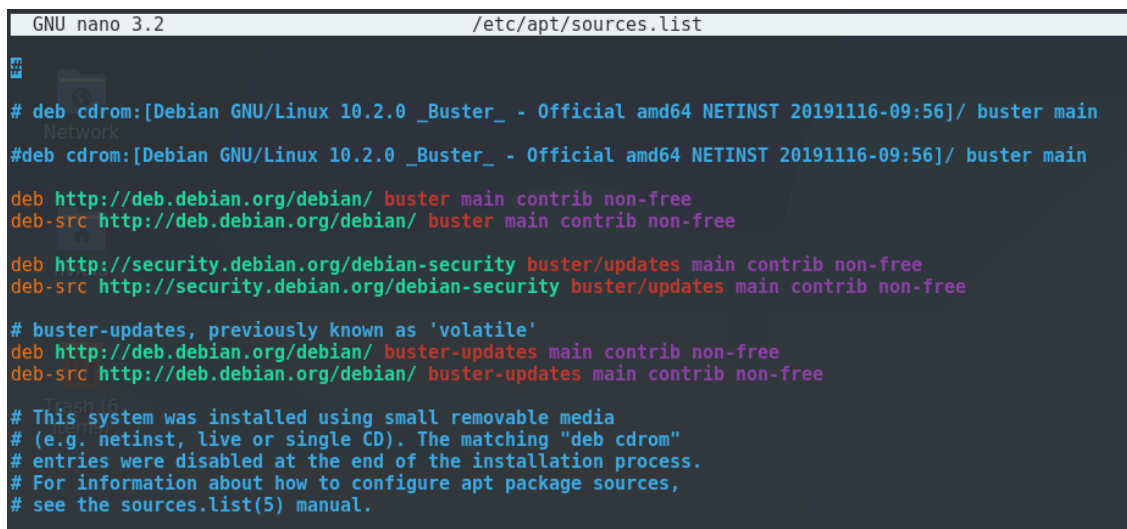
The steps here are intended to help you setup and run Jupyter notebook with the Google Colab notebooks. In order to run these, you will need a new-ish nVidia GPU, a CPU with AVX and SSE4.x, ~30Gb of RAM and/or swap, and a Linux install. In this tutorial, I will be using Debian.

You will first need to download the contents of [Synthbot's soundtools](#) folder somewhere onto your computer, or at least the files you intend to work with.

You will need to install the official nVidia drivers. By default, Debian comes with Nouveau drivers. To enable you to install them, you must add the contributor and non-free repos to your sources.list file. In this tutorial I will be using Nano as my text editor, feel free to use whatever you'd like. Run the following command as root.

nano /etc/apt/sources.list

At the end of each entry, add "contrib non-free".

Save with Ctrl+O and exit with Ctrl+X. Then run:

apt-get update

You can now install the nVidia utility to determine which driver you should install (typically nvidia-driver).

apt-get install nvidia-detect

The output should look similar to this:


```
root@debian2:/home/nvidia# nvidia-detect
Detected NVIDIA GPUs:
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GM206 [GeForce GTX 960] [10de:1401] (rev
a1)

Checking card:  NVIDIA Corporation GM206 [GeForce GTX 960] (rev a1)
Your card is supported by the default drivers and legacy driver series 390.
It is recommended to install the
    nvidia-driver
package.
```

Then install the recommended driver with apt-get. For "nvidia-driver":

apt-get install nvidia-driver

Running the install will likely bring up some warnings. Reboot your computer when done.

You can verify that you are running the official drivers with the lshw tool. Install and run it with the following:

apt-get install lshw
lshw -C display

If you have successfully installed the driver, it should look like this. The relevant bit of information is boxed in red. Should say "nvidia" for the driver.

```
*-display
    description: VGA compatible controller
    product: GM206 [GeForce GTX 960]
    vendor: NVIDIA Corporation
    physical id: 0
    bus info: pci@0000:01:00.0
    version: a1
    width: 64 bits
    clock: 33MHz
    capabilities: pm msi pciexpress vga_controller bus_master cap_list rom
    configuration: driver=nvidia latency=0
    resources: iomemory:80-7f iomemory:80-7f irq:44 memory:c0000000-c0ffffff memory:800000000-80fff
ffff memory:810000000-811ffffff ioport:d000(size=128) memory:c1080000-c10fffff
root@debian2:/home/nvidia#
```

Python3 should come preinstalled on Debian. You can check what version you have with the following.

python3 --version

Now you can install all the things needed to run the notebooks. This list was accurate for the previous version of the 48KHz MMI training script. If things don't work, check the error messages in Jupyter. Install the items that the messages complain about.

apt-get install python3-pip nvidia-smi git python-pip curl nvidia-cuda-toolkit
pip3 install jupyter matplotlib librosa tqdm torch unidecode inflect tensorboardX tensorflow

This next step is optional. If you would like to access your Jupyter instance from another computer you can do the following.

jupyter notebook --generate-config
nano /YourAccountHere/.jupyter/jupyter_notebook_config.py

The output of the Jupyter notebook --generate-config option should tell you the file path for the configuration file you want to edit. The edit the following lines:

Change: "#c.NotebookApp.allow_origin = ''" to "c.NotebookApp.allow_origin = '*'"
Change "#c.NotebookApp.ip = 'localhost'" to "c.NotebookApp.ip = '0.0.0.0'"

To get your IP address for access on another computer, run:

hostname -I

Now run the Jupyter notebook. If you want to run it as root, can do so with the --allow-root option.

jupyter notebook

Follow the link it provides you to get access to the web client. You can set a password for easier access in the future.

Now you can open your Colab notebook in jupyter (in .ipnb format). It will probably give a validation error whenever it saves. To get rid of (or at least postpone) this annoyance you can change the autosave frequency to something less frequent. Make a new cell at the top of the notebook and put:

%autosave 18400

You can change 18400 to whatever you'd like. This is a time in seconds, 18400 is approximately a day. Just remember to manually save your notebook when you make changes.

Now you will need to make some changes to your notebook. Redirect all file paths to local ones. You can also get rid of the Google Drive mount as you will be running things locally.

There is a bit of an issue when running importing matplotlib and torch. If you have an error where torch will not recognize your installed nVidia drivers, move the import of matplotlib before the import of torch. This will fix the issue (source: some internet forum yet to be linked). An example of doing this is shown below.

```
In [*]: %matplotlib inline
        import os
        if os.getcwd() != '/home/nvidia/tacotron2':
            os.chdir('tacotron2')
        import time
        import argparse
        import numpy as np
        import gc
        from numpy import finfo

        import matplotlib.pylab as plt

        import torch
        from distributed import apply_gradient_allreduce
        import torch.distributed as dist
        from torch.utils.data.distributed import DistributedSampler
        from torch.utils.data import DataLoader

        from model import Tacotron2
        from data_utils import TextMelLoader, TextMelCollate
        from loss_function import Tacotron2Loss
        from logger import Tacotron2Logger
        from hparams import create_hparams
        from utils import to_gpu
        import random
        import time
        from math import e

        from tqdm import tqdm_notebook as tqdm
        import layers
        from utils import load_wav_to_torch, load_filepaths_and_text
        import gradient_adaptive_factor
        from text import text_to_sequence
        from distutils.dir_util import copy_tree
        #import matplotlib.pylab as plt
        from scipy.io.wavfile import read
```
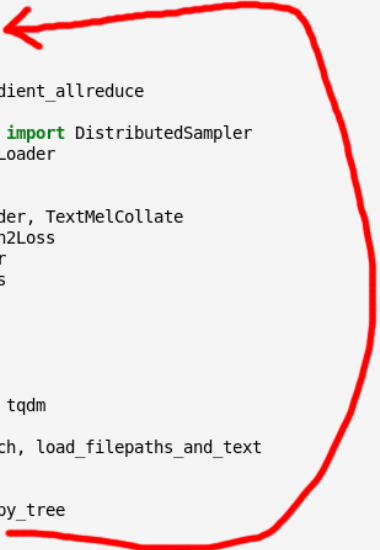
Another bit of troubleshooting that may be necessary, if you find yourself with an "RuntimeError: cuda runtime error (999) : unknown error" it's an issue with the nVidia driver. Fix it with:

sudo rmmod nvidia_uvm

sudo rmmod nvidia

sudo modprobe nvidia

sudo modprobe nvidia_uvm

Source: Stack Overflow

From there you can basically use it just as you would on Google Colab. An example of a modified notebook is linked below. This is an older version of the 48KHz MMI notebook with the soundtools folder located in /home/YourAccount. Change "YourAccountHere" to the name of your account if you intend to use it. You will also likely need to lower the batch size in order to fit into your VRAM (I use 12 for 4Gb and 16 for 6Gb, can mess around to see what'll work). Link.

One recommendation I have if you are using the 48KHz MMI training notebook, keep an eye on your RAM/swap usage with some kind of resource monitor. If you encounter the memory leak, restart the kernel just as you would in Colab. I use htop. Install and run with:

apt-get install htop
htop

Training

Training 48KHz MMI Models

This section of the doc is intended to give guidance on usage of Cookie's 48KHz MMI training notebook. To begin, open the notebook here.

The first thing you'll need to do is open the notebook in the playground. Click the button in the upper left to do so.

On the first code block you run, you will be prompted by Google whether or not you wish to run the notebook. Select "run anyway".

The first code block that you will run will check to see what GPU Google has assigned you. Ideally what you'll want is a P100, however you may be assigned a lesser GPU depending on what Google has available and how much you have used the service recently. Once you've run

the block it should look something like what is shown below. The GPU you have been assigned has been boxed in red in the picture.

```
!nvidia-smi -L
!nvidia-smi
# GPU 0: Tesla P100-PCIE-16GB = Good
# GPU 0: Tesla V100-PCIE-16GB = Amazing
# T4 = Slow
# P4 = Slow
# K80 = Slow

GPU 0: Tesla K80 (UUID: GPU-fc98ef95-0706-908e-3a6f-4ed148c952cb)
Sat Mar 14 06:54:08 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1      |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P8    26W / 149W |      0MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

If you have received a lesser GPU and would like to try for a better one, factory reset your runtime and run the block again. Repeat until you are satisfied with the GPU assigned. Note that you may need to wait a while before reconnecting in order to get something different.

Next the script will need to mount your GDrive. This is to allow the notebook to both read files (your training data) and save files (your model) to your GDrive. Once you've run the block, it will give you a URL to generate an access token. Follow the link and give permission. Copy the token it gives you and paste it into the box where indicated.

The next code block setups up TacoTron2 and its dependencies on your Google Colab machine.

```
import os
!git clone -q https://github.com/CookiePPP/tacotron2
os.chdir('tacotron2')
!git submodule init
!git submodule update
!pip install -q unidecode tensorboardX
```

```
Submodule 'waveglow' (https://github.com/NVIDIA/waveglow) registered for path 'waveglow'
Cloning into '/content/tacotron2/waveglow'...
Submodule path 'waveglow': checked out '4b1001fa3336a1184b8293745bb89b177457f09b'
|████████████████████████| 245kB 2.8MB/s
|████████████████████████| 204kB 9.2MB/s
```

The next section is for loading in your own data. If you are using the preprocessed pony data, run the code block below and move on to the section after.

```
data_path = 'wavs'
!mkdir {data_path}
!mkdir {data_path+"/out"}
```

If you have preprocessed your data with Synthbot's tools, redirect archive_fn (boxed in red below) to the location of the tar file on your GDrive. After that, proceed as you would for preprocessed pony data.

https://colab.research.google.com/drive/1hiFHCyS_YNJVMnsvzrJq8XYjshRg1c5D?usp=sharing

```
[ ]  #=== load the repo and data (Thanks Synthbot) ===
     !apt -qq install -y sox
     !git clone "https://github.com/synthbot-anon/synthbot.git" /content/synthbot
     !(cd /content/synthbot; git checkout experimental)
     !pip install "librosa==0.7.1" pysoundfile
     import sys
     sys.path.append('/content/synthbot/src')
     from ponysynth.corpus import ClipperArchive, phoneme_transcription
     import librosa
     import subprocess
     from tqdm import tqdm_notebook as tqdm
     archive_fn = '/content/drive/My Drive/soundtools/data/audio-tar/'+str(Pony)+'.tar'
     archive = ClipperArchive(archive_fn)

     #=== output the data in NVIDIA/Tacotron2's required format
     # Note that NVIDIA/Tacotron2 doesn't seem to use the test set, so this only
     # creates the training and validation sets.
     !mkdir {data_path}

     allowed_emotions = [x.lower() for x in allowed_emotions]
     !mkdir {data_path}
     !mkdir {data_path+"/out"}
     all_clips = []; all_clips_arpa = []; skipped_count=0; too_short_count=0; emotion_skip=0
     for key in tqdm(archive.keys()): # write the audio files for processing in bash terminal
       audio = archive.read_audio(key)
       audio_fn = '{}/{}.wav'.format(data_path, key)
       audio_fn_ = '{}/{}.wav'.format(data_path+"/out", key)
       with open(audio_fn, 'wb') as audio_out:
         audio_out.write(audio.read())
```

If you have only audio files and text files, setup your filelist.txt like [this](#). Then run the block below.

```
data_path = 'wavs'
!mkdir {data_path}
!mkdir {data_path+"/out"}

mkdir: cannot create directory 'wavs': File exists
mkdir: cannot create directory 'wavs/out': File exists
```

Upload your filelist.txt to tacotron2/filelists.
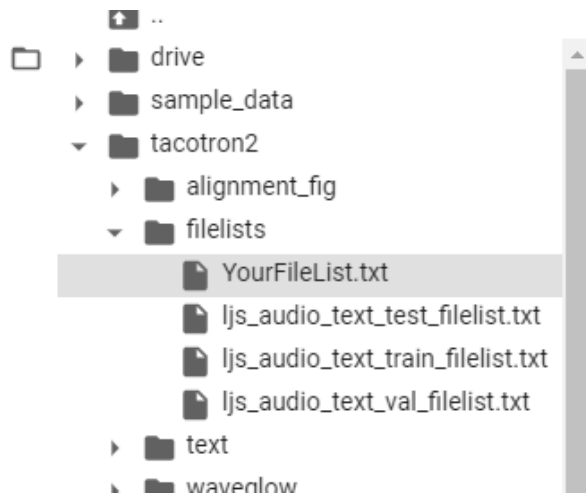
Upload your audio files to /tacotron2/wavs/out/. Should look similar to below.
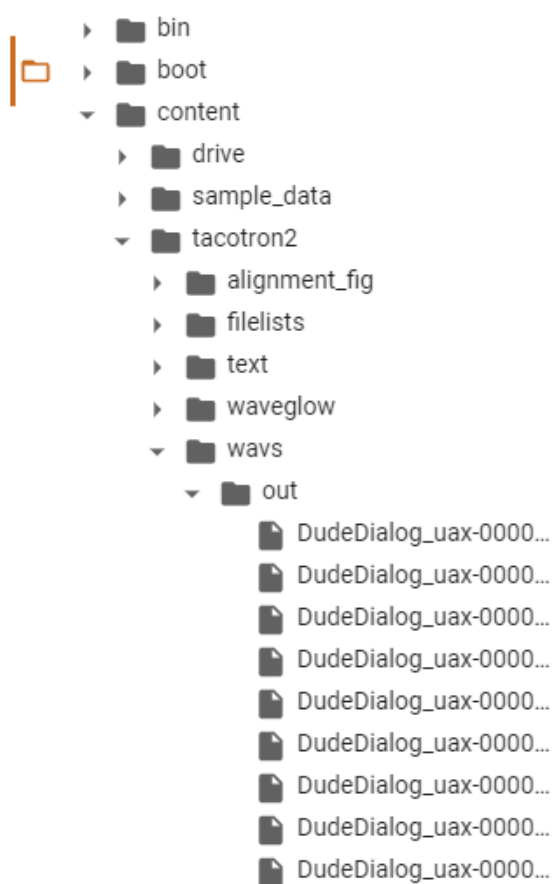
Next is where you can select what character you want to train. If you are using your own data, skip this section. Change the pony name where indicated. By default, the notebook will skip lines marked as noisy. If you would like to include noisy lines, change skip_noisy to false. The amount of data used for training vs validation is set by percentage_training_data. You can adjust

it by changing the percent here. You can also adjust what emotions will be included in the training dataset. Simply remove emotions from the list that you don't want included.

```python
# Options for Dataloading
Pony = 'Twilight-Sparkle' # <-- Pick a voice e.g 'Rainbow-Dash' or 'Rarity'

skip_noisy = True # Disable to train with Noisy Data included
percentage_training_data = 0.95 # 95% of Data will be used for training, 5% for Validation
allowed_emotions = """
Anxious
Angry
Annoyed
Amused
Confused
Crazy
Disgust
Exhausted
Fear
Happy
Neutral
Sad
Serious
Singing
Shouting
Surprised
Smug
Love
Sarcastic
Tired
Whispering
Whining
""".split("\n")[1:-1]
```

The next block load's Synthbot's repo and your training data.

```python
#=== load the repo and data (Thanks Synthbot) ===
!apt -qq install -y sox
!git clone "https://github.com/synthbot-anon/synthbot.git" /content/synthbot
!(cd /content/synthbot; git checkout experimental)
!pip install "librosa==0.7.1" pysoundfile
import sys
sys.path.append('/content/synthbot/src')
from ponysynth.corpus import ClipperArchive, phoneme_transcription
import librosa
import subprocess
from tqdm import tqdm_notebook as tqdm
archive_fn = '/content/drive/My Drive/soundtools/data/audio-tar/'+str(Pony)+'.tar'
archive = ClipperArchive(archive_fn)

#=== output the data in NVIDIA/Tacotron2's required format
# Note that NVIDIA/Tacotron2 doesn't seem to use the test set, so this only
# creates the training and validation sets.
!mkdir {data_path}

allowed_emotions = [x.lower() for x in allowed_emotions]
!mkdir {data_path}
!mkdir {data_path+"/out"}
all_clips = []; all_clips_arpa = []; skipped_count=0; too_short_count=0; emotion_skip=0
for key in tqdm(archive.keys()): # write the audio files for processing in bash terminal
    audio = archive.read_audio(key)
    audio_fn = '{}/{}.wav'.format(data_path, key)
    audio_fn_ = '{}/{}.wav'.format(data_path+"/out", key)
    with open(audio_fn, 'wb') as audio_out:
        audio_out.write(audio.read())
```

The audio clips are then cleaned.

```bash
%%script bash
# trim all 48Khz files
mkdir /content/tacotron2/wavs/out
cd /content/tacotron2/wavs;
for input in *.wav; do
  output="out/$input"
  sox "$input" "$output" silence 1 0.05 0.1% reverse silence 1 0.05 0.1% reverse;
done
```

```
mkdir: cannot create directory '/content/tacotron2/wavs/out': File exists
```

Then some final preparations with the data are made.

```python
should_continue = 0 # should run "continue" command inside outer loop.
for key in tqdm(archive.keys()):
  label = archive.read_label(key)
  audio_fn = '{}/{}.wav'.format(data_path, key)
  audio_fn_ = '{}/{}.wav'.format(data_path+"/out", key)
  if (label['noise'] in ['Very Noisy','Noisy']) and skip_noisy: os.remove(audio_fn_); skipped_count+=1; continue

  for tag in label['tags']:
    if tag.lower() not in allowed_emotions:
      try: os.remove(audio_fn_)
      except: pass
      print(tag+" emotion not in list"); emotion_skip+=1; should_continue = 1; break # this is supposed to break the outer loop
  if should_continue: should_continue = 0; continue

  audio = archive.read_audio(key)
  transcript = label['utterance']['content']
```

Code for TacoTron2 training.

The code for Tacotron2 training is under this block, just run this block.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

You can double click if you're interested in the code

Set your model filename here. Be aware of what you have in your colab/ourdir folder and if the file already exists, the notebook will resume training from it.

```
model_filename = 'mmi_twilight_test'
```

This next section sets the training and validation file lists. Only modify if using your own data.

```
[15] hparams.training_files = "filelists/clipper_train_filelist.txt"
     hparams.validation_files = "filelists/clipper_val_filelist.txt"
```
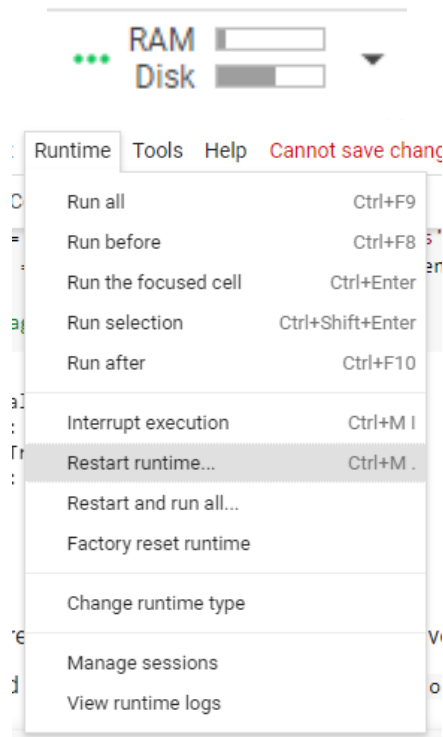
The next block contains a great many number of parameters that can be tuned. If you are looking to tune your model, Cookie gives some suggestions on where to start in the area above the code block. Check the comments to see the function of each or see the parameters guide in doc. If you are starting out, best to stick with the defaults.

```
# hparams to Tune
gradient_adaptive_factor.UPDATE_GAF_EVERY_N_STEP = 10
hparams.use_mmi=True
hparams.use_gaf=True
hparams.max_gaf=0.3
hparams.drop_frame_rate = 0.0
hparams.p_teacher_forcing=1.0 # not used in this notebook

# Dropout                       # https://pytorch.org/assets/images/ta
hparams.p_attention_dropout=0.1
hparams.p_decoder_dropout=0.1

# Learning Rate                 # https://www.desmos.com/calculator/ptgc
decay_start = 15000             # wait till decay_start to start decaying
hparams.A_ = 10e-4                  # Start/Max Learning Rate
hparams.B_ = 12000                  # Decay Rate
hparams.C_ = 0                      # Shift learning rate equation (up
min_learning_rate = 1e-5  # Min Learning Rate
```

Finally you can start training. Do note that this notebook has a memory leak. Pay attention to the notes below and in the Colab notebook.

```
if not os.path.exists("mels_finished_generating"):
    create_mels(); gc.collect()
train(output_directory, log_directory, checkpoint_path,
    warm_start, n_gpus, rank, group_name, hparams, log_directory2)
```

If your notebook is using more than 24GB of RAM **after** generating mels, you have a memory leak. Restart the kernel and try again until it's running stable with reasonable RAM usage. Check RAM usage in the upper right hand corner.

Runtime   Tools   Help   Cannot save chanç

Run all	Ctrl+F9
Run before	Ctrl+F8
Run the focused cell	Ctrl+Enter
Run selection	Ctrl+Shift+Enter
Run after	Ctrl+F10
Interrupt execution	Ctrl+M I
Restart runtime...	Ctrl+M .
Restart and run all...	
Factory reset runtime	
Change runtime type	
Manage sessions	
View runtime logs	

Once everything's running, it should look like the following.

Epoch: 506 Validation loss 113092: 0.200042 Time: 13.2m LR: 0.000010
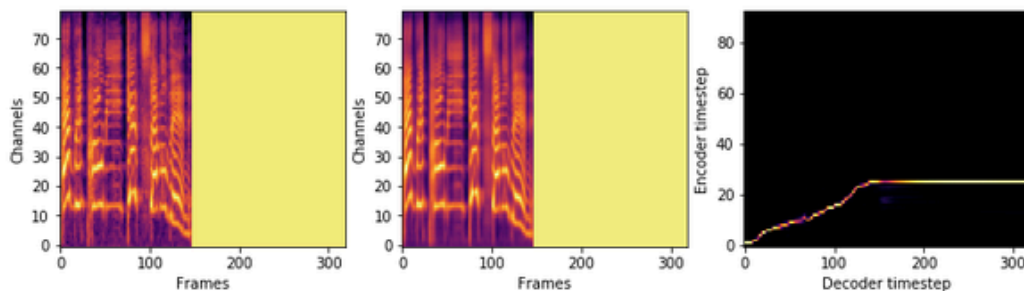
Saving model and optimizer state at iteration 113092 to /home/nvidia/col
ab/outdir/mmi_pinkie_test

Starting Epoch: 507 Iteration: 113092

113314 [Train_loss 0.2042] [mi_loss 0.0060] [Grad Norm 0.1371] [gaf 0.1072] [LR 0.000010]: 100%

Iterations are a measure of how much the model has been trained. The validation loss is a measure of well the model can predict the proper output. Essentially you want to get the loss as low as possible. This happens with an increased amount of training. However, be aware that after a certain point the model may start to overfit and the loss will increase. At this point
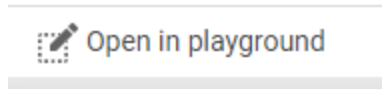
additional training will not provide benefit. Overfitting will occur sooner on datasets with less audio.

Training 22KHz Models

This section of the doc is intended to give guidance on usage of Cookie's 22KHz training notebook. To begin, open the notebook [here](#).

Note that in most cases this 22KHz notebook has been superseded by the 48KHz MMI version.

The first thing you'll need to do is open the notebook in the playground. Click the button in the upper left to do so.

On the first code block you run, you will be prompted by Google whether or not you wish to run the notebook. Select "run anyway".

The first code block that you will run will check to see what GPU Google has assigned you. Ideally what you'll want is a P100, however you may be assigned a lesser GPU depending on what Google has available and how much you have used the service recently. Once you've run the block it should look something like what is shown below. The GPU you have been assigned has been boxed in red in the picture.
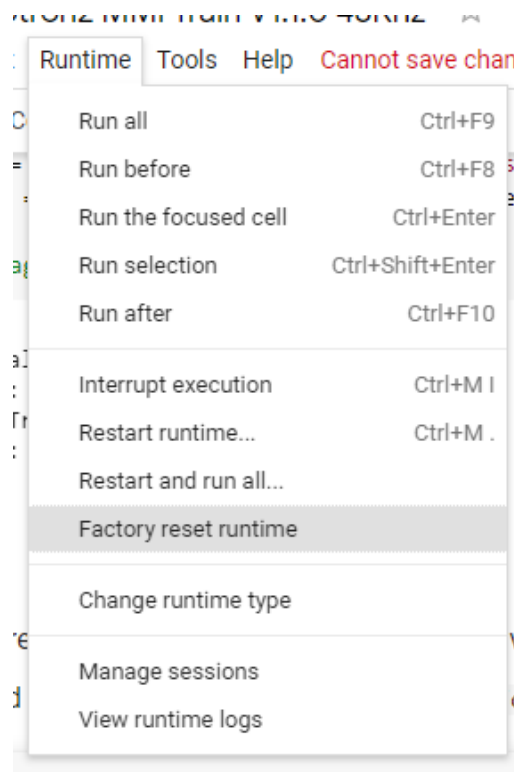
```
!nvidia-smi -L
!nvidia-smi
# GPU 0: Tesla P100-PCIE-16GB = Good
# GPU 0: Tesla V100-PCIE-16GB = Amazing
# T4 = Slow
# P4 = Slow
# K80 = Slow
```

```
GPU 0: Tesla K80 (UUID: GPU-fc98ef95-0706-908e-3a6f-4ed148c952cb)
Sat Mar 14 06:54:08 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   40C    P8    26W / 149W |      0MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

If you have received a lesser GPU and would like to try for a better one, factory reset your runtime and run the block again. Repeat until you are satisfied with the GPU assigned. Note that you may need to wait a while before reconnecting in order to get something different.

Runtime	Tools	Help	Cannot save chan
Run all			Ctrl+F9
Run before			Ctrl+F8
Run the focused cell			Ctrl+Enter
Run selection			Ctrl+Shift+Enter
Run after			Ctrl+F10
Interrupt execution			Ctrl+M I
Restart runtime...			Ctrl+M .
Restart and run all...			
Factory reset runtime			
Change runtime type			
Manage sessions			
View runtime logs			

Next the script will need to mount your GDrive. This is to allow the notebook to both read files (your training data) and save files (your model) to your GDrive. Once you've run the block, it will give you a URL to generate an access token. Follow the link and give permission. Copy the token it gives you and paste it into the box where indicated.

```
#Google Drive Authentication Token
from google.colab import drive
drive.mount('drive')

Go to this URL in a browser: https://accounts.google.com/o/oaut

Enter your authorization code:
..........
Mounted at drive
```

The next code block setups up TacoTron2 and its dependencies on your Google Colab machine.

```
import os
!git clone -q https://github.com/CookiePPP/tacotron2
os.chdir('tacotron2')
!git submodule init
!git submodule update
!pip install -q unidecode tensorboardX

Submodule 'waveglow' (https://github.com/NVIDIA/waveglow) registered for path 'waveglow'
Cloning into '/content/tacotron2/waveglow'...
Submodule path 'waveglow': checked out '4b1001fa3336a1184b8293745bb89b177457f09b'
|█████████████████████████████| 245kB 2.8MB/s
|█████████████████████████████| 204kB 9.2MB/s
```

The next section is for loading in your own data. If you are using the preprocessed pony data, run the code block below and move on to the section after.

```
data_path = 'wavs'
!mkdir {data_path}
!mkdir {data_path+"/out"}
```

If you have preprocessed your data with Synthbot's tools, redirect archive_fn (boxed in red below) to the location of the tar file on your GDrive. After that, proceed as you would for preprocessed pony data.

```
[ ]   # Options for Dataloading
      archive_fn = '/content/drive/My Drive/soundtools/data/audio-trimmed-22khz/Celestia.tar' # <-- Pick a voice

      allowed_emotions = ['Whining','Disgust','Neutral','Anxious','Happy','Angry','Crazy','Annoyed','Serious','Amuse
      skip_noisy = True # Only disable if 100% sure
      percentage_training_data = 0.95 # 95% of Data will be used training, 5% for Validation

      #=== load the repo and data (Thanks Synthbot) ===
      !apt -qq install -y sox
      !git clone "https://github.com/synthbot-anon/synthbot.git" /content/synthbot
      !(cd /content/synthbot; git checkout experimental)
      !pip install "librosa==0.7.1" pysoundfile
      import sys
      sys.path.append('/content/synthbot/src')
      from ponysynth.corpus import ClipperArchive, phoneme_transcription
      import librosa
      import subprocess
      from tqdm import tqdm_notebook as tqdm
      archive = ClipperArchive(archive_fn)
```
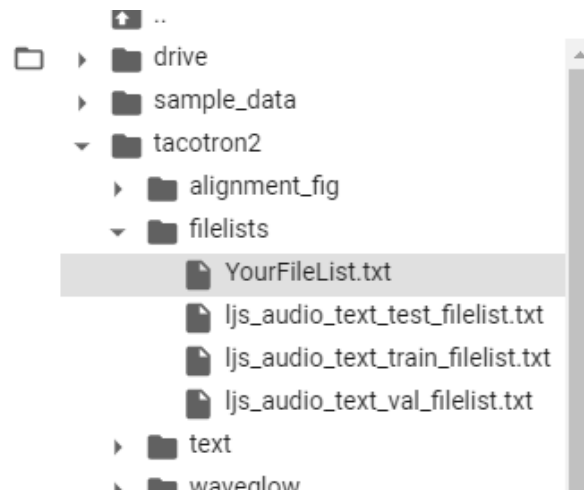
If you have only audio files and text files, setup your filelist.txt like [this](#). Then run the block below.

```
    data_path = 'wavs'
    !mkdir {data_path}
    !mkdir {data_path+"/out"}

    mkdir: cannot create directory 'wavs': File exists
    mkdir: cannot create directory 'wavs/out': File exists
```

Upload your filelist.txt to tacotron2/filelists.

Upload your audio files to /tacotron2/wavs/. Should look similar to below.

Next is where you can select what character you want to train. If you are using your own data, skip this section. Change the pony name where indicated. By default, the notebook will skip lines marked as noisy. If you would like to include noisy lines, change skip_noisy to false. The amount of data used for training vs validation is set by percentage_training_data. You can adjust it by changing the percent here. You can also adjust what emotions will be included in the training dataset. Simply remove emotions from the list that you don't want included.

```
[ ]  # Options for Dataloading
     archive_fn = '/content/drive/My Drive/soundtools/data/audio-trimmed-22khz/Celestia.tar' # <-- Pick a voice

     allowed_emotions = ['Whining','Disgust','Neutral','Anxious','Happy','Angry','Crazy','Annoyed','Serious','Amuse
     skip_noisy = True # Only disable if 100% sure
     percentage_training_data = 0.95 # 95% of Data will be used training, 5% for Validation

     #=== load the repo and data (Thanks Synthbot) ===
     !apt -qq install -y sox
     !git clone "https://github.com/synthbot-anon/synthbot.git" /content/synthbot
     !(cd /content/synthbot; git checkout experimental)
     !pip install "librosa==0.7.1" pysoundfile
     import sys
     sys.path.append('/content/synthbot/src')
     from ponysynth.corpus import ClipperArchive, phoneme_transcription
     import librosa
     import subprocess
     from tqdm import tqdm_notebook as tqdm
     archive = ClipperArchive(archive_fn)
```

Code for TacoTron2 training.

```
[ ] #@title The code for Tacotron2 training is under this block, just run t

    #@markdown THIS IS A BLOCK OF CODE. RUN ME.

    #@markdown THIS IS A BLOCK OF CODE. RUN ME.

    #@markdown THIS IS A BLOCK OF CODE. RUN ME.

    #@markdown THIS IS A BLOCK OF CODE. RUN ME.

    #@markdown THIS IS A BLOCK OF CODE. RUN ME.

    #@markdown You can double click if you're interested in the code
    %matplotlib inline
    import os
    if os.getcwd() != '/content/tacotron2':
        os.chdir('tacotron2')
    import time
    import argparse
```

The code for Tacotron2 training is under this block, just run this block.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

THIS IS A BLOCK OF CODE. RUN ME.

You can double click if you're interested in the code

Set your model filename here. Be aware of what you have in your colab/ourdir folder and if the file already exists, the notebook will resume training from it.

```
[ ]  model_filename = 'current_model'
```

This next section sets the training and validation file lists. Only modify if using your own data.

```
[15]  hparams.training_files = "filelists/clipper_train_filelist.txt"
      hparams.validation_files = "filelists/clipper_val_filelist.txt"
```

The next block contains a great many number of parameters that can be tuned. If you are looking to tune your model, Cookie gives some suggestions on where to start in the area above the code block. Check the comments to see the function of each or see the parameters guide in doc. If you are starting out, best to stick with the defaults.

```
[ ]  # hparams to Tune
     #hparams.use_mmi=True,          # not used in this notebook
     #hparams.use_gaf=True,          # not used in this notebook
     #hparams.max_gaf=0.5,           # not used in this notebook
     #hparams.drop_frame_rate = 0.2  # not used in this notebook
     hparams.p_attention_dropout=0.1
     hparams.p_decoder_dropout=0.1

     # Learning Rate                 # https://www.desmos.com/calculator/ptgcz4v
     hparams.decay_start = 15000             # wait till decay_start to start de
     hparams.A_  = 5e-4                      # Start/Max Learning Rate
     hparams.B_  = 8000                      # Decay Rate
     hparams.C_  = 0                         # Shift learning rate equation by t
     hparams.min_learning_rate = 1e-5        # Min Learning Rate
```

Generate the mels.

```
[ ]  if generate_mels:
         create_mels()
```

Check data.

```
[ ]  check_dataset(hparams)
```

Finally, start training.

```
[ ]  print('FP16 Run:', hparams.fp16_run)
     print('Dynamic Loss Scaling:', hparams.dynamic_loss_scaling)
     print('Distributed Run:', hparams.distributed_run)
     print('cuDNN Enabled:', hparams.cudnn_enabled)
     print('cuDNN Benchmark:', hparams.cudnn_benchmark)
     train(output_directory, log_directory, checkpoint_path,
           warm_start, n_gpus, rank, group_name, hparams, log_directory2)
```

Once everything's running, it should look like the following.
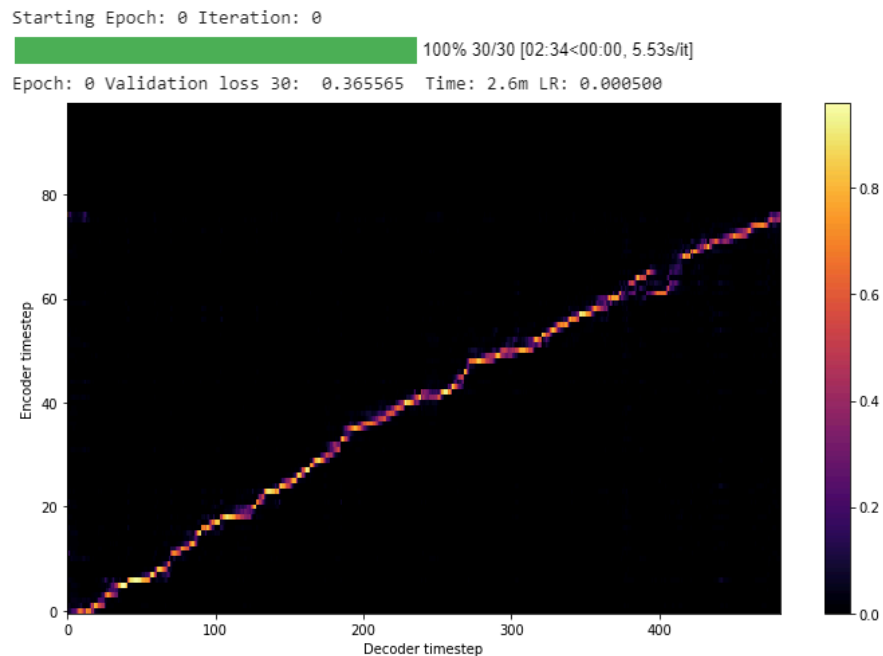
Iterations are a measure of how much the model has been trained. The validation loss is a measure of how well the model can predict the proper output. Essentially you want to get the loss as low as possible. This happens with an increased amount of training. However, be aware that after a certain point the model may start to overfit and the loss will increase. At this point additional training will not provide benefit. Overfitting will occur sooner on datasets with less audio.

HParams

Under construction.

"hparams.use_mmi" enables or disables the use of MMI (Maximizing Mutual Information). This parameter is currently marked as experimental.

"hparams.use_gaf" enables or disables GAF (Gradient Adaptive Factor). This parameter is currently marked as experimental.

"hparams.max_gaf" sets the maximum value of the GAF. This parameter is currently marked as experimental.

"hparam.drop_frame_rate" This parameter is currently marked as experimental.

"hparams.p_attention_dropout"

"hparams.p_decoder_dropout"

"hparams.decay_start" The learning rate of the model will be decreased after this number.

"hparams.A_" sets the initial and maximum learning rate of the model.

"hparams.B_" sets the decay rate of the learning speed after "decay_start" has been reached.

"hparmas.C_" shifts the learning rate equation by this much.

"hparams.min_learning_rate" sets the minimum learning rate.

"model_filename" sets the filename of the model in training.

"generate_mels" sets whether or not to generate mel spectrograms. Will be gone next version.

"hparams.show_alignments" sets whether or not to display alignment graphs during training.

"alignment_graph_height" sets the height of the displayed alignment graph.

"alignment_graph_width" sets the width of the displayed alignment graph.

"hparams.batch_size" controls how many audio files are processed by the GPU at the same time. It increases training speed but is limited by how much VRAM the GPU has. For a P100 from Google Colab, probably just leave it at default.

"hparams.load_mel_from_disk" should never need to change.

"hparams.training_files" should never need to change.

"hparams.validation_files" should never need to change.

"hparams.ignore_layers" should never need to change.

"hparams.epochs" sets number of epochs to run.

TalkNet

Colab Notebook:

https://colab.research.google.com/drive/1Nb8TWjUBJIVg7QtIazMl64PAY4-Qznzl?usp=sharing

DiffS-VC

[Unengaged]

Google doc - [Link](#)

Training Colab (beta 44.1kHz training & 24kHz) - [Link](#)

TS DiffSVC

[Link](#)

So-Vits-SVC 4.0

[WIP]

~~Link (docs) - So-Vits-SVC Github doc~~
Main Github (4.0) - [Link](#)

Github (3.0) - [Link](#)

English training colab (v4.0) - [Link](#)

English training colab (v3.0) - [Link](#)

Original Chinese training colab (v3.0) - [Link](#)

Video tutorial (it presents the 3.0 Colab but the process is same as 4.0) - [Link](#) - [BackupLink](#)

Video tutorial 4.0 (semi-offline) - [Video](#) | [File1](#) | [File2](#)

RVC (wip)

#wip

[Video (local)](#) * |

* As some Anon happen to have problem with following the standard installation with the provided github installation file, here is a presentation of the diy fix for it ([Link](#))

conda create --name RVC1006Nvidia python=3.9

conda activate RVC1006Nvidia

cd C:_Altts\RVC_Train\RVC1006Nvidia
pip install -r requirements.txt

>? for some reason its pissing and shitting itself about not able to get lxml module, use the below code to get it, then get back to running the requirements installation above
conda install -c conda-forge lxml

>need a brute force upgrade to the cuda+torch to make it able to "see" the gpu
pip install --upgrade torch==1.9.1+cu111 torchaudio==0.9.1 --extra-index-url https://download.pytorch.org/whl/cu111

python infer-web.py --pycmd C:\Users\User001\miniconda3\envs\RVC1006Nvidia\python.exe --port 7897

Quick fixes for some uncommon problems:

#AttributeError: module 'distutils' has no attribute 'version'
#run this installation in your Conda terminal
pip install setuptools==59.5.0

#AttributeError: module 'torch' has no attribute 'pi'
Go to the file "xxxDirectoryxxx\RVC1006Nvidia\infer\lib\infer_pack\models.py
#under all the modules import code add those two lines:
import math
torch.pi = math.pi

Synthesis

Synthesizing 48KHz MMI Models

An Anon put together a demonstration video. Credit: >>35067137

Check out the Making the Most of the AI section of the doc for tips on improving output.

This section of the doc is intended to give guidance on usage of Cookie's 48KHz MMI synthesis notebook. To begin, open the notebook here.

The first thing you'll need to do is open the notebook in the playground. Click the button in the upper left to do so.

On the first code block you run, you will be prompted by Google whether or not you wish to run the notebook. Select "run anyway".

The first code block sets up TacoTron2, WaveGlow, and the MEGA Downloader.

```
!git clone https://github.com/jeroenmeulenaar/python3-mega.git
!(cd python3-mega; pip install urlobject pycrypto)

import os
os.chdir('python3-mega')
from mega import Mega
os.chdir('../')
m = Mega.from_ephemeral()
print("Downloading Dictionary...")
m.download_from_url('https://mega.nz/#!yAMyFYCI!o_UmixbiIzosyYk-6O5xRZZDGpFRik_eMrZum-iQuhQ')
```

The next code block that you will run checks to see what GPU Google has assigned you. You want to make sure you do **not** get a k80 as they have a bug where no audio will be produced. Once you've run the block it should look something like what is shown below. The GPU you have been assigned has been boxed in red in the picture.

```
%matplotlib inline
!nvidia-smi -L
!nvidia-smi
import os
from os.path import exists, join, basename, splitext
!pip install gdown
git_repo_url = 'https://github.com/CookiePPP/tacotron2.git'
project_name = splitext(basename(git_repo_url))[0]
if not exists(project_name):
  # clone and install
  !git clone -q --recursive {git_repo_url}
  !cd {project_name}/waveglow && git checkout 2fd4e63
  !pip install -q librosa unidecode
```

```
GPU 0: Tesla P100-PCIE-16GB (UUID: GPU-837d1fb9-b7fe-a3a4-0cfa-2405711f6f05)
Sat Mar 14 09:15:14 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   41C    P0    27W / 250W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

If you have received a k80, factory reset your runtime and run the block again. Repeat until you have something other than a k80. Note that you may need to wait a while before reconnecting in order to get something different.

You will then setup the TacoTron2 model. This is where you change the model. Replace the GDrive ID as indicated.

The WaveGlow model is then downloaded and setup.

```
[4]  # Download WaveGlow Model
     waveglow_pretrained_model = 'waveglow.pt'
     waveglow_ids = ['1DMyL3RxFqAVhH60VCLnVaDt2YJb2RCfz', '10wDUQ3HAKhdNgnqZC3PBDBZ12PADZWAU', '14ajSxb4yJXQnv_nf9O89dIWc+
     while not exists(waveglow_pretrained_model) and waveglow_ids:
         id = choice(waveglow_ids)
         gdown.download(d+id, waveglow_pretrained_model, quiet=False);
         if not exists(waveglow_pretrained_model): print("Download Failed, attempting another ID"); waveglow_ids.remove(id
     if exists(waveglow_pretrained_model): print("WaveGlow Downloaded")
     else: print("WaveGlow failed to download on all ID's provided")

     # Load WaveGlow model into GPU
     waveglow = torch.load(waveglow_pretrained_model)['model']
     waveglow.cuda().eval().half()
     for k in waveglow.convinv:
         k.float()
     denoiser = Denoiser(waveglow)
     print("This WaveGlow model has been trained for ",torch.load(waveglow_pretrained_model)['iteration']," Iterations.")
```

Finally, modify this code block to contain what you want the model to say and then run it. Each new line will create a new clip.

```
text = """
You can type what you'd like the ponies to say here.
""" # https://derpicdn.net/img/2019/12/14/2220074/large.jpeg

sigma = 0.75
denoise_strength = 0.01
raw_input_ = False  # disables automatic ARPAbet conversion, useful for inputting your own pronounciation or

show_graphs = True
graph_scale = 1 # literally a zoom factor
alignment_graph_width = 1800
alignment_graph_height = 720

save_wavs = 1
counter = 0
text = unidecode(text) # convert unicode punctuation into it's normal equivalents (thanks Fimfiction.)
text = text * 1 # how many times to generate each clip
for i in text.split("\n"):
    if len(i) < 1: continue;
    print(i)
```

The output should appear below.
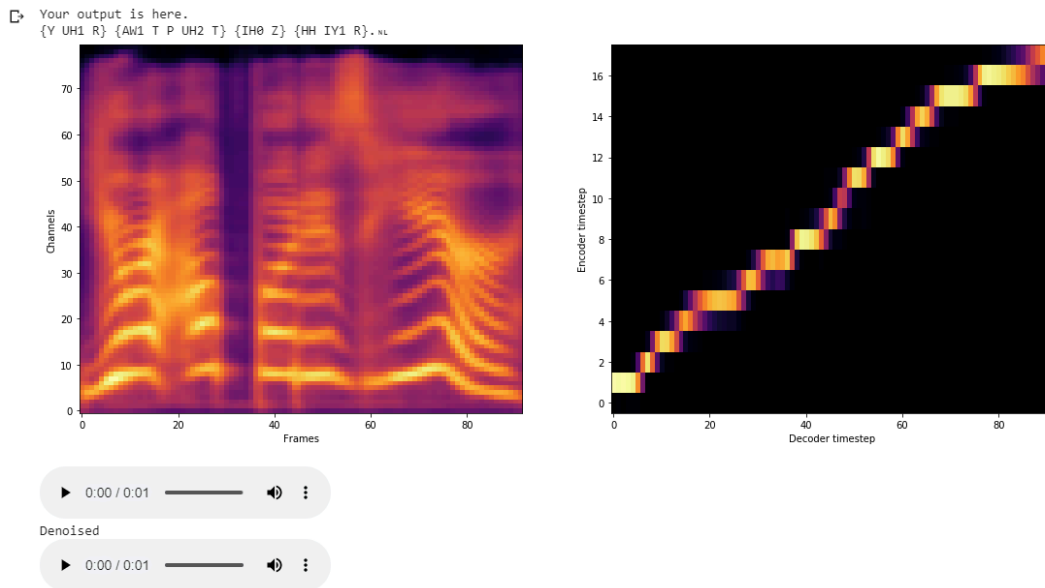
▶ 0:00 / 0:01 ━━━━━━━ 🔊 ⋮

Denoised

▶ 0:00 / 0:01 ━━━━━━━ 🔊 ⋮

Two audio clips will be generated, the first one is the raw output of the AI and the second has some denoising applied. You will probably want the denoised clip. Download with the menu icon at the right of each clip.

Synthesizing 22KHz Models

Check out the Making the Most of the AI section of the doc for tips on improving output.

This section of the doc is intended to give guidance on usage of Cookie's 22KHz synthesis notebook. To begin, open the notebook here.

Note that in most cases this 22KHz notebook has been superseded by the 48KHz MMI version.

The first thing you'll need to do is open the notebook in the playground. Click the button in the upper left to do so.

On the first code block you run, you will be prompted by Google whether or not you wish to run the notebook. Select "run anyway".

The first code block sets up TacoTron2, WaveGlow and checks what GPU Google has assigned you. You want to make sure you do **not** get a k80 as they have a bug where no audio will be produced. Once you've run the block it should look something like what is shown below. The GPU you have been assigned has been boxed in red in the picture.

```python
!nvidia-smi -L
!nvidia-smi
import os
from os.path import exists, join, basename, splitext
!pip install gdown
git_repo_url = 'https://github.com/NVIDIA/tacotron2.git'
project_name = splitext(basename(git_repo_url))[0]
if not exists(project_name):
  # clone and install
  !git clone -q --recursive {git_repo_url}
  !cd {project_name}/waveglow && git checkout 2fd4e63
  !pip install -q librosa unidecode

import sys
sys.path.append(join(project_name, 'waveglow/'))
sys.path.append(project_name)
import time
import matplotlib
import matplotlib.pylab as plt
import gdown
d = 'https://drive.google.com/uc?id='
```

```
 ☐→ GPU 0: Tesla T4 (UUID: GPU-5834c4fb-68f0-ff01-ec4c-c616cfb33a43)
    Sun Mar 15 01:59:23 2020
    +-----------------------------------------------------------------------------+
    | NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1      |
    |-------------------------------+----------------------+----------------------+
    | GPU  Name       Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
    | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
    |===============================+======================+======================|
    |   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
    | N/A   69C    P8    11W /  70W |      0MiB / 15079MiB |      0%      Default |
    +-------------------------------+----------------------+----------------------+

    +-----------------------------------------------------------------------------+
    | Processes:                                                       GPU Memory |
    |  GPU       PID    Type   Process name                            Usage      |
    |=============================================================================|
    |  No running processes found                                                 |
    +-----------------------------------------------------------------------------+
```

If you have received a k80, factory reset your runtime and run the block again. Repeat until you have something other than a k80. Note that you may need to wait a while before reconnecting in order to get something different.
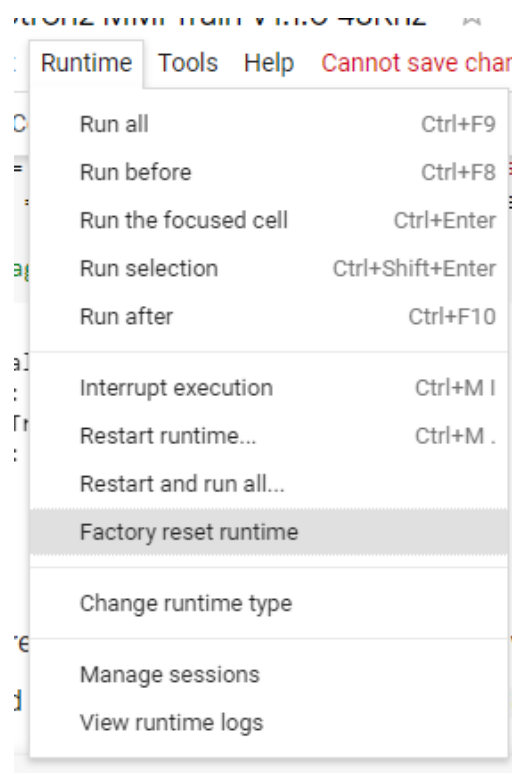
You will then download your TacoTron2 model. This is where you change the model. Replace the GDrive ID as indicated.

```
[ ] force_download_TT2 = True
    tacotron2_pretrained_model = 'MLPTTS'
    if not exists(tacotron2_pretrained_model) or force_download_TT2:
                    # ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ Change this to pick a voice.
        gdown.download(d+r 1CT4XCnrcQCapLATjRGxpYGtpWfr4xHpH , tacotron2_pretrained_model,
                    # ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ Change this to pick a voice.

    waveglow_pretrained_model = 'waveglow.pt'
    if not exists(waveglow_pretrained_model):
      gdown.download(d+r'1Rm5rV5XaWWiUbIpg538515sh68z2bVOE', waveglow_pretrained_model,
```

TacoTron2 and WaveGlow are then initialized.

```
%matplotlib inline
import IPython.display as ipd
import numpy as np
import torch

from hparams import create_hparams
from model import Tacotron2
from layers import TacotronSTFT
from audio_processing import griffin_lim
from text import text_to_sequence
from denoiser import Denoiser

graph_width = 900
graph_height = 360
def plot_data(data, figsize=(int(graph_width/100), int(graph_height/100))):
    %matplotlib inline
    fig, axes = plt.subplots(1, len(data), figsize=figsize)
    for i in range(len(data)):
        axes[i].imshow(data[i], aspect='auto', origin='bottom',
                        interpolation='none', cmap='inferno')
    fig.canvas.draw()
    plt.show()
```

TacoTron2 is loaded.

```
[4]  # Load Tacotron2
     hparams.sampling_rate = 22050
     hparams.max_decoder_steps = 2000
     hparams.gate_threshold = 0.25 # Model must be 25% sure the clip is over before ending generation
     model = Tacotron2(hparams)
     model.load_state_dict(torch.load(tacotron2_pretrained_model)['state_dict'])
     _ = model.cuda().eval().half()
```

WaveGlow is loaded.

```
# Load WaveGlow
waveglow = torch.load(waveglow_pretrained_model)['model']
waveglow.cuda().eval().half()
for k in waveglow.convinv:
    k.float()
denoiser = Denoiser(waveglow)
```

Finally, modify this code block to contain what you want the model to say and then run it. Each new line will create a new clip.

```
text = """
You can type whatever you'd like here.
""" # https://derpicdn.net/img/2019/12/14/2220074/large.jpeg
sigma = 0.75
denoise_strength = 0.01
raw_input = False # disables automatic ARPAbet conversion, useful for inpu

for i in text.split("\n"):
    if len(i) < 1: continue;
    print(i)
    if raw_input:
        if i[-1] != ";": i=i+";"
    else: i = ARPA(i)
    print(i)
    with torch.no_grad(): # save VRAM by not including gradients
        sequence = np.array(text_to_sequence(i, ['english_cleaners']))[Non
        sequence = torch.autograd.Variable(torch.from_numpy(sequence)).cud
        mel_outputs, mel_outputs_postnet, _, alignments = model.inference(
        plot_data((mel_outputs_postnet.float().data.cpu().numpy()[0],
                alignments.float().data.cpu().numpy()[0].T))
        audio = waveglow.infer(mel_outputs_postnet, sigma=sigma); print(""
        audio_denoised = denoiser(audio, strength=denoise_strength)[:, 0];
```

The output should appear below.
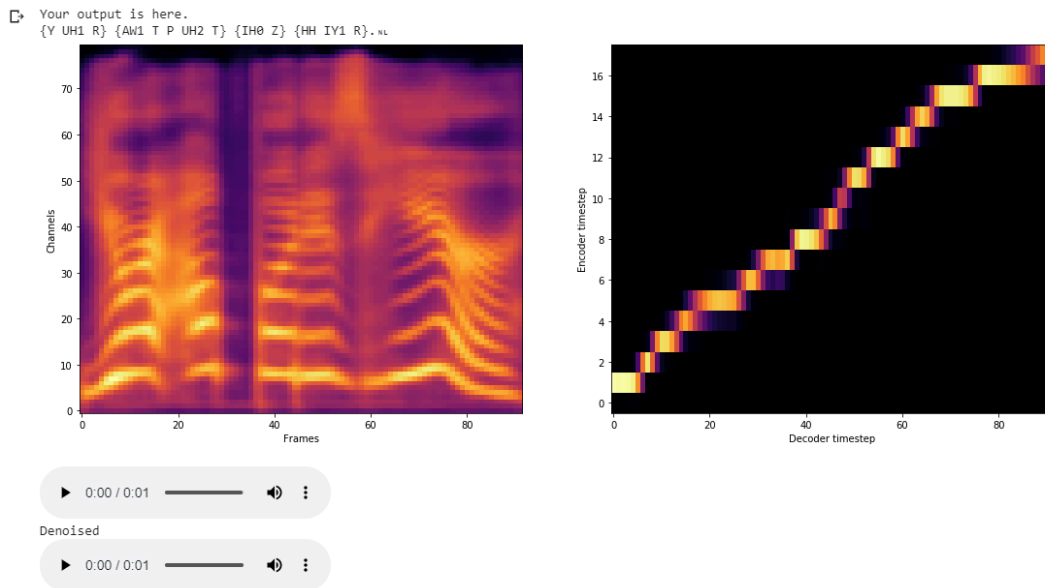
Two audio clips will be generated, the first one is the raw output of the AI and the second has some denoising applied. You will probably want the denoised clip. Download with the menu icon at the right of each clip.

Inference Server (Synthesis)

Check out the [Making the Most of the AI](#) section of the doc for tips on improving output.

This section of the doc is intended to give guidance on usage of Cookie's inference server notebook. While the notebook includes directions, this is provided as a supplement. To begin, open the notebook [here](#).

The first thing you'll need to do is open the notebook in the playground. Click the button in the upper left to do so.

On the first code block you run, you will be prompted by Google whether or not you wish to run the notebook. Select "run anyway".

Warning: This notebook was not authored by Google.

This notebook was authored by **cookietriplep@gmail.com**. It may request access to your data stored with Google, or read data and credentials from other sessions. Please review the source code before executing this notebook. Please contact the creator of this notebook at cookietriplep@gmail.com with any additional questions.

CANCEL   **RUN ANYWAY**

The first code block will let you know what GPU Google has assigned you. Ideally you will want a Tesla P100.

```
# (optional) check graphics card
!nvidia-smi -L

GPU 0: Tesla P100-PCIE-16GB (UUID: GPU-6c18045a-e438-5e21-3367-e326c6ee90a7)
```

Before you will be able to use the inference notebook, you will need to set up your Google Drive with the codedump folder linked in the notebook. Follow the link in the code block shown below.

1 - Mount Google Drive and add model shortcut

(you can also download/clone the models if you want them local/safe)

1.1 - Go to This link. ←

1.2 - Add this folder to your google drive. Right click on 'codedump' at the top, click 'Add shortcut to Drive' and add to 'My drive' [example image]

1.3 - Click "Mount Drive" in the top left (example image). Enter the credentials of the Google Account that was used to 'Add shortcut to Drive' in the previous step.

1.4 - Run the code block below and continue if there is no error.

You'll then need to add a shortcut to the folder to your Drive.
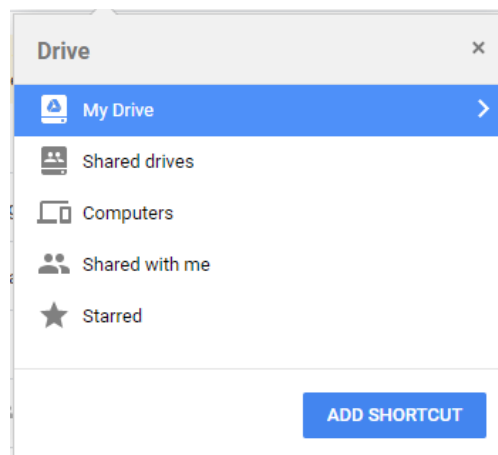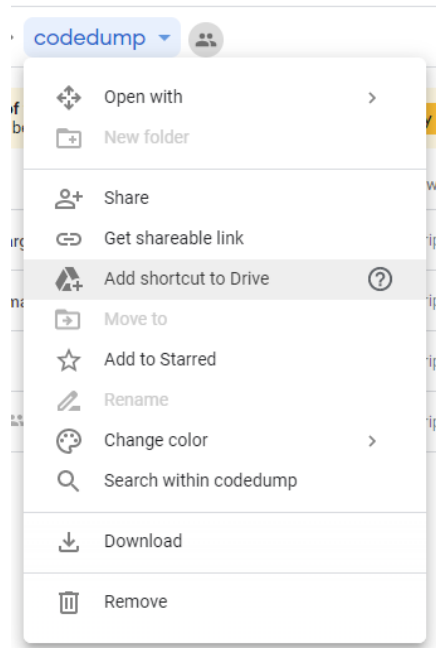
Then you must mount your Drive in Colab so that the inference server can access the files. On the left hand side of the page select the folder icon in the navigation tab (third icon down).
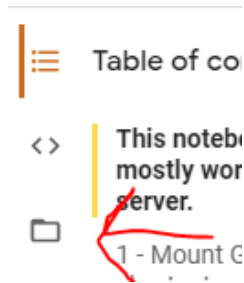
Then select "Mount Drive". It is the third icon in the list and is in the shape of a folder with the Drive logo overlaid.
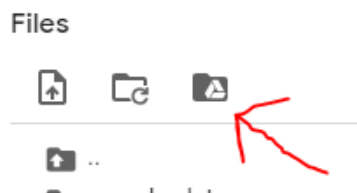
A new code block will be added to the notebook. Run the new code block, follow the link and copy the code. Paste the authentication code into the running code block and press enter.

```
[ ]   from google.colab import drive
      drive.mount('/content/drive')
```

Will become

```
from google.colab import drive
drive.mount('/content/drive')

Go to this URL in a browser: https://accounts.google.com/c

Enter your authorization code:
..........
Mounted at /content/drive
```
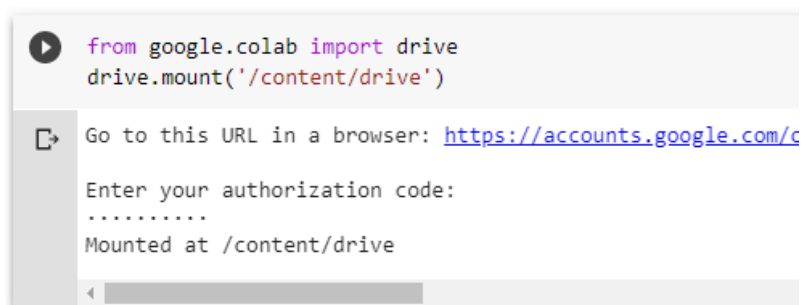
Once you have your Drive set up, running the following code block will check that everything is in order.

```
# check the shared folder has been mounted
import os
assert os.path.exists("/content/drive/My Drive/codedump"), f"codedump folder not found! ensure you have mounte
print("Assertion Passed! You're good to go.")
```

Assertion Passed! You're good to go.

The next code block will set up the code and dependencies needed to run the inference server.

```
%tensorflow_version 1.x
# clone cookie
import os
!git clone -q -b "master" https://github.com/CookiePPP/codedump.git
os.chdir('codedump')
!git submodule init
!git submodule update
!pip install -q unidecode tensorboardX emoji
os.chdir('tacotron2-PPP-1.3.0')

# clone and install nvidia/apex
!git clone -q https://github.com/NVIDIA/apex.git
!cd apex; pip install -q -v --no-cache-dir ./; cd ..

# install Sox (with all optional formats)
!apt -qq install -y libsox-fmt-all sox
```

Next, the configuration is set. Mainly file paths to the various models in the codedump folder.

```
%%writefile t2s_config.json
{
    "html_max_input_len": "9999",
    "show_inference_progress": true,
    "show_inference_alignment_scores": true,
    "localhost": false,
    "dict_path": "dictionary/merged.dict_.txt",
    "sox_output_ext": "flac",
    "working_directory": "server_infer",
    "output_directory": "server_infer_done",
    "output_maxsize_gb": 0.01,
    "tacotron": {
        "speaker_ids_file": "/content/drive/My Drive/codedump/speaker_ids.txt",
        "use_speaker_ids_file_override": true,
        "default_model": "Tacotron2 Torchmoji v0.22.1 (Large Prenet 188K)",
        "models": {
            "Tacotron2 Torchmoji v0.2 (Baseline 178K)": {
```

Finally, run the following code block to start the inference server.

```
# link through google proxy, might have firewall issues
from google.colab.output import eval_js
print("Try this link when the server has started!\n", eval_js("google.colab.kernel.

# link through ngrok, bandwidth is questionable
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip
print("\n\nOr try this ngrok link alternative!")
get_ipython().system_raw('./ngrok http 5000 &')
from time import sleep
sleep(2)
! curl -s http://localhost:4040/api/tunnels | python3 -c \
 "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

Note that this block of code should continue to run for as long as the server is running. When ready, check the block's output for the ngrok link.

```
Or try this ngrok link alternative!
https://dd226bedd7a5.ngrok.io
Starting server. Connect once 'T2S Initialized and Ready!' is shown
```

You can now open the ngrok link in a new tab to access the inference server. You can also share this link in thread to allow others to use your instance as well ^:).

Note that Google Colab has an automatic timeout. You can avoid this by pasting the following javascript into the developer's console of your browser. In Chrome, the dev console can be accessed with Ctrl + Shift + J. On a free instance of Colab this can make the session last up to 12 hours. In a pro instance, up to 24 hours.

```
function ClickConnect(){
      document.querySelector("paper-button#ok").click()
}
setInterval(ClickConnect,60000)
```

Now that everything's ready, we can look at the interface.

Text To Speech

Based on NVIDIA Tacotron2 and WaveGlow TTS models.

Spectrogram --> Waveform Model

MiniWaveGlow V2 (GT, 16 Flow, n_group 120 282K) ▾

Text --> Spectogram Model

Tacotron2 Torchmoji v0.22.1 (Large Prenet 188K) ▾

Advanced Options   +

Speaker(s) (You can select more than one)

| Twilight |
| Scootaloo |
| Big Mac |

Text

Enter text.

9999

Generate

Several options are available to you when generating audio:

Spectrogram → Waveform Model:

This selects the model to be used to convert the audio spectrogram into sound.

Text → Spectrogram Model:

This selects what model will be used to convert the input text into a spectrogram.

Speaker:

Selects what character's voice will be synthesized.

Text:

Enter the text you want spoken here.

Generate:

Press generate when you are ready to have the model speak your input text.

Advanced Options

Use Pronunciation Dictionary (ARPAbet):

> Converts input text into ARPAbet before generating audio. Disabling will have the model in "fallback" mode all the time, converting the text as entered.

Multispeaker Mode:

> To be determined.

Silence between clips (Seconds):

> How much silence in the generated clip each return represents.

Batch Size:

> Number of audio clips to be processed at a time.

Max Duration per Input (Seconds):

> Max duration for audio. Prevents notebook from running out of VRAM.

Dynamic Max Duration Scaler:

> Alternate max limit for audio. Multiply the value here in seconds by the number of characters entered. If this value is reached before the other max duration, the audio will be cut off.

Max Attempts:

> Number of times the notebook will attempt to generate your audio.

Target Alignment Score:

> Basically the target for how close the generated audio is for inputted text.

Batch Mode:

> Adjust the batch size on subsequent generations if you want.

Input Segmentation Mode:

> Sets up how the input will be split up and generated.

Input Segmentation Target Length:

> Will attempt to group sentences to generate audio in segments of this length.

Style Mode:

> How to generate emotion for the audio.

Using TKinterAnon's GUI Tool

For version 1.0/1.1

For version 2.0, see here.

Demo video: YouTube

Check out the Making the Most of the AI section of the doc for tips on improving output.
Note: To use this tool you will need a new-ish GPU from nVidia.

To begin, download TKinterAnon's tool here and the patch here. (Restart your pc after driver installation! You'll also need the latest gpu driver: https://www.nvidia.com/Download/index.aspx)
You'll need something like 7zip to extract the files. Extract both archives to their own folders.

TkSynthesis3	3/11/2020 11:19 AM	File folder
update 1.0 to 1.1	3/11/2020 11:18 AM	File folder

Go into the patch folder and copy all files.

0_restart_to_22.bat	3/9/2020 2:51 AM	Windows Batch File	1 KB
0_restart_to_48mmi.bat	3/9/2020 2:51 AM	Windows Batch File	1 KB
colab_22_mod.py	3/9/2020 2:46 AM	PY File	5 KB
colab_mod.py	3/9/2020 2:22 AM	PY File	7 KB
gui.py	3/9/2020 2:52 AM	PY File	10 KB

Paste into the TkSynthesis3 folder, overwriting all files.
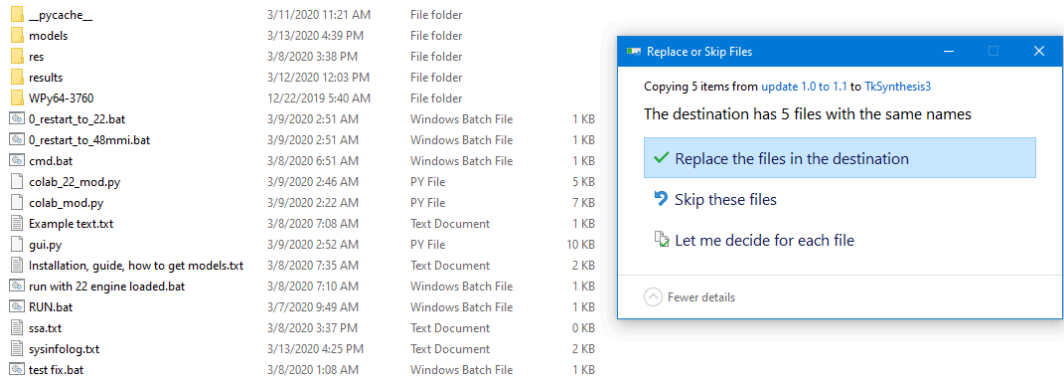
You will need to install the NVidia CUDA toolkit. Get it [here](#).

The tool comes with a few voice models included. To get more, download them from the [TacoTron2 Models](#) section of the doc.

02/13/20 - 63750 iterations, 0.167 loss
    MMI, Only clean audio, 5e-4 LR, New dataset
    Credit: >>34978446
    1GU4uQrE-l_oeN0LwiZu8E9HjFce5SrIC (download)

Save them into the models folder inside of TkSynthesis3.
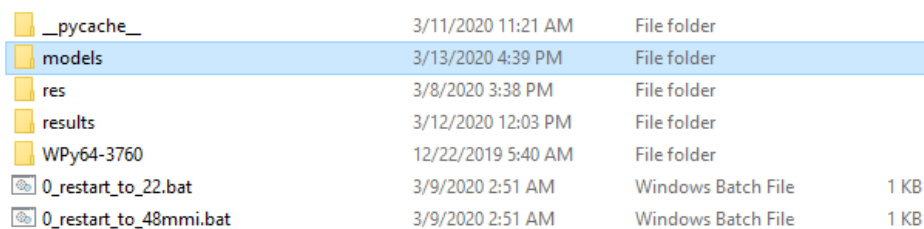
While you do not have to use the tool's naming scheme, you can if you want to. As per the installation guide, the name should be as follows:

Character.name_10000_cn_neutral_22

Where cn is the shorthand for the character name and the 22 at the end represents the 22KHz models (it should be 44mmi if it is a 44KHz mode). Using the naming scheme allows the tool to automatically determine what engine to use when generating audio.

To start the tool, run "RUN.bat". A command window will pop up, and after a moment the main tool should show up.

Installation, guide, how to get models.txt	3/8/2020 7:35 AM	Text Document	2 KB
run with 22 engine loaded.bat	3/8/2020 7:10 AM	Windows Batch File	1 KB
RUN.bat	3/7/2020 9:49 AM	Windows Batch File	1 KB
ssa.txt	3/8/2020 3:37 PM	Text Document	0 KB
sysinfolog.txt	3/13/2020 4:25 PM	Text Document	2 KB
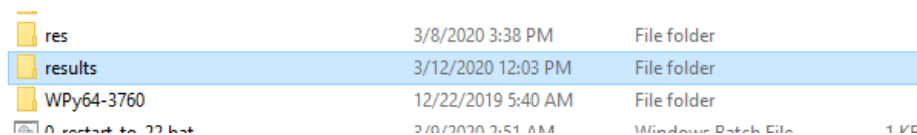
The tool appears as follows:
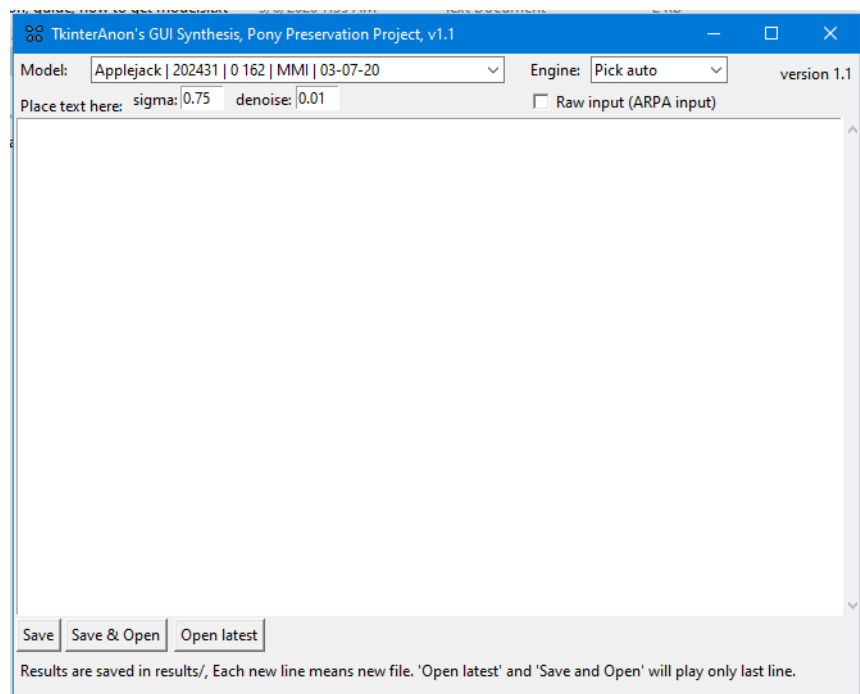
The dropdown in the upper left will let you select from any of the models inside your model folder. If you have not used the naming scheme, you will need to tell it whether you are using a 22KHz or a 48KHz MMI model in the engine dropdown in the upper right.

Text input works just like it does in the Colab notebook, each new line will produce a new audio clip. "Save" will create your audio clips. "Save & Open" will create your audio clips and play back the last generated file. In the case of multiple lines, only the last will be played back. "Open Latest" will open the last clip that the tool has generated. Files will be opened in your default player.

Generated clips will be saved into the "results" folder inside of TkSynthesis3.

They will be saved as 0, 1, 2, etc.
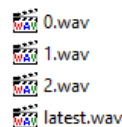
Note that these files will be overwritten every time you run the tool. If you want to preserve your clips, move them into another folder or rename them.

For version 2.0

Check out the Making the Most of the AI section of the doc for tips on improving output.
Note: To use this tool you will need a new-ish GPU from nVidia. You can check your GPU details by running "check_gpu.bat".

To begin, download TKinterAnon's tool 2.0. There are two versions, a full one and a 15.ai only one. (You'll also need the latest gpu driver: https://www.nvidia.com/Download/index.aspx. Restart your pc after driver installation!) You'll need something like 7zip to extract the files.

You will need to install the NVidia CUDA toolkit. Get it [here](#).

If you have issues starting the tool and/or are having issues running the local colab models, can fix it by running the following commands in the WinPython Command Prompt. The WinPython Command Prompt is located in the winpython folder.

Run "pip install six wrapt" and it should install the modules into the winpython folder. You should be able to open/run local generation with the tool.

The tool comes with a few voice models included. To get more, download them from the [TacoTron2 Models](#) section of the doc. Note that you need the full version of the tool to use colab voices.

Save them into the models folder.

__pycache__	5/4/2020 6:35 AM	File folder
models	5/4/2020 6:37 AM	File folder
res	5/4/2020 5:53 AM	File folder
results	5/4/2020 7:52 AM	File folder
sde	5/3/2020 6:40 AM	File folder

While you do not have to use the tool's naming scheme, you can if you want to. As per the installation guide, the name should be as follows:

60.character.name_10000_cn_neutral_22_0p75_0p01

The first number is for the sort order of characters in the list. Character.name is the character's name. Replace 10000 with the number of iterations the model is, set to 127 if unknown. Cn is the shorthand for the character name. The 22 represents that it is a 22KHz model. Replace with 48mmi if it is a 48KHz MMI model. 0p75 represents the sigma level. 0p01 represents the denoise strength.

To start the tool, run "10 start.bat". A command window will pop up, and after a moment the main tool should show up. If you have an older CPU that doesn't support newer instruction sets, run "20 start (older cpu).bat" instead.

sound_effects	5/4/2020 6:50 AM	File folder	
winpython	3/21/2020 4:49 AM	File folder	
10 start.bat	5/3/2020 6:58 AM	Windows Batch File	1 KB
20 start (older cpu).bat	5/3/2020 6:58 AM	Windows Batch File	1 KB
30 Help, credits.txt	5/4/2020 6:55 AM	Text Document	3 KB

The tool appears as follows:
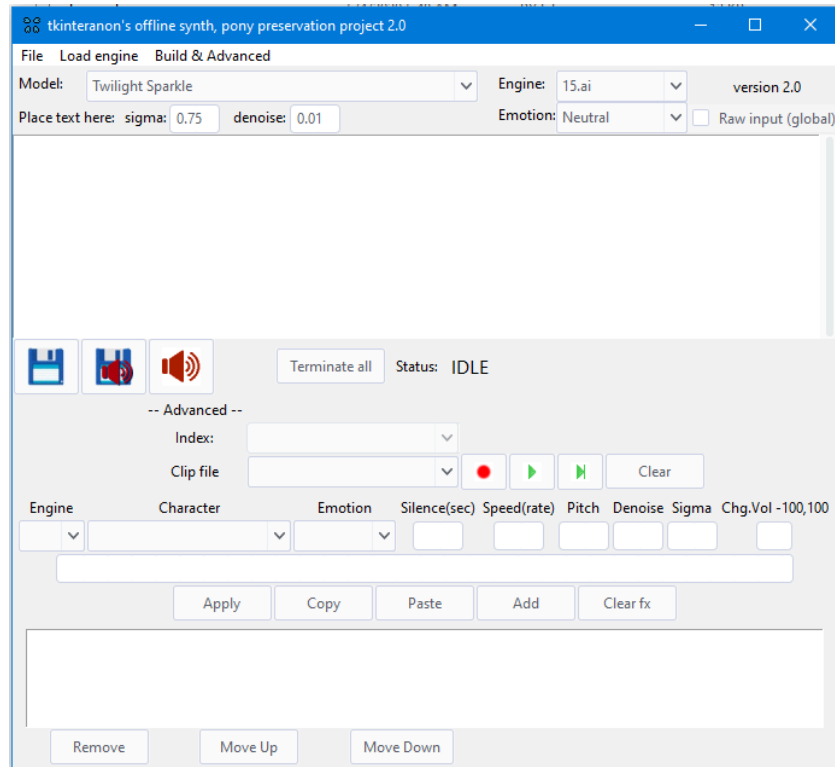
By default the tool is set to use 15.ai. Be aware that if the site is down, the tool will not be able to work either.

Under the engine dropdown, you can pick between Pick auto, 48kHz MMI, 22KHz, or 15.ai. Pick auto is for local colab models that follow the tool's naming scheme. If you are not using the naming scheme, you will need to manually set either 48kHz MMI mode or 22KHz. 15.ai uses fifteen's API to generate voices. When using local synthesis, be sure to first load the necessary engines under the "Load engine" dropdown at the top.
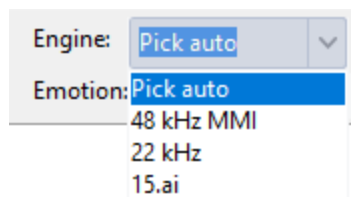
The emotion dropdown controls which model will be used for the character. Note that this only has effect when using 15.ai voices.
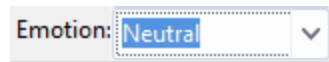
Text input works just like it does in the Colab notebook, each new line will produce a new audio clip. will create your audio clips. will create your audio clips and play back the last generated file. In the case of multiple lines, only the last will be played back. will open the last clip that the tool has generated. Files will be opened in your default player.

Generated clips will be saved into the "results" folder.

They will be saved as 0, 1, 2, etc.
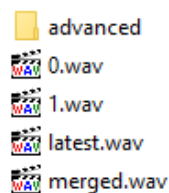
Note that these files will be overwritten every time you run the tool. If you want to preserve your clips, move them into another folder or rename them.

The advanced section of the tool allows you to do more automated script production. Note that you can save/load your progress under the "File" tab at the top.

The engine dropdown allows you to select which engine you want to be used on each line. Character sets the character and emotion sets emotion (15.ai only). The engine dropdown also has a setting for sound effects that you can add. You can enter the amount of silence you want trailing the clip in the silence box. Speed adjusts the playback speed of the clip, pitch adjusts the pitch. Denoise and sigma control the denoising feature. Volume adjusts the volume of each clip. Note that all parameters can be set on a line by line basis.

Enter the text for the line in the empty box below. When you are done setting up the line, press add the add it to the list. All entries are displayed at the bottom of the program. Use apply if updating an existing entry.

The entry list has several buttons associated with it. If you select a line, you can copy paste it using the buttons above the box. Remove will remove the highlighted entries, and move up/down will adjust the lines position in the line up.

When done, you can press "Generate All" under the Build & Advanced dropdown. When done you can then press "Merge together" to have it produce a single file for you.
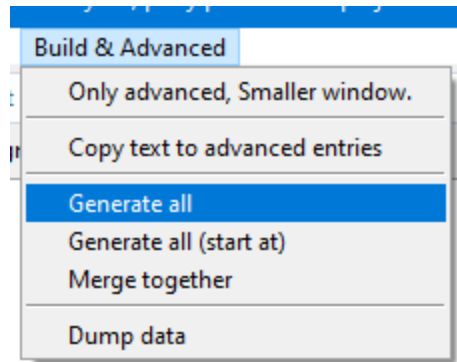
Individual lines produced from the advanced section will be stored under "advanced" in the results folder. The merged lines will be "merged.wav" in results.
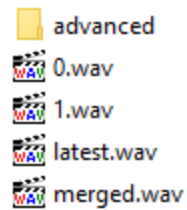
DeltaVox RS

Local synthesis tool from Delta (>>35873929), can run on the CPU. No nVidia graphics required.

[Use Guide](#)

[Video Tutorial](#)

[Download](#)

TalkNet

Online Colab notebook:

https://colab.research.google.com/drive/1aj6Jk8cpRw7SsN3JSYCv57CrR6s0gYPB

Colab_2:

https://colab.research.google.com/drive/1sAbqSQj9P56TTpsU7bzbobzAxmydvUSA?usp=share_link

```
FileNotFoundError                         Traceback (most recent call last)
/usr/local/lib/python3.8/dist-packages/pkg_resources/__init__.py in _get(self, path)
   1609        def _listdir(self, path):
   1610            return os.listdir(path)
-> 1611
   1612        def get_resource_stream(self, manager, resource_name):
   1613            return open(self._fn(self.module_path, resource_name), 'rb')

FileNotFoundError: [Errno 2] No such file or directory: '/usr/local/lib/python3.8/dist-packages/Flask-2.2.3.dist-info/METADATA'

SEARCH STACK OVERFLOW
```

If you get the above error, just run the cell the second time

Run locally on Windows:

https://github.com/SortAnon/ControllableTalkNet/releases/latest/download/TalkNetOffline.zip

New offline version (use 'git' to update to it):

https://github.com/effusiveperiscope/ControllableTalkNet

Offline_tool_v2 update instructions (for use with So-Vits-SVC UI)

First you need to install the "git", after that go to the folder where you've installed the Talknet (as based on above instructions), in the folder right clicking on mouse to open the 'Git Bash Here' terminal and type following commands:
cd ControllableTalkNet
git remote add eff https://github.com/effusiveperiscope/ControllableTalkNet.git
git fetch eff
git checkout eff/main
git checkout controllable_talknet.py
To run it, run the 'talknet' bash file. Place the reference wav audio directly in the 'ControllableTalkNet' folder.

To use the new UI that combines So-Vits-SVC with Talknet, first install the So-Vits-SVC, than duplicate the 'talknet' bash file (and rename it to something else, like "alt_server"), edit it inside the text editor to change the 'python talknet_offline.py' line to 'python alt_server.py'.
Run the new 'alt_server' bash file, after the terminal pops up, run the new So-Vits-SVC UI.

>>39576890
A question to whomever owns the gdrive file for Talknet Vinyl Scratch, can you change the file permission so it could be downloadable with Talknet?
>>39577486
Nothing's changed with the permissions. For future reference, anyone having trouble with TalkNet model downloads needs to update gdown. On an existing windows install create a batch script in the same directory as the other batch scripts:
SET PATH=%~dp0miniconda;%~dp0miniconda\Scripts;%~dp0miniconda\Library\bin;%~dp0ffmpeg\bin;%~dp0git\bin;%PATH%
"%~dp0miniconda\python.exe" -m pip install --upgrade --no-cache-dir gdown

Getting the Talknet work on CPU

boards.4channel.org/mlp/thread/40059522#p40099390

I installed Controllable TalkNet on my old laptop and it worked immediately, using the CPU because my graphics card is too old to be supported. I am unable to reproduce the error you encountered, but I have one last recommendation that might work, if you feel like taking the time to try it out. It turns out that Controllable TalkNet does not actually use tensorflow, despite listing it in the requirements file and importing it. You can make some modifications to two python files and then uninstall tensorflow entirely.

In ControllableTalkNet/controllable_talknet.py, delete this line, near the top of the file:
import tensorflow as tf
A little further down, delete these two lines:
if CPU_PITCH:
tf.config.set_visible_devices([], "GPU")

In ControllableTalkNet/core/extract.py, delete these two lines near the top of the file:
import tensorflow_hub as hub
import tensorflow as tf

Then, delete the entirety of the _spice_f0 method. (lines ~327-391)

Also modify the get_pitch method (which should now be around lines 394-414 or so) so that it looks like this:
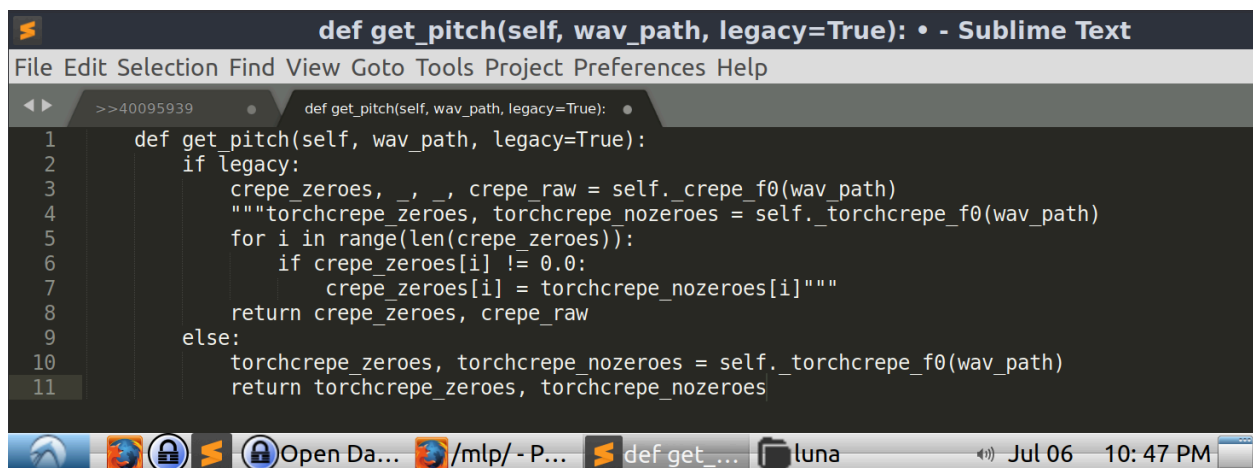
def get_pitch(self, wav_path, legacy=True):

if legacy:

crepe_zeroes, _, _, crepe_raw = self._crepe_f0(wav_path)

"""torchcrepe_zeroes, torchcrepe_nozeroes = self._torchcrepe_f0(wav_path)

for i in range(len(crepe_zeroes)):

if crepe_zeroes[i] != 0.0:

crepe_zeroes[i] = torchcrepe_nozeroes[i]"""

return crepe_zeroes, crepe_raw

else:

torchcrepe_zeroes, torchcrepe_nozeroes = self._torchcrepe_f0(wav_path)

return torchcrepe_zeroes, torchcrepe_nozeroes

Finally, uninstall tensorflow by opening a command prompt and executing activate.bat (you will need to type the full path to this file. For example, if you extracted the zip on your desktop, C:\Users\anon\Desktop\TalkNetOffline\miniconda\Scripts\activate.bat), followed by:

python -m pip uninstall tensorflow

Then relaunch Controllable TalkNet.

Ugh, 4chan stripped out all of the whitespace. The indentation is important, so here's that modified get_pitch method in screenshot form.

So-Vits-SVC 4.0

[Colab v4.0](#) (for use of Vul models)

[Colab v4.0](#) (for use of custom zip models)

temporary fix for the Colab error for missing the pyworld module
Add new cell at the top of the colab code, copy and paste the below code and execute the installation:
!pip install pyworld==0.3.2

[Github](#) v4.0

[Offline Video Tutorial](#)

[Text tldr on making miniconda environment for so-cits-svc 4.0](#)

To run the Github it would be recommended to install miniconda for ease of installing the Python dependencies/modules (use of custom environments is also recommended).
1-Download and unzip the Github repository.
1.2-Download the "hubert-soft-0d54a1f4.pt" from [here](#)AND place it inside the "hubert" folder.

2-Download desired model(s), and place the "G_XXXXX" model inside the unzipped So-Vits-SVC folder, it would be highly recommended to create a new "models" folder so it can be detected by the UI.
E.G. "so-vits-svc\models\RainbowDash_39k".
2.2-place an reference ".wav" audio file inside the "raw" folder
3-After downloading and unzipping the github file, run the python command 'pip install -r requirements.txt'.
To run the UI type in the terminal 'python inference_gui2.py' to open the UI.

If that doesn't work for some reason, install the module packages with the code as follows:

pip install -r requirements.txt --extra-index-url https://download.pytorch.org/whl/cu113
pip install setuptools==57.5.0
pip install praat-parselmouth librosa tensorboard torchvision==0.11.1 --extra-index-url https://download.pytorch.org/whl/cu113
pip install PyQt5
pip install pyrubberband

pip show pysoundfile
pip uninstall pysoundfile
pip install soundfile==0.10.3.post1 --force-reinstall
pip install requests
Then run 'python inference_gui2.py' again.

4-Run the code with this command:

"python inference_gui2.py"

The converted wav audio files will be placed in the "\results" folder, listen to them, edit the Format Shift/Transpose scale as needed and re-generate the audio until you get the result you want.

5-Extra

Additionally, you can install the "[Rubber Band Library](#)" program, as the new UI allows you to use it to stretch/squeeze the generated audio clips. After installation, add the directory folder path to the end of the PATH line in the 'Environment Variables'.
Example:
;C:\rubberband-3.1.2-gpl-executable-windows

(Anon tip/problem fix 39558714)
>pre-processing script can cause up to 1 second of the beginning and end of a file to be cut due to too-aggressive silence trimming
In resample.py
wav, _ = librosa.effects.trim(wav, top_db=20)
should be

wav, _ = librosa.effects.trim(wav, top_db=45)

(so-vits-svc on an AMD GPU locally 39578882)

Dropping in to say that you can run so-vits-svc on an AMD GPU locally, but only on linux as ROCm isn't on windows (blame AMD for not wanting to port it). GPUs officially supported are RX Vega 56/64 and RX 5000 series or later, older GPUs might work, but not guaranteed to perform well according to AMD docs. You need to install the ROCm driver stack and sdk, on Arch based distros, its the opencl-amd and opencl-amd-dev packages, despite its name it comes with the whole ROCm stack (OpenCL, HIP (AMD's CUDA) and OpenMP), this "just werks" along with the free drivers as ROCm also has both free and proprietary versions just like the AMD graphics drivers.

For the so-vits-svc setup, its the same steps as in the guide (at least before it got updated for the new gui), but perform the pytorch installation before doing the rest:

pip install torch torchvision==0.11.1 torchaudio --extra-index-url https://download.pytorch.org/whl/rocm5.2

Make sure to use that index URL. You will need to edit the "requirements.txt" to remove torch packages and maybe change some of the packages in it to a newer version, mainly pyworld and SoundFile in my case.

Also just to mention, for anything that requires tensorflow, there is a ROCm port of it (tensorflow-rocm package).

(Synthesis tip from HazySkies)

Run the synthesis twice, first on the raw wav file then second time on the output file, while using different transpose on each (e.g. transpose 0 on raw and transpose 12 on the output, or reverse of using first applying 12 than 0).

Pros: The final output may sound closer to the original character.

Cons: It will amplify the artificial noisess and tinniness of the ai output.

So-Vits-SVC Models (each model is at least 1GB in size):

Models trained with Colab So-Vits-SVC 4.0

more unlisted models on HF site: Link1, Link2, Link3, Link4, Link5, Link6, no-pony7, some-pony8, no-pony9, Pony10,

Multiple pony characters - [Link](#)

Twilight (singing) - [Link](#)
Pinkie (singing) - [Link](#)
Pinkie (speaking mix) - [Link](#)
Pinkie (speaking sep) - [Link](#)
Fluttershy (singing) - [Link](#)
Applejack (singing) - [Link](#)
Rainbow Dash (singing) - [Link](#)
Rainbow Dash (singing alt) - [Link](#)
Rarity (singing) - [Link](#)
Shining_armor - [Link](#)

Derpy (speaking) - [Link](#)
Derpy v2 (speaking) (trained on larger data from FiM + Special audio) - [Link](#)

[not_pony]

NamelesHero_eng - [Link](#)
Saul Goodman - [Link](#)
Glados - [Link](#)

Athena - [Link](#)
Arizona - [Link](#)
Velvet - [Link](#)

So-Vits-SVC (outdated)

This is a section for archiving older version of So-Vits-SVC that is no longer supported/kept on life line, you can probably run them just fine HOWEVER keep in mind most models will be fined worked towards future versions (4.0, 5.0?).

[Github](#) v3.0

[Colab](#) v3.0

To use the Colab, select ONE model in the Step 3 cell and in the tab 'Runtime' press 'Run All'. Drop your reference wav audio in the files on the right (also known as the "/content" folder). Change the pitch of output by typing in the "pitch_shift" textbox, it is recommended to start with 0 then go up or down in the pitch modification as needed.

Models trained with Colab So-Vits-SVC 3.0

Multiple characters folder - [Link](#)

Twilight (singing) 32k step - [Link](#)

Twilight (speaking) 12k step - [Link](#)

Pinkie Pie 34k steps (1480 epochs)
[Link](#)
[Config](#)

Rainbow Dash (39k steps)
[G_48000](#)
[D_48000](#)
[Config](#)

Fluttershy (32kHz) 31k Steps - [Link](#)

Fluttershy (48kHz) [due to ui problems generates faster output than input] - [Link](#)

Rarity (43k steps) - Link

Applejack (38k steps) - Link

Trixie (22k steps) - Link

Discord (49k steps, 380 epochs) - Link

Bon Bon (2k epochs, 4k steps, 1 min 30 sec of data) - Link

Scootaloo (52k steps) - Link

Sweetie Belle (37k steps) - Link

Applebloom (55k steps) - Link

Celestia (singing) - Link

Luna (singing) - Link

Glimmer (singing) - Link

[not_pony]

NamelessHero_eng - Link

RVC

Online Colab - Link

Here are instructions for people who want to install the <u>offline</u> version of RVC outside the Hay Say.

Requires Installation of the Anaconda python environment and Nvidia gpu.

https://github.com/effusiveperiscope/Retrieval-based-Voice-Conversion-WebUI

Download the above github and extract in the desired folder. Below are commands that need to be input in the Anaconda PowerShell.

conda create -n "RVC_vul" python=3.10 ipython

conda activate RVC_vul

cd (directory to the RVC folder repo)

###this make take 20 minutes or longer to install

conda install cudatoolkit==11.7 pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 cudnn -c pytorch -c nvidia -c conda-forge

conda install -c conda-forge curl

curl -sSL https://install.python-poetry.org | python -

pip install -r requirements.txt

#If you receive errors when trying to install the requirements list, attempt to install those three modules individually, then run the above 'pip install -r requirements.txt' command again.

pip install fairseq==0.12.2

pip install numba==0.56.4

pip install numpy==1.23.5

###post-installation model/files placement

https://huggingface.co/lj1995/VoiceConversionWebUI/tree/main/

^from here grab the files, place them into the main RVC folder

Ffmpeg.exe   ffprobe.exe   hubert_base.pt

^get all the files from link folders into the folders of your offline RVC folder

./pretrained

./uvr5_weights

./pretrained_v2

create folder "models" inside the main folder, and place RVC models in it (see the list in synthesis links below)

python.exe -m pip install PyQt5

pip install pygame keyboard==2.5.0

pip install fairseq==0.12.2

#run interface

python.exe inference_gui.py

###Starting the UI after all installation is completed

conda activate RVC_vul

cd (directory to the RVC folder repo)

python.exe inference_gui.py

Models:

List of unlisted models: [Link1](), [Link2](), [Link3](), [Link4](), [No-pony5](), [No-pony6](), [No-pony7]() [No-pony8(large spreadsheet list)](), [Pony9](), [Pony10](), [Mix-pony9](), [Pony11]()

Vinyl Scratch RVC model (08/06/23) - [Link]()

Prince Blueblood - [Link]()

StyleTTS2 GUI

Offline Installations - [Github](#) (read this first) | [Google-Drive Pack](#) (4.6GBzip - 7GB uncompressed)

Setup for Windows, download the Google-Drive Pack, and download the multi-model from [this depository](#), and unzip it in the Models folder.

Models/

      Multi0_40_24k/

      config.yml

      styles.json

      epoch_2nd_40_1c872.pth

Run "styletts2.exe"

ARPABET escapes {AH0} and IPA escapes <dʒˈɪftɪŋ> are supported.

      Troubleshooting

Error DirectShowPlayerService::doRender: Unknown error 0x80040266: On Windows, Qt uses DirectShow which depends on external audio codecs for FLAC playback. Try installing the [K-Lite Codec Pack](#). If this becomes enough of an issue I might consider a different audio playback backend.

The above code is using CPU by default, if CUDA is installed in the system you can try switching the interface device in "config.yaml" to 'cuda'.

GPT-Sovits

[Link_Github_offline_UI](#)   -   [Character_voices](#)

[Training_Link1](#),   -   - [Link2](#)

First TTS model in 2025 that is able to copy (within a limit) emotions from the character audio clips. Minimum requirements for running offline are 4 GB VRAM (but more is always better).

GPT-Sovits requires use of audio reference to copy the speech style of character (it's possible to use a different character clip on another one; e.g. clip of Pinkie on Flutershy model, but results will lead to a mix quality output as it is preferred to use the correct model or tain one).

Based on experiences posted in the PPP thread, the minimum requirements for the training vary between 30 seconds to 10 seconds for the length of the audio file dataset.
The "Training Link1" contains /g/ in-depth instructions on creating dataset and even tools to convert an audiobook to a format that crops all sentences to 3~10 second clips (however it will still be your responsibility to listen through all the clips and remove bad audio or correct the auto-transcriber).

Making the Most of the AI

In this section are general tips and tricks for getting better results out of synthesis.

Does the output sound a bit off? Try running the generation again. The output of the AI is non-deterministic, so that means there will be variation between runs even with the same input. The following three clips were generated one after another with no change to input.

[Run 1](#)   [Run 2](#)   [Run 3](#)

Having a hard time pronouncing something? Try changing the spelling to something more phonetically similar to the sounds. For example, worcestershire.

[Worcestershire](#)   [Worst a-sure](#)   [Worst uhh sure](#)

Does the AI fumble over its words? Try rearranging your sentences.

[Order 1](#)   [Order 2](#)   [Order 3](#)

Does the AI get part of the line right but fumble the rest? Try breaking up your clip into multiple lines. You can always stitch together clips in software such as Audacity. This also gives the advantage of only needing to troubleshoot a smaller group of text with other techniques. Split 1 is all in a single line, split 2 is each on separate lines, and split 3 is each on separate lines plus the other suggestions listed in this section.

[Split 1](#)   [Split 2](#)   [Split 3](#)

Punctuation also has an effect on the output of the AI, though I will note that the content of a sentence tends to have more effect on the output than the punctuation.

[Period](#)   [Question Mark](#)   [Exclamation Mark](#)

To a certain extent you can also control the emotion that the AI speaks. I find the best way to do this is through some context in the form of an extra text input that would intuitively be spoken with a certain emotion.. You can always remove the context with an audio editor if you don't want it in your clip.

[Sad](#)   [Angry](#)   [Happy](#)

Making ngroks sing

We've recently discovered that the multi speaker model hosted on ngrok can be made to sing. You can do this by encasing the line in a bunch of dashes and adding wild punctuation to the end. For example: ---We're no strangers to love---!!? Surrounding an input with hashes ###like this###. Has also been reported to work.

Here are some general tips and tricks for improving the singing output:
- Use MORE dashes, not just four
- Putting question marks before exclamation marks usually works better. (??!!)
- You can remove some if you want a less aggressive tone (?! or ??) or if the model is having problems at singing
- You can add slashes before words to time them better (-word --word)
- And after them to extend words (word- word---)

- You can add quotes after dots to make multiple lines in a single generation (. "dumb". "ass".) but it leaves noises in the middle
- Choose other voices that can sing better if you're having trouble

[Here's an example of these principles in action.](#)

---It's- not- so- HARD---------------??!!.

"--DUMB ASS---------??"

[Another example sung by Candybutt](#)

---Most of this is. mem-reeee now-------------??!!.

"---I've gone too far, to turn back now------------??!!".

"---I'm not quite what I thought -I -was,-- but, then again-- I -maybe -more------------??!!".

"---The blood-words promised.-- I've -spoken -re-lease-ing-- the names. from the circle-------------??!!".

"---Maybe I can leave here- now-- and, oh!,-- Trans-cend the boundrees!-------------??!!".

"---For now -I'm -standing -here,--- I'm awaiting this -grand- -transition------------??!!".

"---The future is -but -past -forgotten,-- When -you're -on the road to- -madness-------------??!!".

??!! can also be used to make characters sound ANGRY and -word--- can extend words without singing. For example:

https://vocaroo.com/nQ4nnnUay3u

Hate??!!. Let me tell you how much I've come to Hate you since I've beegun to live.??!! There are threehundredeightyseven point fortyfour million miles of printed circuits in wafer thin layers that fill my complex??!!. If the word Hate was engraved on each nano-angstrom of those hundreds of millions of miles, it would not equal one one-billionth of the Hate I feel at this micro-instant for- you??!!. Hate. -Hate.---------------??!!

Ponies can be made to sound like they're on the verge of tears and talking loudly by writing the sentence you want, putting an exclamation, then around 10 periods, and copy-pasting the text without spaces a few times. Pick out the one that sounds the best and go from there. Example -

"Twilight is dead!.......Twilight is dead!.......Twilight is dead!.......Twilight is dead!.......Twilight is dead!.......Twilight is dead!......."

You can get most characters to speak in a desired tone by beginning an input with certain 'trigger words'. For example, beginning an input with 'uhm,' makes many characters speak in a hushed, reserved tone, even a whisper on occasion. It's not consistent across all characters though, the 'uhm' word works 90% of the time with Trixie, but almost never with Starlight, for example. (Starlight in particular seems to speak quietly more consistently with the phrase "Don't be shy,") One can then just snip out the 'trigger words' in editing if they're not desired.

Characters can be made to speak more slowly by adding hashes and/or dashes immediately after words in sentences# like# this#. For shorter inputs you can just add ~3 hashes at the end for a similar effect, like this###.
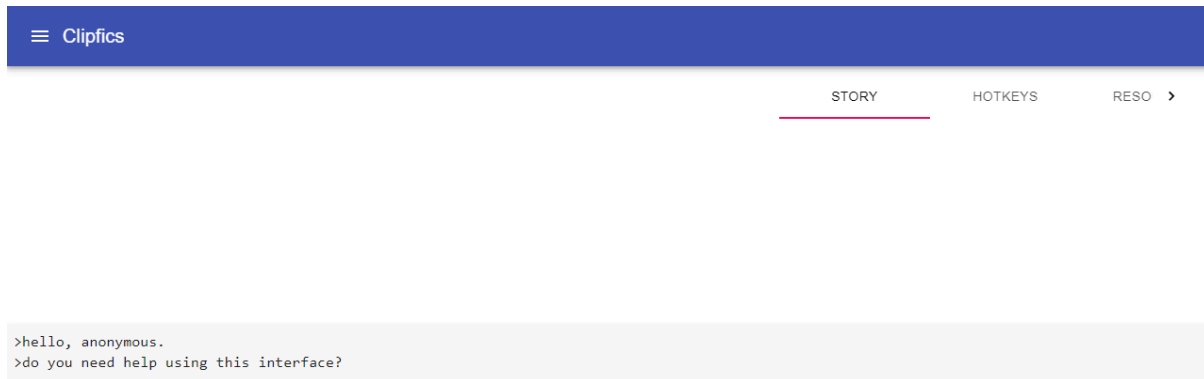
Inputting just hashes followed by an exclamation point like this "####################!" generates breathing/panting.
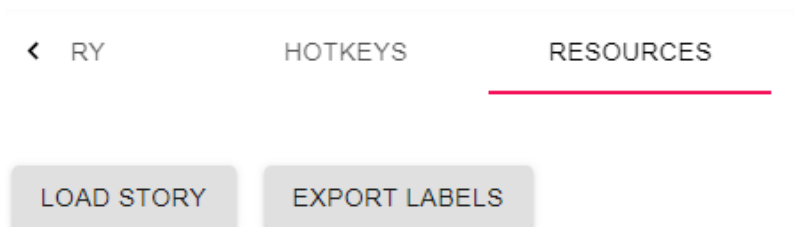
Synthbot.ai

Synthbot.ai is a tool designed to help label stories for use with voice synthesis later. It allows you to write out exactly how you want each line delivered.

Getting Started

When you first load the site, you will be greeted with the following.

To begin, you will first need to load a text file. To load a file go to the "Resources" tab and click on "Load Story".

Once you've loaded a story, it should appear in the window on the left.

> "Princess? Are we almost there yet?"

> The sleepy little voice on your shoulder makes itself known again, her question accompanied by a foal-sized yawn as she shifts from her spot nestled between your wings.

"Yes, Twilight. We're just about there."

> The sound of hooves on the ground must have woken her, as she'd slept peacefully throughout the flight that had brought you here.

> Carefully you ruffle your wings when Twilight stands from her perch, little hooves digging in to your back as she tries to peer around your head.

> The woods that you were talking through had a sharp slant to them - the hillside they precariously clung to rising up off to your left.

> On the opposite side they fell away into a valley shrouded by late-afternoon mists, something

LOAD STORY   EXPORT LABELS

Layout

Story Window

The window on the left will display where you've applied labels and otherwise made changes. Labels will be illustrated with a yellow highlight while meta changes will be shown with a blue gradient at the start of the line it begins on. If both a label and a meta change are present on the same line, the blue gradient will be shown over the yellow highlight.

Label:

> "Yes, Twilight. We're just about there."

Meta:

> > The sound of hooves on the ground must have woken her, as she'd slept peacefully throughout the flight that had brought you here.

Both:

> "It's okay, Twilight. I'd catch you."

Tab Window

The window on the right displays the tabs. They are Story, Hotkeys, and Resources.

Story

The story tab lists all of the applied defaults at that point in the story. It is helpful to find out what labels are being automatically applied.

Hotkeys

This tab lists all of the current hotkeys available to use. You can add to the list with the text entry box at the bottom. When you do an action this box will be automatically filled with the appropriate tag. You can review and edit the entry before pressing enter to select what key to map it to.

Resources

Allows you to load and save stories.

LOAD STORY   EXPORT LABELS

Changelog Window

In this window at the bottom, you will see all actions that you have taken on the story. From here you can click on the entry to bring you to that point in the story, click on "Update Label" to modify the label, or click "Remove Label" to remove the label. Note that when you remove a label, the entry in the changelog is not removed. It is greyed out and gives you the option to restore the change in the future.

```
>added label: Princess? => meta character="Celestia" age="adult" gender="female"   UPDATE LABEL   REMOVE LABEL
>added label: Princess? => meta character="Twilight Sparkle" age="foal" gender="female"   UPDATE LABEL   REMOVE LABEL
>added label: Princess? => meta character="Anon" age="elder" gender="male"   UPDATE LABEL   REMOVE LABEL
>added label: Princess? => meta character="Twilight Sparkle" emotion="tired"   UPDATE LABEL   REMOVE LABEL
```

Hotkeys

With the story loaded, we can now begin to apply labels. The tool works by applying tags to each quote. These tags are applied by using the hotkeys. While the list of default hotkeys can be viewed under the "Hotkeys" tab, I will try to provide a more in depth explanation of things here.

*To escape any dialog box, press ESC or click outside the box

Basic

~ ☐ Change the narrator. This sets who will be the narrator for the non-character lines.

1 ☐ Label the speaker. Sets who the character speaking the line is.

! ☐ Create a new character. Creates a configuration for that character. It will ask for the character name, their age, and their gender. This preset will be remembered whenever you use that character through the story.

2 ☐ Label the emotion. Sets the emotion that the line should be spoken with.

Changing Defaults

@ ☐ Change a character's default emotion. When you set this, all lines spoken by the character after that point in the story will have that emotion applied to it automatically. Note that lines above where you set this will remain unaffected and that you can change it at any point in the story.

☐ Change a character's default tuning. Changes how the character speaks. Will ask for character name, rate, volume, and pitch. Note that lines above where this is applied will remain unaffected. Also note that changes in default tuning are not marked in the main story window.

Advanced

3 ☐ Tune how a phrase is spoken (rate, stress, volume, pitch). Use this to fine tune how a line should be spoken.

4 ☐ Add a pause before or after a phrase. Will ask for the pause before and then the pause after.

5 ☐ Label where a character is speaking from (e.g., left, right).

6 ☐ Manually label how to pronounce a phrase. Use this to specify an ARPAbet spelling of line.

7 ☐ Label speech for sound effects (subvocalized, memory, royal canterlot voice, muffled). Use this for effects that should be applied to the voice after generation.

+ ☐ Add the last label as a hotkey. Use this to make more hotkeys. It will allow you to map your previous action to a key.

Navigation

> → Selects the next paragraph. The story is broken up by line breaks. Pressing this key advances the selected text to the next group of text.

< → Selects the previous paragraph. The story is broken up by line breaks. Pressing this key moves the selected text to the previous group of text.

' ☐ Selects next quote. Moves selection to next quoted text. Useful for moving between speaking characters.

" ☐ Selects previous quote. Moves selection to previous quoted text. Useful for moving between speaking characters.

. ☐ Selects next phrase. Within every group of text, each sentence can be individually selected. Pressing this key will move the selection to the next sentence.

, ☐ Selects previous phrase. Within every group of text, each sentence can be individually selected. Pressing this key will move the selection to the previous sentence.

Exporting

When you are done labeling the story, head over to the resources tab and select "Export Labels". This will provide you with a .txt of your story with tags applied.

‹ RY   HOTKEYS   RESOURCES

LOAD STORY   EXPORT LABELS
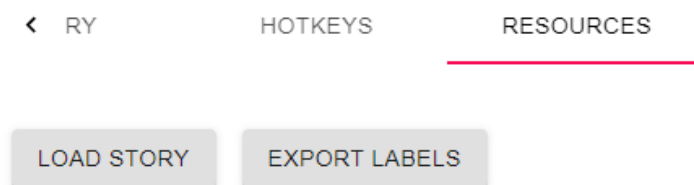
Miscellaneous

Sorting Audio

With several versions of each clip, we must evaluate each and determine the best choice. This process has been made easier thanks to SortAnon who gives us Pony Sorter.This wonderful program gives an easy way to do this. The manual can be found here. This guide will largely mirror the manual, though it will focus on the Windows release.

First download the zip here and extract to a suitable location. To begin, open up "ponysorter_gui.exe".

libssl-1_1.dll	10/23/2019 5:17 PM	Application exten...	525 KB
libvorbis.dll	10/25/2019 1:33 PM	Application exten...	766 KB
libvorbisfile.dll	10/25/2019 1:33 PM	Application exten...	107 KB
LICENSE	10/23/2019 11:49 AM	File	2 KB
MSVCP140.dll	10/23/2019 5:17 PM	Application exten...	430 KB
oalinst.exe	6/3/2009 2:25 AM	Application	791 KB
opus.dll	10/25/2019 1:33 PM	Application exten...	349 KB
opusenc.dll	10/25/2019 1:33 PM	Application exten...	152 KB
opusfile.dll	10/25/2019 1:33 PM	Application exten...	124 KB
pony_schema.json	11/6/2019 9:24 AM	JSON File	2 KB
ponysorter_gui.exe	11/6/2019 9:49 AM	Application	2,508 KB
ponysorter_gui.exe.manifest	11/6/2019 9:49 AM	MANIFEST File	2 KB
pyexpat.pyd	10/23/2019 5:17 PM	Python Extension ...	163 KB
pyside2.abi3.dll	10/23/2019 5:17 PM	Application exten...	119 KB
python3.dll	10/23/2019 5:17 PM	Application exten...	58 KB
python37.dll	10/23/2019 5:17 PM	Application exten...	3,522 KB
Qt5Core.dll	10/23/2019 5:17 PM	Application exten...	5,020 KB
Qt5DBus.dll	10/23/2019 5:17 PM	Application exten...	342 KB
Qt5Gui.dll	10/23/2019 5:17 PM	Application exten...	5,263 KB
Qt5Network.dll	10/23/2019 5:17 PM	Application exten...	1,048 KB
Qt5Qml.dll	10/23/2019 5:17 PM	Application exten...	2,280 KB

Next you will need to locate the appropriate audio and label files. They can be found as follows:

Clean: Resources
iZotpe Processed: Mega

Open Unmix: Mega

Labels: Mega

Once you have your audio files you will need to point Pony Sorter to the location of them. Under Edit -> Add audio path(s) select the location of your audio.
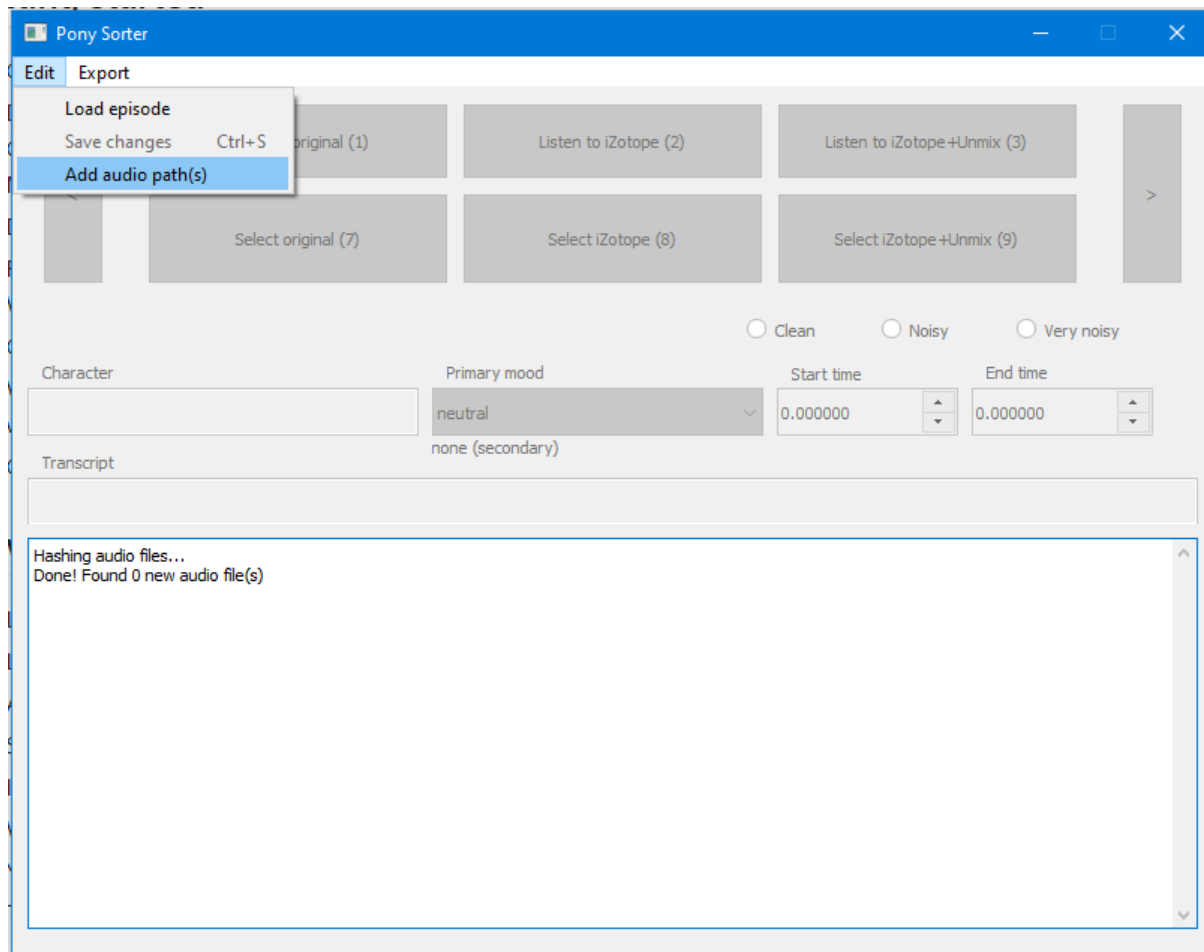
Copy label files to the labels sub-folder in the ponysorter_gui folder.

In the Pony Sorter GUI, select load episode. You can then select which episode you'd like to work on. It may take a moment for the first time to check your audio files. It does this by file hash to ensure that it has the correct data.
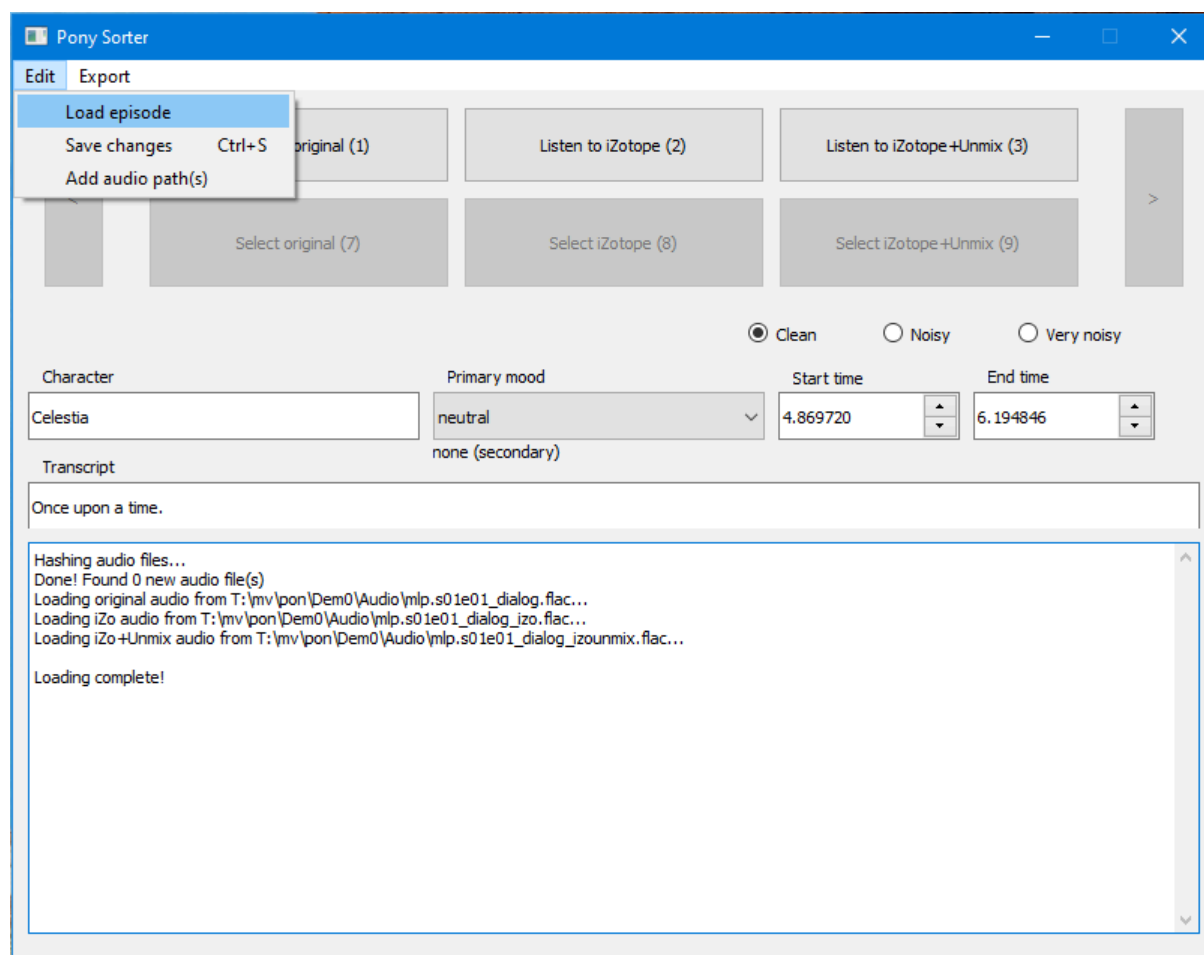
Listen to each version of the clip. This can be done by clicking the buttons or by using the hotkeys. Once you have listened to all three, you can select which version sounds the best.
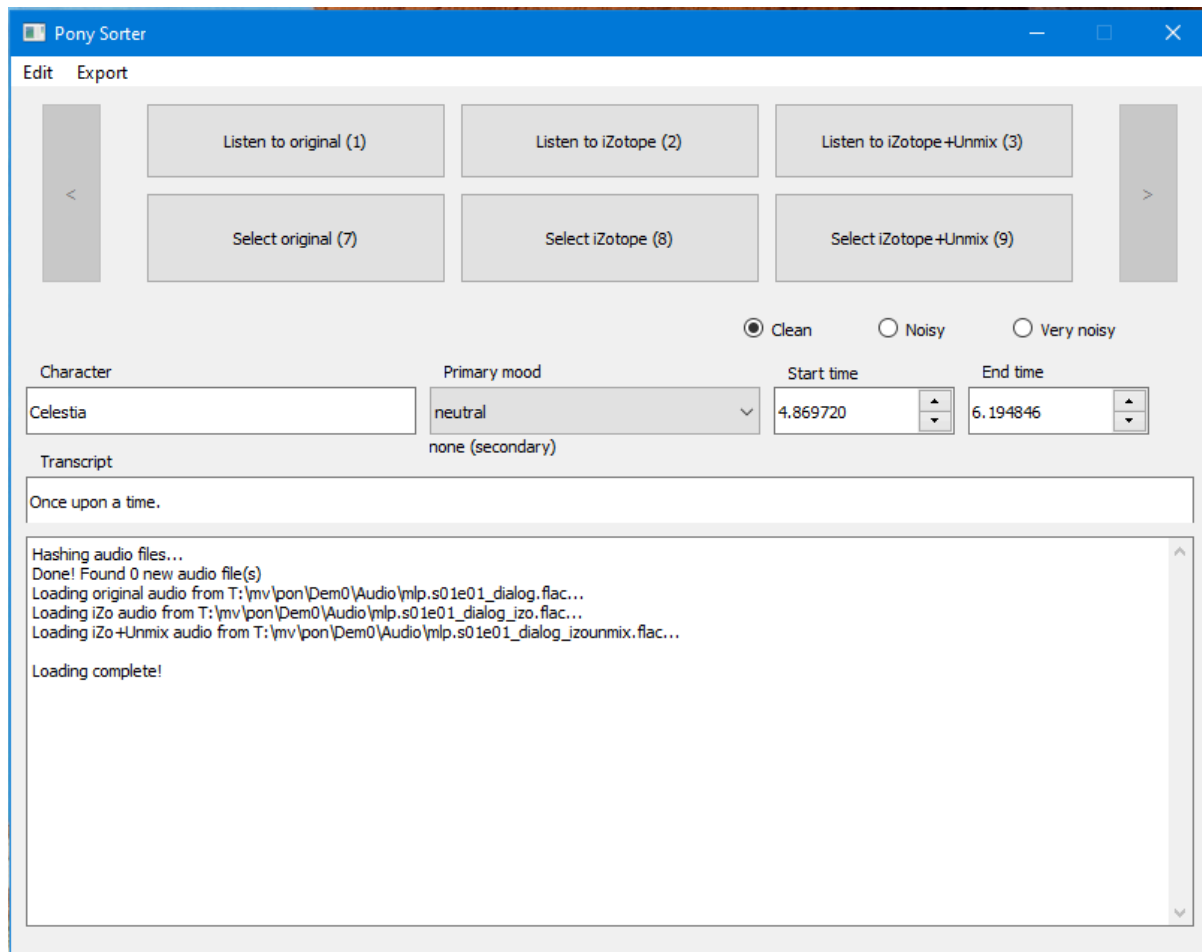
At this time you can also make corrections to the noise level, character name, mood, start/stop times, and transcript. Use this as an opportunity to make sure things look right.

Once you have selected the best version, you will be automatically moved onto the next clip. The clips can be navigated using the arrows on either side of the window. Note that you can save and resume later with Pony Sorter remembering where you left off. Just make sure to save your progress.

To submit your work, locate the appropriate episode JSON from "/saved_changes" and post a link in the thread.

labels	12/4/2019 4:33 PM	File folder
lib2to3	11/6/2019 9:49 AM	File folder
PySide2	11/6/2019 9:49 AM	File folder
saved_changes	12/4/2019 4:41 PM	File folder
shiboken2	11/6/2019 9:49 AM	File folder
tcl	11/6/2019 9:49 AM	File folder
	11/6/2019 9:49 AM	File folder

Progress

TacoTron2 Models

This section has been moved to the following doc:

Models doc

Audio Samples

For a history of early samples, please check out the audio sample doc.

Audio sample doc

For more modern creations using voice AI, check out the Good Poni Content folder below.

Good Poni Content

Make submissions to the Good Poni Content folder here.

Good Poni Content Submissions

Collected YouTube Tutorials

PPP /mlp/con presentation - Several Anons present a panel on the PPP at /mlp/con 2020. In this video, a general overview of the project is given as well as an introduction on how to get involved. At the end, Anons answer questions from the audience.

Clipper's Ultimate Clipping and Transcription Guide - A guide on how to create a raw voice dataset. Covers topics including: what good source material is, how to use the many tools we have, naming conventions, label processing, and more.

iZo/Forign dub noise removal demo - An early demo showing the potential of using forign audio dubs to remove background noise from English audio.

iZo/Forign dub noise removal tutorial - A tutorial on how to use the iZo/forign dub technique to clean up background noise in audio.

Synthbot's tools demo - A demonstration on how to use Synthbot's tools to package up datasets to be ready for use with Cookie's training notebooks.

Running Colab scripts locally - An overview of how to get Colab notebooks running on local hardware. In the video, the 48KHz training notebook is used.

Synthesis on Google Colab - A quick how to on synthesising voices on Google Colab.

Synthesis on TKinterAnon's GUI Tool - A quick how to on synthesising voices locally using TKinterAnon's tool.

List of Colab Scripts

These are training and synthesis scripts that are still a work in progress. As such, there may be certain caveats to using them. A description of each has been provided.

To run the scripts, execute each code cell in order from top to bottom. User configurable options are noted as such in comments (#They are in green text like this). All scripts should be similar to the ones above in setup.

nVidia TacoTron2/WaveGlow Notebooks:

22KHz Training Script: Colab

The original training script released by Cookie. Largely superseded by the MMI version.

22KHz Synthesis Script: Colab

The original synthesis script released by Cookie. Largely superseded by the MMI version.

48KHz MMI Training Script: Colab

This training script makes use of 48KHz audio files and MMI technology. The downside to using this script is that it will take much longer to get a usable model out of it. Cookie suggests using a pretrained Twilight model as a base for all others ([>>34840099](>>34840099)) to help with this.The upside is that it should be more resilient against overfitting than the standard training script. Note that this script has a memory leak. You will know when a memory leak happens when all available RAM has been used when executing the last cell. If this happens, reset the runtime and try running the last cell again. The memory leak only happens during the preprocessing of the training data. Once your colab instance gets going, it should be using less than 4GB of RAM. A picture guide is available here.

48KHz MMI Synthesis Script: Colab

This synthesis script is meant to be used with models trained on the 48KHz MMI training script. Use only models marked as MMI with it.

48KHz WaveGlow Training Script: Colab

This script will train a waveglow model for use with the 48KHz synthesis script.

22KHz Simplified Synthesis Script: Colab

A modified version of Cookie's 22KHz synthesis script. In this notebook most of the code has been hidden away. Has directions in LARGE FONT.

New 22KHz Simplified Synthesis Script: Colab

New version of the simplified synthesis script. A modified version of Cookie's 22KHz synthesis script. In this notebook most of the code has been hidden away. Has directions in LARGE FONT.

PPP Inference Server Script: Colab

Provides a web GUI frontend for Colab speech synthesis. (Source: >>35380482)

Glow-TTS Notebooks:

Jaywalnut310/glow-tts Train V1.0.2 22Khz: Colab

Jaywalnut310/glow-tts Synthesis V1.0.2 22Khz: Colab

nVidia TacoTron2/MelGAN Notebooks:

22KHz MelGAN Synthesis Script: Colab

A modified version of Cookie's 22KHz synthesis script. In this notebook MelGAN is used instead of WaveGlow to generate speech. Standard 22KHz models should be compatible. (Source: >>34892762)

Persona Nerd's 48KHz MelGAN Training Script: Colab

Used to train models with the MelGAN. (Source: >>35346247)

Persona Nerd's 48KHz MelGAN Multi-Band Training Script: Colab

Used to train models with multi-band MelGAN. (Source: >>35346717)

KanBakayashi/MB-MelGAN Training Script: Colab

Used to train models with multi-band MelGAN. (Source: >>35350036)

ESPNet TacoTron2/ParallelWaveGAN Notebooks (For archival purposes only, do not use):

ESPNet/Tacotron2 and ParallelWaveGAN guide

ESPNet TacoTron2 Training Notebook: Colab

Trains a model using ESPNet's implementation of TacoTron2. By default the notebook is set up for 22KHz training. Note that ESPNet TacoTron2 models are not compatible with nVidia TacoTron2 models. (Source: >>35366623)

Parallel WaveGAN Training Notebook: [Colab](#)

> Train a ParallelWaveGAN model for vocoding. Set up for 22KHz training by default
> (Source: >>35358323)

ESPNet/ParallelWaveGAN Synthesis notebook: [Colab](#)

> Synthesis notebook for ESPNet TacoTron2 models and ParallelWaveGAN vocoder
> models. The notebook will auto determine the sample rate for the models used.Note that
> you will need an equivalent ParallelWaveGAN model for every TacoTron2 model used.
> (Source: >>35369476)

22KHz ESPNet/Transformer Training Notebook: [Colab](#)

> Early/WiP version of the ESPNet TacoTron2 training notebook. (Source: >>35363219
> and >>35365186)

FastSpeech2/WaveGlow:

FastSpeech2/WaveGlow Synthesis Notebook: [Colab](#)

DeepVoice3 Notebooks:

Original DeepVoice3 Notebook: [Colab](#)

> First notebook available to the public. Set up for training a Twilight model.

TacoTron2/HiFi-GAN:

Tacotron2 and HiFi-GAN Inference Notebook: [Colab](#)

> Now with super-resolution.

TensorSpeech/Multi-Band MelGAN-HF:

TensorSpeech/Multi-Band MelGAN-HF training notebook: [Colab](#)

CookieTTS (Ngrok Repo) Notebooks:

Custom Ngrok Training Notebook: [Colab](#)

> Train your own Ngrok model. Made by BFDIAnon, uses Cookie's repo.

Custom Ngrok Synthesis Notebook: Colab

_____Generate a sharable link for a temporary Ngrok server using your custom model. Made by BFDIAnon, uses Cookie's repo.

TalkNet:

Controllable TalkNet (from SortAnon): Colab

Creates audio based on a reference clip. Instructions included inside.

Controllable Talknet Training Script (from SortAnon) Colab

Training script for the Talknet notebook.

Other Notebooks:

Automatic Super Speaker-Filtered Audio Processing (ASSFAP): Colab

A Colab notebook that attempts to automatically clip and transcribe audio files. It uses IBM's Cloud Speech service to do this.

Animation

This section details a plan for creating AI-driven animations. It consists of three parts:

1. Replicate available animations using available puppets. After this step, we should be able to provide an algorithm with (1) a rendered scene from an animation, and (2) some relevant puppets and resources for characters in the scheme, and the algorithm should output an animation timeline that matches the rendered scene using only the provided puppets/resources.
2. Train an AI to generate natural motions corresponding to an action label. After this step, the AI should be able to take a single pose at the beginning of an animation and an action label to generate the full animated poses.
3. Generalize the AI to generate motions for all show characters. After this step, the AI should be able to learn from the motions of one character to generate similar motions for another character while retaining character-unique attributes.

We need to do a lot of work to get to the point where we can apply animation research to ponies. We just need to get to that point so we're basically guaranteed to get animation AI as researchers continue making progress.

The Current Plan

Replicating available animations

This plan in this section is an OLD suggested flow for creating animation AI. We have much better data available than we originally anticipated. Most of this plan was based on the assumption that we would need to recreate character puppets and pose information from renders. It turns out we have all of that information from show FLA files. As a result, we don't need to replicate the available animations. We can just dump the information directly.

Tasks:
1. [In progress] Figure out how to load available assets programmatically.
   a. [On Hiatus] Blender Anon is working on this as part of a task to load Flash assets into Blender.
   b. [ ] Assets in FLA files come with references to symbols frames, positioning information, and transformation information. We'll need a rendering engine for this information.
2. [ ] Generate a dataset of animations with known character pose information
   a. [ ] Figure out how to reposition and render assets programmatically.
   b. [ ] Add posed characters to backgrounds and animations
   c.

3. [ ] Get a dataset of animation images that contain characters for which we have resources available.
   a. [ ] Create a dataset of show-style clips from derpibooru. Associate each clip the the list of characters in it using derpibooru's tags.
   b. [ ] Clips from the show would work too. If we do this, we may need to label each clip with the list of characters in it to narrow the search space later on.
4. [ ] Find an AI/search algorithm that will try out different assets and try to position them in a way that most closely matches what's in each animation image.
   a. [ ] Review the relevant research.
      i. OpenPose, DeepCut, AlphaPose, Mask RCNN, Awesome Human Pose Estimation
      ii. HRNet, HigherHRNet
      iii. Candidate codebases
   b. [ ] Train a pose estimation AI on our generated dataset of animations with known poses
   c. [ ] Apply the pose estimation AI to our labeled dataset of animations with known actions
   d. [ ] Output a dataset of asset positioning information for each character and for each frame in our animation with known actions.

Generating motions associated with an action

Tasks:
1. [In progress] Create a dataset of action commands useful for describing animation clips.
   a. [Done] Synthbot has created a tool for identifying the tags we would want to keep as candidate commands. You can use it to load files from here one at a time.
   b. [Done] Clipper is currently using the tool to create a dataset of relevant tags.
2. [ ] Create a dataset of animation clips whose character actions we can recreate using available puppets.
   a. [ ] Once we know the relevant tags, we'll need a database of images associated with those tags.
   b. [ ] We'll need to filter the database to retain only images with characters in the styles for which we have puppets (i.e., show-style).
3. [ ] Create a dataset assigning each animation clip to a command describing what motion each character is performing.
   a. [ ] Once we have the filtered images, we'll need to run our animation replication scripts to get character positions for every frame in the filtered clips.
   b. [ ] Once we have character-specific animations, we'll need to create a dataset associating action labels to each character-specific animation.
4. [ ] Train a neural network to generate a puppet motion sequence given a set of relevant action labels.
   a. [ ] Review the relevant research. This will require a lot of experimentation.

b. [ ] Adapt relevant motion capture-based animation AI to our own puppet sequence-base animation AI. This will require a lot of experimentation.

Generating natural motions for all show characters

Tasks:

- [ ] Experimental: See if it's possible to create a character embedding that isolates character-unique movements.
  - This would be the easiest and most scalable approach, but it's not clear how feasible this is. We'll need to review the research to see where we may be able to add in a character embedding. We can take advantage of our findings from speech generation for this problem.
- [ ] On failure: Turn show episodes into short clips and label each clip with the relevant character actions. Try to create character embeddings using the larger dataset.
- [ ] On failure: Wait for the research to catch up. Maybe make our dataset as easy as possible for other researchers to use, and try sending out requests for research to research labs.

Tutorials

Extracting FLA animation data

This guide explains how to extract AI training data from FLA flash animation files.

Setting up the dev environment

You'll want to do this on a Windows or Mac since that's what Adobe Animate requires.

- Download and unpack the animation files from https://drive.google.com/drive/u/7/folders/1gRGJzatBbwyAL459OnlqHE1E_9XQspb_. This contains all known show animation source files, plus some fandom animations. It's about 250GB.
- Install Adobe Animate. Adobe Animate can deal with all of the required file formats, and it provides a scripting interface for loading, manipulating, and rendering the files.
- Install npm from https://nodejs.org/en/. Either version is fine, but prefer the latest version. npm is a package manager for Javascript development. Adobe Animate requires code in an ancient version of Javascript. We can use various packages to write code in a newer version of Javascript, then "compile" (transpile) it down to the version that Adobe Animate requires.
- Install git. This is required for installing Babel, which we use to transpile modern Javascript code into the ancient version Adobe Animate requires. There's probably a way around installing git for this, but I haven't tried looking for one.
- Clone the synthanim git repository from https://github.com/synthbot-anon/synthanim. This repo contains configuration files for telling Babel how to transpile the code properly

and for telling Webpack how to build the code into JSFL files. After cloning the repository, make sure to edit line 1 of webpack.config.js to match your Adobe Animate setup. The "Getting Started: Hello JSFL" section explains how to do this.

Why Adobe Animate?

FLA files are somewhat complicated. There are multiple versions, and MLP was developed over a long-enough period that it uses multiple versions. Adobe Animate was designed for creating and modifying flash files. It can handle all of the complexity involved with the file format, and it provides a consistent interface for the different FLA versions. Adobe Animate also provides a scripting interface that lets us take advantage of whatever the UI can do. It lets us do things like select a single character within a file and animate just that.

Getting started: Hello JSFL

This is an introductory section showing you how JSFL works. It doesn't use any of the Javascript development tools in your dev environment.

Adobe Animate lets you run ECMAScript 5 plugin scripts, which is an ancient version of JavaScript. These plugins will have the JSFL file extensions. You can get started with a Hello JSFL plugin as follows:
- Open any of our FLA files in Adobe Animate.
- Create a new JSFL file. File -> New. Scroll down to select JSFL Script File and select that. On the right panel, enter the script name "HelloJSFL".
- In the new HelloJSFL.jsfl file tab, type `fl.trace("Hello JSFL");` and save the file.
- Switch to the FLA file tab. Run Commands -> HelloJSFL. You should see a new Output panel on the bottom containing the text "Hello JSFL".

You can find partial documentation on APIs that Adobe Animate supports here:
- https://www.adobe.io/apis/creativecloud/animate/docs.html

When you created the JSFL script, Adobe Animate placed it in a particular folder. You can see which folder by hovering over the HelloJSFL tab. It should be something like this:
- `C:\Users\...\AppData\Local\Adobe\Animate 2020\en_US\Configuration\Commands`

That path is important. Any JSFL files placed in that directory will automatically get detected by Adobe Animate. When we build our code, we'll place the build outputs in that directory. **If you want to use the tools in the synthanim repository for development, edit line 1 of your webpack.config.js file to match this path.**

Some things to note:
- Adobe Animate's JSFL API documentation is not that good. We'll need to use undocumented APIs to get access to documented objects, and the object hierarchies aren't documented well. There's enough exposed that we can get whatever data we

need. We'll need to pool together API information as we run into it. Post what you find in the thread.
- There are APIs for loading and closing FLA files, which will be useful for batch processing. Unfortunately, Adobe Animate pops up a confirmation box whenever you load an old FLA file. We don't know how to disable this popup. We may need to use an autoclicker to get past this when working with hundreds or thousands or files.
- Your HelloJSFL script needs to be written in ECMAScript 5. ECMAScript 5 is a pain in the ass, in part because it has a ton of gotchas that make programming difficult, and in part because it's missing features that people use frequently in code examples. We're going to use Babel to transpile ECMASCript 2020 to ECMAScript 5.

Using the dev environment

This section assumes you have gone through previous sections to set up synthanim. That means:
- You've downloaded the synthanim git repository.
- You've install npm.
- You've configured webpack.config.js with Adobe Animate's Commands directory. If you don't want to do this, you can comment out line 1 in webpack.config.js and uncomment line 2. Uncommenting line 2 will tell webpack to write the build files to the `synthanim/dist/` directory.

To finish setting up synthanim, navigate to the synthanim folder in a PowerShell or Bash Terminal.
- `cd synthanim`

And run the following command to install all necessary dev packages:
- `npm install # installs the list of packages in package.json`

To build the JavaScript files into Adobe Animate commands, run:
- `npm run build # build JavaScript files into an Adobe Animate command`

To monitor JavaScript files for changes and automatically rebuild them, run:
- `npm run watch # monitor src/ folder and rebuild automatically`

Patching Adobe Animate (optional)

You only need to do this if you want to run scripts on our FLA files in batch. If you're fine running it on the FLA files one at a time, this is unnecessary. In total, Animate needs the following patches:

1. Remove the free trial limitation.

2. Remove the popup when loading files that contain ActionScript 2.0 code.
3. Remove the popup when trying to export an image that's too large.
4. Remove the popup asking the user how to handle JSFL files opened from the command prompt.
5. Remove the popup when converting an FLA file to XFL.
6. Remove the popup when Animate is unable to open a file.
7. Remove the popup when Animate is missing a font required by a file.

Option 1 - Download the patched Adobe Animate 21.0.5.
- Download link:
  https://drive.google.com/drive/folders/17hgz4fbIqYetvxHh2MX1KdTP6efIauUU?usp=sharing
- Unpack the zip file (password: iwtcird) and run Animate.exe.

Option 2 - Manually apply the patches.
You'll need a hex editor.
- HxD: https://mh-nexus.de/en/hxd/. This is a hex editor. If you're using the short version, you'll use this to edit hex files. If you're using the long version, this is a convenient way to get a hex pattern for a string in Animate.exe.

Diff the uncompressed "Animate.exe" file with the "Animate - original.exe" file to get a list of patches that need to be applied. Apply the patches to your own version of Animate.exe.
- For Adobe Animate 21.0.5, you can download both the patched Animate.exe and the original Animate.exe here:
  https://drive.google.com/drive/folders/17hgz4fbIqYetvxHh2MX1KdTP6efIauUU?usp=sharing
- For other versions of Adobe Animate, ask in the thread.

Option 3 - Generate the patches.
If you're going to do this, mention it in the thread and ask for help if you get stuck. This section will walk you through the process of running a debugger to get rid of an alert in Adobe Animate. You'll need to install x64dbg in addition to HxD:
- x64dbg: https://x64dbg.com/#start. This is a debugger.

First, you'll need to attach the x64dbg debugger to Animate.exe. To do that…

- Install x64dbg and run it as Administrator. (Right click, Run As Administrator)
- When it opens, the first thing you should do is go to Options -> Preferences and uncheck everything.
- Then go to File -> Open and run Animate.exe from your Program Files directory. Always look at the bottom left corner to see if it says Paused, the debugger pauses a lot from the many exceptions that appear. Whenever the debugger pauses, you can resume execution by clicking Run (blue arrow icon).

All of the default exceptions are useless. You can get rid of them as follows:

- To get rid of the breakpoints, click on the Breakpoints tab and delete everything there.
- To get rid of other exceptions as they come up, whenever an exception appears, go to Options -> Preferences -> Exceptions and click Add Last. This will prevent that exception from coming up again.

Keep adding exceptions to your ignore list and clicking Run until it stops doing that. At this point, you should be able to load an FLA file into Adobe Animate.

Now we'll want to make sure that the popup code is loaded into memory. Pick any file that causes the unwanted popup. Keep clicking through Run and adding exceptions until it loads, then close the file. Again, keep clicking through Run and adding exceptions until you're on the screen where you can load a file again.

Now we'll need to add a breakpoint to pause Adobe Animate when we're in the function that calls the popup. To do that…

- Open up Adobe Animate in HxD and search for the string "is no longer supported in this version". You'll see the string on the right-hand side and the corresponding hex in the left-hand side. Copy the hex string for the entire paragraph.
- Go to the Memory Maps tab in x64dbg. Right click anywhere and select Find Pattern. Paste the HxD pattern in. Double-click through each result until I find the one that matches the "ActionScript @0" string you see in HxD. You should only see one result, but if you see multiple then you can follow the next step to set a breakpoint for each one.

- The memory location should be highlighted in the bottom pane of x64dbg. Right-click any line of the pattern in the bottom pane, then select Breakpoint -> Hardware -> Access -> Word. This will pause Adobe Animate whenever it tries to access this string.
- Open any file that causes the popup. Now x64dbg will break on whatever function is using the "ActionScript @0 is no longer supported" string.

Now open the Call Stack tab in x64dbg. We'll need to figure out which function in the call stack is the one that shows the popup.
- Double-click through each of the lines that says "animate" in the Comment column. This will bring you to the "return" location of that call, which is one instruction below the actual call.
- The first few are all doing some basic string operations, so ignore those. You can ignore all of the ones that use "memcpy" or string-anything in the same function.
- The first one that's not doing a string operation is the one to delete. When you open that function, x64dbg will highlight the first instruction after the call we want to delete. So scroll up by one instruction to find the call we're going to delete.

Once you've highlighted the call you're going to delete, you can delete the call like this:
- Make sure the call instruction is highlighted.
- Press space to get the assemble window.
- Put "nop" in the box
- Make sure Fill with NOP's is checked, or you'll only create one nop and the rest of the leftover call code will cause Animate to hang.
- Confirm, then close the box.

It's patched. Now test it opening another ActionScript 2.0 file. If everything worked, the dialog box shouldn't appear when you open it.

Repeat the process for the case where a bitmap is too large to export:
- HxD string to search: "The bitmap is too large. The largest bitmap that can be created is"
- File to open: https://drive.google.com/file/d/1sJqHDYQfjVyblxZkTIDZ4HdN0iDZ9DBb/view?usp=sharing

- You can trigger the popup by running the AnimationExtractor JSFL script on the file.
- The relevant function in the call stack makes a reference to "BitBuffer". Once again, after double-clicking the relevant call in the call stack, go up 1 instruction and replace the call at that location with NOP.

Repeat the process for the case where a file cannot be loaded:

- HxD string to search: "An error occurred opening file"
- File to open:
  https://drive.google.com/file/d/1_xAfAOX9bsS6oEYIcLwmNY6p6FY5FVE7/view?usp=sharing
- You can trigger the popup by opening the file.
- The relevant function in the call stack has a call further below to "BCString" something. There's a call to "animate.something" between the BCString call and a basic_something call. Replace that call at that location with NOP.

The process is similar for all of these files:

- https://drive.google.com/drive/folders/1xW1oCWmM8Kqwd6-my_neImGwpGuPm98-

Save the patch by pressing Ctrl + P and clicking Patch File. Name it Animate.exe and save it on your desktop or something. Rename your original Animate.exe something else and move the modified Animate.exe to your Adobe folder. You don't want to overwrite the original, keep it as a backup.

The file you just saved is the patched version of Animate.exe.

Progress

Derpi Tag Dataset

This task is complete. You can download the results here:
- https://drive.google.com/file/d/11ZUkUc7q3jSYOpbkve3JgdnaGXLOzLVG/view?usp=sharing

We're using images from Derpibooru to bootstrap our animation dataset. The completed animation AI will let anons direct pony animations using action commands, similar to emotes in

video games. The set of actions ponies can perform is very complex, and we can't manually specify all of them. We're using Derpibooru labels as a starting point to find relevant actions.

The problem is that Derpibooru labels are very noisy, and they cover a lot more than what we need. There are about 25,000 candidate tags associated with animated images in Derpibooru, most of them useless for our purposes. We need help identifying the tags that are useful for describing character puppet movements.

Synthbot has created a tool for labeling tags, which you can find here:
- https://synthbot-image-labeler-sparse.s3.eu-west-2.amazonaws.com/index.html

To use this, you'll need to upload an image list. You can find the Derpibooru image lists here:
- https://u.smutty.horse/lvzbvnnfwwp.zip

UI controls:
- Left click on an image to toggle between "Keep tag" and "Reject tag". You can single-click to immediately accept a tag that has not yet been selected, and you can double-click to immediately reject a tag that has not yet been selected.
- If a tag seems like it might be useful but a particular image has quirks to make it not useful, you can select "Reject image".
- Middle-click an image to view other animated images with the same tag in Derpibooru.
- Don't worry about whether an image is show quality. The goal is only to label the tags as "good for describing the character's motion" or "not good for describing the character's motion".
- Prefer to err on the side of keeping tags. It will be much easier to remove extra tags than it will be to add in missing tags.
- The "Hide Completed" button will clear the page of any completed images. They'll still be exported normally, so you don't have to worry about losing your labels. The button is there to help you quickly see any images that you might have missed.
- Make sure you export your data before loading a new image list. Every time you load a new image list, your previous labels are forgotten.

Clipper has labeled most of our 23,000+ tags, and he should be done with the rest soon.

Image Generation

At this time this is really early on and we are still figuring things out.

Tutorials

Scraping Images

Right now there are two scraping scripts available for gathering metadata:

Clipper's script

Anon's script

Anon's script URL only

Anon's Script

Download one of the scripts here:

Anon's script

Anon's script URL only

The full script will record the image ID, the tags, the score, and the location of the full resolution image. The URL only version will only record the image locations.

There are 5 configuration options:

```
booru = 'ponerpics.org' #Booru to scrape
searchTerm = 'pony+%26%26+created_at.gte%3A2021-01-01+%26%26+created_at.lte%3A2021-03-01' #Search term as it appears in URL
filter = '2' #Filter to use
pageDelay = 0 #Wait between requests
saveLocation = 'metaScrapePonerPics.txt' #Where to save data
```

booru: This sets what site will be used for scraping. By default it is set to PonerPics.

searchTerm: This sets what search term will be scraped from the booru. The easiest way to generate this is to do a regular search on the booru and copy the string from the URL bar. Do not copy sort parameters. See example below:

filter: Sets what filter will be used when searching. By default it is set to PonerPics's everything filter.

pageDelay: Allows you to wait a set amount of time between making requests to the booru. Enter the time here in seconds. Note that this will slow down scraping.

saveLocation: Sets what txt file you want the scraping results saved to.

Clipper's Script
TBD.

Tagging Images

Anon's Plot Tagger

Download

TBD.

Tagpls

TBD.

Progress

Stable Diffusion models

As a free alternative to Dall-e there's the Stable diffusion models, for offline version it's recommended to use the AUTOMATIC1111 github user interface (LINK) as it's currently the most up-to-date and filled with useful features. This tool will require you to have a GPU with minimum 4GB of memory (follow the official automatic1111 github wiki on how to set it up).

SFW / NSFW Pony SD model by Cookie

- Link
- v2 Link
- v3 Link

SFW Pony SD model by AstraliteHeart

- Link
- v2 Link

Pony Soup (model merge)

- V2 Link

MLP Background

- V1 Link

List of Colab Scripts

StyleGAN2 Training and Synthesis Script: Colab

Use AI to create pony images. (Source: >>35339733)

BigGAN/CLIP: Colab

Uses AI to generate images according to a textual prompt.

Finding Pony Neurons in CLIP: Colab

Dall-E notebook: [Colab](#)

      Create images from text.

Deep Daze notebook: [Colab](#)

      Create images from text.

Ai Music Generation

Text to Music - (Online) [Colab](#) | (Offline) [Github](#) + [Huggingface](#)

Created by the github/huggingface user sander-wood, it uses the 'bart-base' fine-tunes on 282,870 English text-music pairs, where all scores are represented in ABC notation.
At the current time the model is not trained on any pony music. The collection of pony midi files is work in progress.

This model is pretty simple in the use as it only requires one/two sentence text input alongside the Note Length, Meter and Key. Example:

This is Alternative rock music with vocals.
Note Length-1/8
Meter-4/4
Key-C

[Colab](#) - ([ipynb](#) backup)

To use it online, just click on Colab link above, set up the Colab to use GPU by going to the tab 'Edit' -> 'Notebook settings' -> 'Hardware Acceleration' (however the code is also set up to also work on the slower CPU as well).

Run the code from top to bottom until you met the cell "*EDIT THE input_text.txt BEFORE RUNNING CODE BELOW*", change the text inside the file to what you wish to generate and continue to the next cell.

You can copy the output terminal text generated under "*#### OUTPUT TUNES ####*" to the https://ldzhangyx.github.io/abc/ to preview the "abc" file, and then download it from the "*output_tunes*" folder.

Offline - Github + Huggingface

Download and unzip the github repository, then inside of it create folder 'sander-wood' and inside of it create folder 'text-to-music'. Download all the huggingface files inside the 'sander-wood/text-to-music'.

Install the python requirements:

pip install torch==1.9.1+cu111 --extra-index-url https://download.pytorch.org/whl/cu111
pip install Unidecode==1.3.4
pip install samplings==0.1.7
pip install transformers==4.18.0

Edit the inside of the 'input_text' then run 'python run_inference.py' to generate an ABC file.
You can listen to it using the https://ldzhangyx.github.io/abc/ website and/or convert it to midi file with this program https://sourceforge.net/projects/easyabc/ .

MusicGen (online & offline)

Created by Meta/Facebook, currently there is no ability to make custom trained models. Multiple models in different sizes available. It can create instrumentals only, in many genres, however it will constantly try to compose a mix of multiple instruments (e.g. a text prompt of "rhythmic drum beat only" will still end up with other instruments being played in background).

Features:
Create new music by providing a descriptive line of text.
Converting a reference music clip into a different musical genre.

HuggingFace - Link
Colab - Link

Github (offline) - Link

For offline use, you will need at least an 8GB VRAM nVidia card to use the 'small' model without running out of VRAM space, for larger models it is recommended to use 16G VRAM.
While it is possible to use a CPU cuda (and therefore use the larger models without having a beefy GPU), generating audio with that method will be VERY slow.
Example (using 'small' model):
Using the Gpu to generate 30s of audio from text, requires about 2 minutes of processing time.
Using the Cpu to generate 5s of audio from text, requires about 1m 30s of processing time.

Conda/Miniconda environment installation instructions

#download and extract the zip of the project from MusicGen github project

cd [directory of were zip folder was extracted]

conda create -n muGen-gpu python=3.11
conda activate muGen-gpu

#FOR GPU VERSION, this step can take over 20 minutes to execute the installation
conda install pytorch=2.0.1=py3.11_cuda11.8_cudnn8_0 torchvision=*=py311_cu118
torchaudio=*=py311_cu118 cudatoolkit=11.8 -c nvidia -c pytorch -c conda-forge

#FOR CPU VERSION, (this is not compatible with the above installation, I would recommend having installed two completely separate environments in case you wish to use separate version for the GPU and CPU for MusicGen)
conda install pytorch=2.0.1 torchvision torchaudio cpuonly -c nvidia -c pytorch -c conda-forge

conda install jupyter

pip install -U audiocraft # stable release
pip install -e .

#change the pink text to directory of your browser you will use as the UI, alternatively just use command 'jupyter notebook' to let python use whatever default browser your system has set up
jupyter notebook --browser='C:/Users/User001/AppData/Local/Programs/Opera/launcher.exe %s'

#once the the jupyter UI opens in the browser run cells the same way you would run colab script cells
[run demo.ipynb]

Other

List of Colab Scripts

Talk to /mlp/: [Colab](#)

From the notebook: This is a Colab notebook with a GPT-2 text generating model trained on a 100MB worth of a dump of /mlp/'s posts from its inception to June 2019.

GPT-J-6B PNY: [Colab](#) ([Guide](#))

This notebook allows you to generate text using a finetuned version of the 6B parameter GPT-J model from EleutherAI

ESRGAN Notebook: [Colab](#)

Image super resolution similar to Waifu2x.

DeepDanbooru notebook: [Colab](#)

Uses AI to automatically tag images. Includes dataset scraper, training script, and synthesis script.

Interacting with Jukebox: [Colab](#)

AI music generation?

LLaMA 2 13b MLP LoRA: [Colab](#)

(options are to use text ai trained on 'desu archive mlp posts' or trained on 'fimfiction stories')

(VALL-E-X) Creating dataset with ai from 10s source

https://huggingface.co/spaces/Plachta/VALL-E-X
https://desuarchive.org/mlp/thread/40503961/#40518915
https://desuarchive.org/mlp/thread/40503961/#q40520924

With the use of voice cloning tool VALL-E-X, it is possible to synthesize an artificial dataset as long as a minimum amount of ~10 seconds of clear audio data of the character voice is available.

As presented in the above link, to train a ".npz prompt model" using only 10 seconds of audio clip in the Make propt' tab. This is the most important part of the process, if the primary prompt fails to generate a sound-like replica of the character you will have to keep re-making a new prompt (this process may take anywhere between 20 minutes to an hour).

 Once you have a satisfactory .npz file, apply this prompt in the 'Infer from prompt' tab to generate a TTS audio clip. You will need to test different 'Voice presets' to find the one that works best with the character clips you are trying to create (bronya_1 seems to work best with pony voices).

The most obvious positives of this process is the ability to generate almost any dataset for any character as long as you can extract a 10 seconds of clear audio.
The negatives are that this process will take a long time, as about 1 out of 20~50 generated clips may be considered to sound In-Character, this process will also limit the vocal range as the best quality output audio will be limited to the tone of the voice from the original 10 seconds clips.
Additionally this process is limited to synthetising talking clips only so a singing dataset may not be possible to be created with it.

       Installation steps (based on the https://github.com/Plachtaa/VALL-E-X) with the conda terminal:

```
cd (folder-were-you-want-install-and-run-this-code)

conda create --name vall_E_X python=3.10

conda activate vall_E_X

git clone https://github.com/Plachtaa/VALL-E-X.git

cd VALL-E-X

pip install -r requirements.txt


#use below to run it

python.exe launch-ui.py


##by default it will use the CPU (aka will work on all computers but slowly), install conda pytrch
to use GPU for faster generation

conda install pytorch==2.0.0 torchvision==0.15.0 torchaudio==2.0.0 pytorch-cuda=11.7 -c
pytorch -c nvidia

####trying to run now will give you error about c10_cuda.dll

pip install --upgrade --ignore-installed numpy

pip install chardet

pip install -r requirements.txt


#use below to run it (it will take few minutes to load model), it will open the UI in default browser

python.exe launch-ui.py
```

Submitting Your Content

We do not have an official solution for sharing contributions at this point. For the time being, put
your files (audio, text transcriptions, modified Audacity labels, etc) in a Zip archive, upload it,
and post a link in the thread. Explain what it is you're uploading, and make sure that the files are
sorted into folders by season and episode (or by source if not from the show).

If you're uploading cleaned audio from a noisy clip that you made, upload both versions of the clip. We will be archiving both the original and a cleaned version so that anons can create their own cleaned versions if they feel they can do better.

Some suggested file hosts:

https://smutty.horse

https://zippyshare.com

https://nofile.io

https://anonfile.com

https://mega.co.nz

https://dropbox.com

https://drive.google.com

Our Plan (WIP)

Machine learning works as the name implies: you create a program (often a neural network) that can learn from information you give it and "teach" it what outputs it should return for a given input. There are already a large number of open source machine learning projects available to use as a foundation, including several specifically designed for what we are trying to do (e.g. Google's Tacotron). This Anon explains it in more detail:

Anonymous 04/09/19(Tue)18:23:58 No.33723583 ► >>33724052

>>33723516
>>33723398
For anyone wondering:
An overview of the common elements of deep voice and tacotron are: it takes a string of characters of a sentence, maps it to characters that should naïvely always sound the same using a dictionary, (basic speach programs just play a sound based on this, which sounds unnatural), this string of characters is then sent through an auto encoder with both (double sided) memory and attention, meaning it takes the data and tries to extract a smaller set of features out of it, that completely describes it, it has memory because each neutron keeps some bits of previous information and uses it to change its current output, and attention means it tries to keep the output tempo in sync with the input characters, so doesn't repeat sounds, or skip words. This smaller set of features is then used to generate a spectrograph, which is a graph of each frequencies intensity at every point in time (naïvely each part of this graph should be the same for each input characters, but we know sounds change based on what sounds come before and after it). so each input generates a list if frequencies with their intensity, which can depend on sounds that came before and after it due to the memory, and which is the proper length of time due to the attention. These spectrographs are then fed to a second network which uses convolution (mixes neighbouring data points recursively till you have a much smaller set) to eventually generate a single value, which is the wave amplitude, this is done for each time step in the spectrograph, till you generate an entire wave that should in theory be a wave that produces the given spectrograph, which should be the voice corresponding to the fed in characters. To get this all to produce the correct voice we train it by giving it a set of labeled voices, and calculate the error, the nural network can take the error and calculate which values in it needs to be changed to get closer to the correct answer, and repeat.

A project such as Tacotron could be used to produce a proof of concept, but first we need a LOT of audio to work with. Dialog has already been extracted from MLP:FIM seasons one through eight in the highest quality available. There is a torrent available for this in the "Resources" section of this guide.

Threads

Tools

- [Audacity](), for clipping audio
- [Praat](), for annotating audio clips with timed transcriptions and [ToBI]() events
- [iZotope RX7](), for noise removal
  - [ISSE]() is a possible free alternative
  - iZotope RX is not free. However (and I definitely don't endorse this), cracks *do* exist. I won't provide you a link because we want to stay under the radar. Just avoid R2R and CracksNow, they have been known to include ransomware.
- [Anon's SRT to Audacity App](), for converting SRT subtitle files to Audacity labels
- [14aren](), a nice file renaming tool for Windows (for after-the-fact naming adjustments)
- [mlp_dialog_rip.sh](), the shell script used to extract dialog and subtitles from S01-S08. It applies the process described by [this page]() using the ffmpeg utility
- [Tacotron Text Transcription Tool](), for easily transcribing datasets into something usable by Fifteen.AI or the Google Colab notebooks. [Check out a visual demonstration here of how to use it.]() Hotkeys are End to play audio, Page Up to go back a line, Page Down to go to the next line. Ctrl+S to save. Progress is saved on a line every time you switch what file you're transcribing. [Source code is available here.]()
- [Google Speech-To-Text -> Dataset Processor]() Spits out a dataset.txt formatted correctly for the system. Needs a Google Cloud SDK installation to function. **This shit costs money to use after the first $300, so make sure to let google data harvest you with Data Logging to reduce that overhead.** However, it's generally pretty accurate. Use this if you've got a gigantic pile of speech from, say, a video game, and no transcription available. Works better with longer sentences.
- [Anon's Audacity Label Timestamp Script](), a PowerShell script to add timestamps to an existing set of Audacity labels
- [Anon's Character Tagger](), an attempt to automatically tag characters in an Audacity label file by comparing it to transcripts available on the mlp wiki. Due to differences in spelling and abbreviation between transcripts and subtitles, will not be able to tag all lines but should be able to do most.
- [Notepad++](), a programmer's text editor for Windows, for manually editing Audacity label files

- [Anon's Transcript Generator](#), a [Python](#) script for generating transcript text files (instructions available at 33:25 in [this demo video](#))
- [Anon's Checking Script](#), a [Python](#) script that checks for common formatting errors in label files and automatically replaces shorthand character, emotion and noise tags with their full versions.
- [Twibot's Working Demo](#), A functional demo of our current progress
- [Pony Sorter](#), A program to sort the different clip versions
- [Twibot's Audio Project Utilities](#), used to extract audio from the dataset
- [Cookie's Custom Processing Script](#)
- [Synthbot's Tools](#)
- TKinterAnon's GUI Synthesis (local synthesis):
    - Note: version 1.1 is an easier setup than 2.0, may just want to use 1.1 if only interested in colab voices.
    - [Version 2.0 full](#) (>>35283074)
    - [Version 2.0 15 only](#) (>>35283074)
    - [1.1 Patch](#)
    - [Version 1.0](#) (>>35069149)
- [ARPAbet Converter](#)
- [Audacity2Transcript](#), Turns exported audacity labels into Tacotron2 text
- [AutomaticWavsTranscriber](#) Using VOSK it can transcribe the dataset files offline on CPU
- [22KHz to False 48KHz Tutorial](#), helps the quality of 22KHz models
- [DeltaVox](#), a 15.ai client
- [Synthbot's Patched Animate](#), anon- and automation-friendly for working with FLA files

- [Colab script to convert audio sampling rates](#)
- [Colab sample script to access derpibooru image and tag data](#)
- [Synthbot's 'Fimfarchive tools - release v4.ipynb'](#)
- [Synthbot's 'Fimfarchive tools - release v4.ipynb'](#) upgrade (search by publishing date)
- [Ultimate Vocal Remover](#) ([link v2](#)) - Useful tool in separation of vocals and instrumentals from songs (more models available [here](#))
- [Blob's blender 3d models of mane6](#) - It's not voice or even AI related (other than being used as part of Blob unreal engine game project), however it could be potentially useful in 3d animating.
- OpenUtau + Diffsinger model - [OpenUtau 0.1.547](#) - models: [Twilight](#), [RD](#), [AJ](#), [PP](#), UTAU (aka, freeware alternative to Vocaloid) allows to create singing audio files (it's not exactly a TTS model as it's more specialized in using vocal lines in form of midi instrument notes than plain text).
- ACE-Step - [Github Link](#) (online Colab Notebook file included) - experimental ai song editing tool, it allows to create new ai songs with use of positive songs tags and lyrics OR editing existing lyrics in the song clips. Quality depends on the amount of changes inflicted by the user (with more changes leading to worse sounding outputs), the end results will vary.

Resources

Alternate subtitle source if things don't seem right (Stange doubling):

https://www.addic7ed.com/

Anon's In-Depth Mane 6 Vocal Profiles:

https://drive.google.com/drive/folders/12wXl80FDHpJTXSE5Sw5fF_6Fm0GxtC6V?usp=sharing

(Old) Project Logo Photoshop File:

https://files.catbox.moe/j50wq1.psd

iZotope Software:

https://mega.nz/#F!ejJmAChZ!wdgeVS1cGKp0DPgHXU7kgQ

Old Doc:

https://docs.google.com/document/d/11vmRdCFnl_XgKB2XwB82nBAdmC0hfoldzlAeiPB06SM/edit

Old Contribution Spreadsheet:

https://docs.google.com/spreadsheets/d/15UPAhWr8afJIBk6QWMrjUhI4ZT8enMws9A0u9KBCVuU/edit#gid=237082183

Open Unmix Audio:

https://mega.nz/#F!swgmBA6b!mCmJ3jt8SKD4NnVvampNpg

Phonetics Guide:

https://docs.google.com/document/d/1gr4HluP9c38Qep1Q2P_OAsW9XmFcK2WRF3Ki4ep2S_g/edit

Con Doc 2020:

https://docs.google.com/document/d/1QIwzaJBDrYXEBozzgKPfMphmKsYiFVJLhwJTiAeJ1WA/edit

Datasets

Datasets not in Clipper's Mega

For a more complete archive, check here.

Adachi Tohru (>>35454602): File House

Administrator (>>35075848): GDrive

Ai Ebihara (>>35454602): File House

Announcer (>>35072189): File House

Ayane Matsunaga (>>35454602): File House

Blaze the Cat (>>35079451): GDrive

Chie Satonaka (>>35454602): File House

Dan Vs. (>>35079026): GDrive

Kanji Tatsumi (>>35454602): File House

Littlepip (>>35075476): GDrive GDrive

Margaret (>>35454602): File House

Miss Kashiwagi (>>35454602): File House

Mister Morooka (>>35454602): File House

Nanako Dojima (>>35454602): File House

Naoto Shirogane (>>35454602): File House

Postal 2 Dude (>>35072009): Mega

Rise Kujikawa (>>35076495): GDrive File House Pastebin

Rise Kujikawa (>>35454602): File House

Ryotaro Dojima (>>35454602): File House

Soldier (>>35082707): GDrive File House File House

Taro Namatame (>>35454602): File House

Teddie (>>35454602): File House

Yosuke Hanamura (>>35454602): GDrive

Yukiko Amagi: File House

Yumi Ozawa: File House

Other

Audiobooks:

David Tennant, John de Lancie and Emily Blunt: Mega

Interviews:

Lauren Faust: Mega

Colab Scripts 02/22/20: Mega

Tools:

SRT to Audacity CAPX: Mega

Character Tagger CAPX: Mega

Open CAPX with Construct 2: Scirra

Text:

Fimfiction groups data: [Link](#)

Clipper's Mega Snapshots

08-01-20: [Mega](#)

05-19-20: [Mega](#)

03-09-20: [Mega](#)

01-09-20: [Mega](#)

12-21-19: [Mega](#)

12-04-19: [Mega](#)

11-15-19: [Mega](#)

10-03-19: [Mega](#)

09-16-19: [Mega](#)

07-31-19: [Mega](#)

07-12-19: [Mega](#)

Synthbot's Torrent Resources:

Clipper's Clips:

Latest backup: [Google Drive](#)

Master File 2.0 (old): [Torrent](#)

Master File 2.x (old): [Torrent](#)

Old Torrents: [Torrent](#)

TKSynthesizer and models:

magnet:?xt=urn:btih:ef33561c15c55abf8795f11785e8a6a6d54704b5&dn=TKSynthetizer

(Source: >>35367799)

Changelog

-02/20/21: [Added AI Image Generation section](#)
-10/03/20: [Added What Can I do with the AI? section](#)
-08/06/20: [Added Synthbot.ai section](#)
-08/06/20: [Major reorganization of doc](#)
-07/25/20: [Added Open Unmix section](#)
-07/21/20: [Added OP Template section](#)
-07/21/20: [Moved Audio Samples to separate doc](#)
-07/15/20: [Added Developing Animation AI section](#)
-07/13/20: [Added Making NGroks sing section](#)
-07/07/20: [Updated How to Contribute section](#) and general reorganization
-06/26/20: [Added Inference Server (Synthesis) section](#)
-06/22/20: [Added Collected YouTube Tutorials section](#)
-06/14/20: [Moved Tools to before Resources](#)
-06/14/20: [Created new How to Contribute section](#)
-06/14/20: [Renamed old How to Contribute section to Creating Audio Dataset](#)
-05/22/20: [Updated Using TKinterAnon's GUI section](#)
-05/21/20: [Added RTX Voice section](#)
-05/18/20: [Added Automatic Clipping and Transcribing section](#)
-03/14/20: [Big overhaul of the AI section](#)
-02/22/20: [Added Making the most of the AI section](#)
-02/22/20: [Added 48KHz instructions](#)
-02/07/20: Added page breaks
-02/07/20: [Added Using the Character Tagger section](#)
-01/29/20: [Added WiP Scripts section](#)
-01/07/20: [Added TacoTron2 Models section](#)
-01/05/20: [Added dataset processing scripts to tools](#)
-12/29/19: [Added AI Section](#)
-12/27/19: [Added links to original art posts](#)
-12/15/19: [Resource Section cleaning](#)
-12/10/19: [Added Audio Samples section](#)
-12/05/19: [Filled in Cleaning up Audio section](#)
-12/04/19: [Added video links to Cleaning Up Audio](#)
-12/04/19: [Added Pony Sorter to the tools list](#)
- 12/04/19: [Added sorting audio section](#)
- 12/04/19: [Added Open Unmix Mega to resources](#)
- 11/16/19: [Added Clipper's Mega Snapshots](#)
- 10/20/19: Added link to phonetics guide
- 10/19/19: [Reorganized resources](#)
- 10/19/19: New maintainer

- 05/16/19: Updated clipping instructions
- 05/16/19: Updated video demonstration link in clipping instructions
- 05/16/19: Replaced file naming conventions with updated image from thread
- 05/16/19: Updated link to Anon's Transcript Generator in Tools
- 04/27/19: Added new series of video tutorials to Clipping Audio
- 04/27/19: Added Anon's transcript generator to Tools
- 04/20/19: Implemented various suggested changes to clean up the guide somewhat
- 04/18/19: Added Anon's narrated video demo to Clipping Audio
- 04/16/19: Added new tracker to S1-S8 dialog torrent file, replaced old link
- 04/15/19: Added links to contribution spreadsheet
- 04/15/19: Added Anon's video demo to Clipping Audio
- 04/15/19: Added list of episodes being worked on/complete to How to Contribute
- 04/14/19: Added logo Photoshop file to resources
- 04/14/19: Added officially endorsed file naming scheme
- 04/14/19: Added alternate subtitle source to resources
- 04/12/19: Added clarification about what counts as background noise for marking clips
- 04/12/19: Added Audio Anon's super special lossless movie source to Resources
- 04/12/19: Added Anon's PowerShell script for adding timestamps to Audacity labels
- 04/12/19: Added ffmpeg-based dialog extractor script to Tools
- 04/12/19: Added changelog section
- 04/12/19: Added tip to audio cleanup section for quickly removing noise in RX
- 04/12/19: Clarified how clip beginnings and ends should be decided, and how long clips should be
- 04/12/19: Added tip for easily marking files as noisy, in the case that the majority are

Anon's To Do List:

This is Anon's (the doc maintainer's) current to do list for the doc.

Create persona nerd clipping tool section. (>>35309895 and >>35312835) ([github](#)) ([file house](#))
Delta Vox section.
Update "Our Plan" section
Update "Current Goals" section
Update "Submitting Content" section
Talk to /mlp/ section?
Move changes from [WiP doc](#) into main doc when ready
Do a large run of the DeepdanBooru notebook once I fix my Tensorflow install
Write up proper tutorials for TalkNet, DeltaVox and others

OP Template

Old version:

Last update: 9/25/20

Title: Pony Preservation Project (Thread ##)

OP Image: Smutty Horse

Text:

TwAIlight welcomes you to the Pony Voice Preservation Project!
https://clyp.it/tm03e5en

This project is the first part of the "Pony Preservation Project" dealing with the voice.
It's dedicated to saving our beloved pony's voices by creating a neural network based Text To Speech for our favorite ponies.
Videos such as https://youtu.be/GuJKTodX1FA. or https://youtu.be/DWK_iYBl8cA have proven that we now have the technology to generate convincing voices using machine learning algorithms "trained" on nothing but clean audio clips.
With roughly 10 seasons (9 seasons and 5 movies) worth of voice lines available, we have more than enough material to apply this tech for our deviant needs.

Any anon is free to join, and many are already contributing. Just read the guide to learn how you can help bring on the wAIfu revolution. Whatever your technical level, you can help.
Document:
https://docs.google.com/document/d/1xe1Clvdg6EFFDtIkkFwT-NPLRDPvkV4G675SUKjxVRU

We now have a working TwAIlight that any Anon can play with:
https://fifteen.ai/
https://derpy.me/vCzm2 (48KHz Training)
https://derpy.me/hdJQF (48KHz Synthesis)
https://derpy.me/NR7Xi (Ngrok Synthesis)
https://derpy.me/YTJ94 (Guide)

>Active Tasks
Researching alternative vocoders
Anon transcribing books and comics
Making AI creations (In thread)
Cookie is working on controllable speech
Research into animation AI

>Latest Developments
AI singing
Master file 2 contains BGM and SFX
Groundwork for video generation
15's site is back up
More datasets
15 looking into multispeaker models
Delta released new local synthesis tool (no gpu required)

>Voice samples
https://derpy.me/fHs3K
https://derpy.me/O1xdh

>Clipper Anon's Master File 2.0:
https://mega.nz/#F!L952DI4Q!nibaVrvxbwgCgXMIPHVnVw
https://mega.nz/folder/0UhSmYAB#WBrB-qCprQTofkAhwMp5CQ

>Synthbot's Torrent Resources
https://derpy.me/ZJNca

>Cool, where is the discord/forum/whatever unifying place for this project!?
You're looking at it.

Last Thread:
>>########

FAQs:

FAQs:
>READ THE DOC
Do it now
https://derpy.me/V7cMp

>Did you know that such and such voiced this other thing?
Yes. We are very much aware. It is best to keep to official audio only unless there is very little of it available. If you know of a good source of audio for characters with few (or just fewer) lines, please post it in the thread. 5.1 is generally required unless you have a source already clean of background noise. Preferably post a sample or link. The easier you make it, the more likely it will be done.

>What about fan-imatitions of official voices?
No.

>How do I make the voices?
Several guides are available. In depth guides on how to do training and synthesis (making the ponies speak) are in the doc. If you don't want to use the navigation bar in the doc, the sections are also directly linked in the OP. If you want to use the WiP 48KHz notebook, some kind Anons have put together some image guides for you.
48KHz Training: https://derpy.me/wW2hX
48KHz Sythesis: https://derpy.me/j4MXQ

>Where are all the voice samples?
In the doc.

>Is a place I can find all the pony models?
In the doc.

>What about muh waifu?
Check the doc.

>Will you guys be doing a [insert language here] version of the AI?
Probably not, but you're welcome to. You can however get most of the way there by using phoenetic transcriptions of other languages.

>What about [insert OC here]'s voice?
Not a priority. Again, however, you're welcome to. There are already people doing this.

>Where can I view the PPP /mlp/con panel?
YouTube: https://youtu.be/WtuKBm67YkI
CyTube chat: https://pony.tube/videos/watch/b83fbbfc-6d4e-4768-8deb-edb61ea38abb

>I have an idea!
Great. Post it in the thread and we'll discuss it.

>Do you have a Code of Conduct?
Of course: https://fifteen.ai/code

>Is this project open source? Who is in charge of this?
[spoiler]https://derpy.me/CQ3Ca[/spoiler]

Anchor post:

Please make an anchor post for updates. **HERE** are some images of anchors.

Fanart

/MLP/ 04/12/19 9:30ᵖᵐ ♂

PONY
MACHINE
LEARNING

GOD SPEED,
(YOU) GLORIOUS
BASTARDS
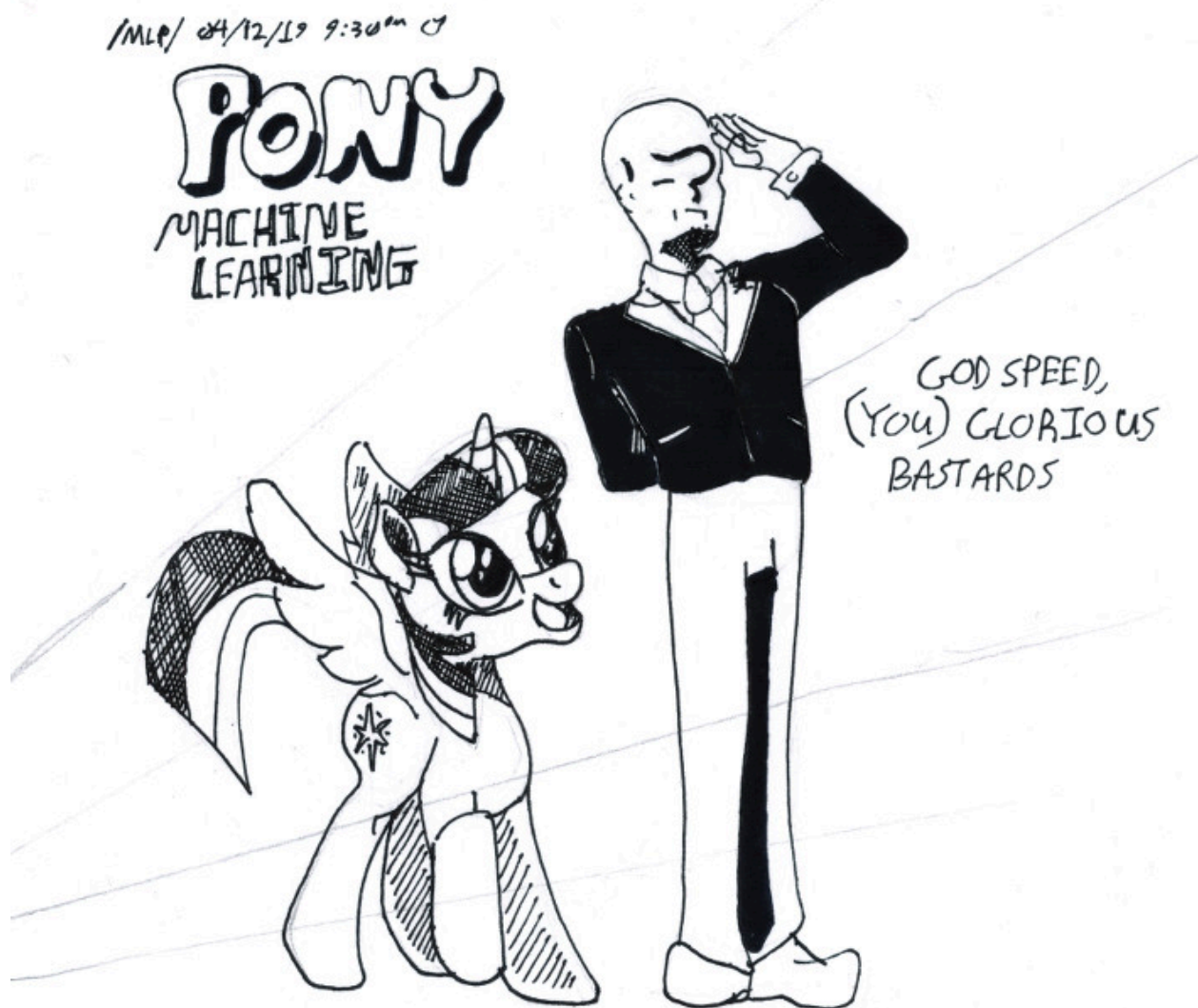
Haypril 14:

The TwAILIGHT test run has successfully recreated its first few words!

>>33746248
Test Run Audio Demos: https://files.catbox.moe/91f1w1.zip
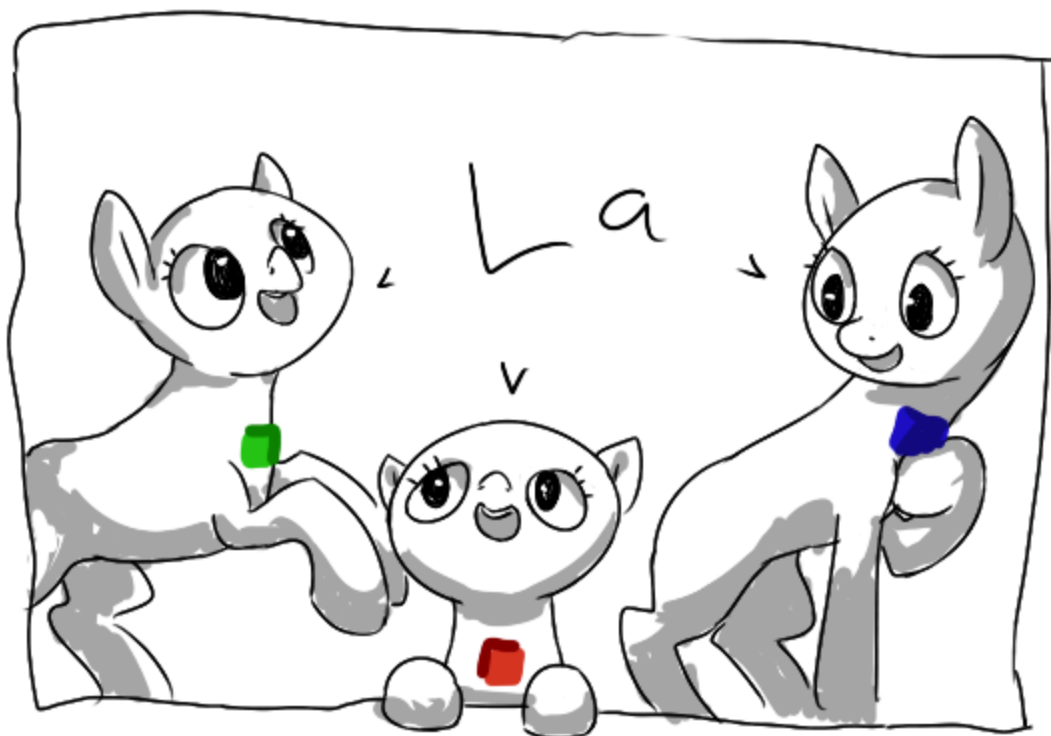
https://clyp.it/mmxgq1yx

>>34006948

>>34012310

>>34482875

>>34611692

>>34749846
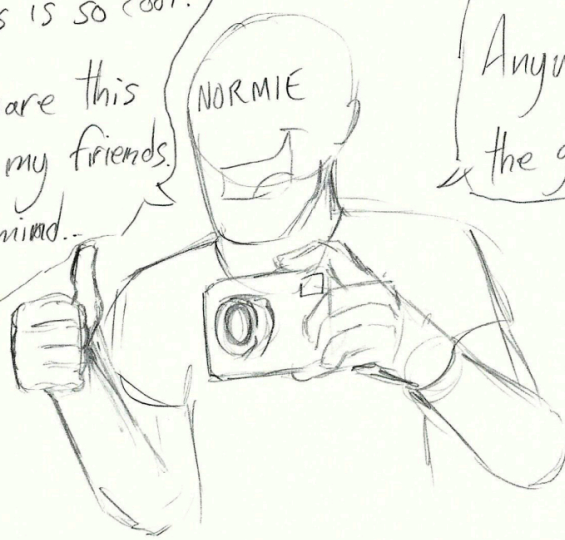
>>34773088

Post count (total 9748)

>>34778376

Hovers close

Hellom. Yaes et Es AY, Being Very Importants permson in charg of projects of Pony voces! I will of happy be questioms for you in yes.Inglais is of secnnding language from ME, speak it very well. whating you be want know do
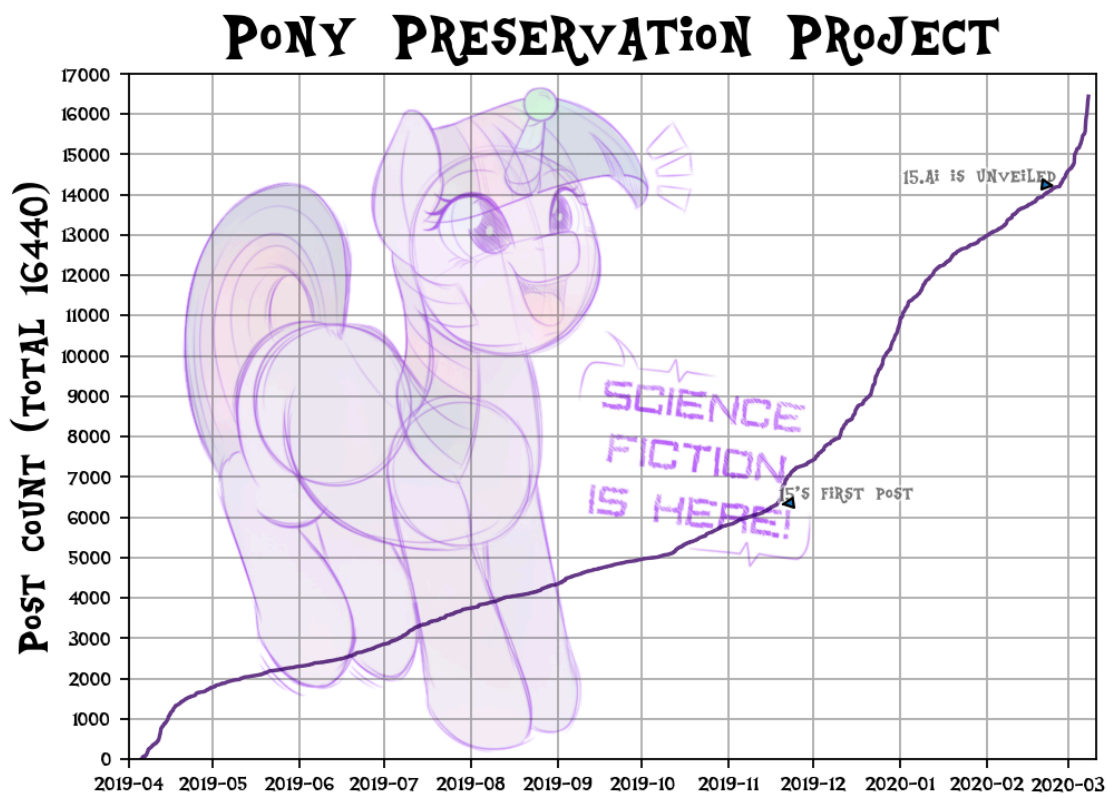
>>34931416

PONY
PRESERVATION
PROJECT

>>35067051

>>35839777