General user’s guide to TensorVox

Requirements

In order to run TensorVox, you need:

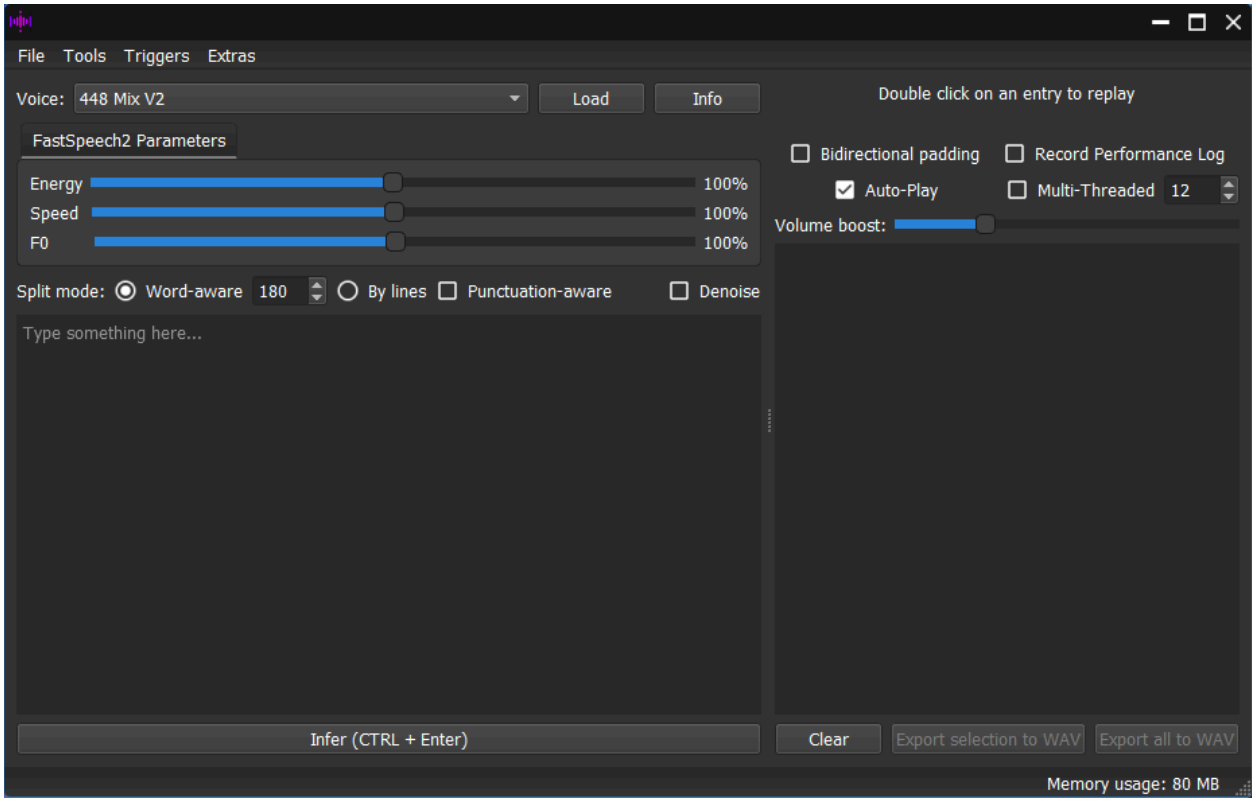
1. 64-bit Windows 10 (although I have heard reports of it working in 8.1)
2. A CPU that supports the AVX instruction set. Anything made in the last 10 years and isn’t a Xeon will most likely do

Notably, no CUDA is required.

Installation

Download the latest release (the 90MB TensorVox.zip file, not the source code), unzip it anywhere, in the TensorVox folder run TensorVox.exe. Before generating speech you will first need some models, you can find those I made public in this Google Drive folder. Inside that folder you will also find the Current models doc file, which will also explain how to install them (it’s easy!)

The following image is one instance of the program without a loaded model: (description in next page)

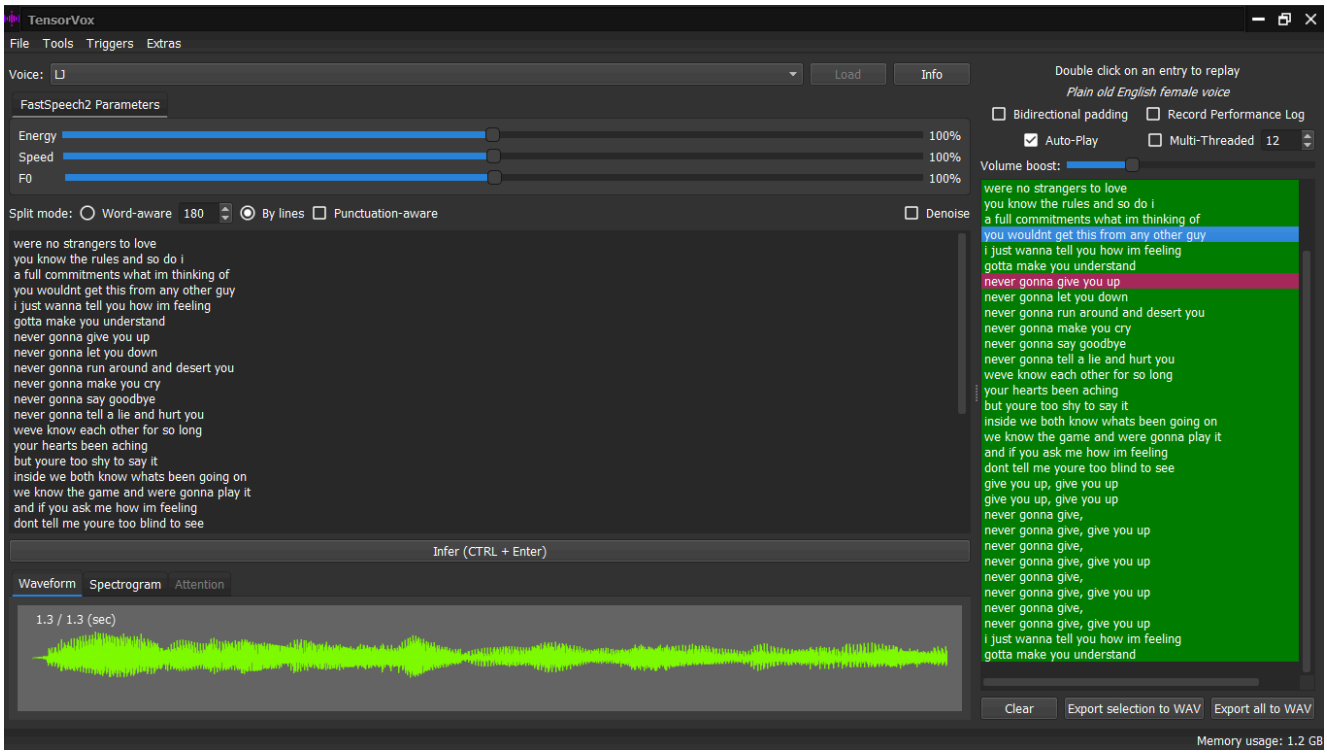


To start, click the dropdown menu next to *Voice* and select one (or keep it as is) and click **Load** to load the model. Note that for every loaded model the program uses 500MB of RAM idle. Also, you don't have to explicitly load the model, clicking the **Infer** button without a loaded model will automatically load it first. If the model is a multispeaker one, once you click Load, the list of speakers will appear.

Note: When you load a model it might freeze for a bit. Don't worry, this is normal; it's loading the model.

Inference

See the *Split mode* over there? Basically, since the TTS engine cannot work on very long input before destabilizing or consuming too much memory, we split it into chunks.



Your sidebar is where the process of your utterances is reported

- **No color** means the utterance is currently waiting
- **Blue** means the utterance is being processed by the engine.
- **Green** means the utterance has been successfully completed. The audio will automatically play and you will be able to replay it by double clicking the entry.
- **Dark magenta** means the utterance is currently playing.

You can click on an individual green entry and click *Export selection to WAV* to export it; or you can wait for all of your list to be green and click *Export all to WAV*, which will take all the entries, join their audios together, and save that to a WAV file. This is especially useful for generating long speeches and saving them. If you hold shift while clicking the aforementioned button, it will export all the utterances separately.

Tip: To stop audio from playing, press the Escape key

Note: The first utterance often takes a bit more time to generate than normal (2x sometimes) as it initializes the engine, but then the rest will be fairly fast.

Split Modes

There are two available long input split modes:

1. **Word-aware.** This goes through words (to prevent a split cutting off a word) and delimits a chunk when the total length in characters exceeds what is specified
2. **By lines.** It's quite self-explanatory, the one you saw in the above image

If you check *Record Performance Log*, by every line you generate, it will save to the internal log which you will be able to export in the **File** menu. This records the Real Time Factor (RTF) metric to measure performance (how quick inference is) compared to the length of the output audio.

FastSpeech2 models are more susceptible to long sentence problems, which is why the auto-split is set to 500 when a Tacotron 2 model is loaded and 180 for an FS2 unless it's manually modified.

The formula for RTF, where ***P*** is the time to process an input of duration ***I***, is the following:

$$RTF = \frac{P}{I}$$

Basically, the lower the RTF, the better.

Phonemes

The text-to-speech engine doesn't directly take characters as an input, but phonemes, specifically *stressed ARPA* (for English) as found in the CMU Dict. For example, *green* is *G R IY1 N.* Other languages may have different formats, but the usage is the same.

When your input is processed, it gets turned into phonemes by a neural network trained on known utterances which also learns to generate spelling for novel words. This process is automatic, but in case you want to, there are two ways of inserting your own phoneme inputs:

1. Wrap it in curly braces (more convenient for many phonemes)

2. Prepend the @ symbol

In both cases, input will be highlighted by the editor in **bold magenta**. This feature is useful for inserting your own pronunciations. If you need some guidance, you can select a certain chunk of the text and hit **CTRL + P**, or Triggers->Phonemize selection, and it will automatically process all the selected words, turning them into explicit phonetic input.

(Next only applies to FastSpeech2)

Additionally, there is a "phoneme" recognized by the model but not present in any dictionary, that is, the **SIL** token, which stands for silence. Commas, dots, and other punctuations are converted to **SIL**s, and you may input it like a normal phoneme.

*There are actually two special phonemes; apart from SIL, there's **END** that specifies the end of a sentence, but it's only supposed to be used during training. Regardless, the model will recognize it if you decide to input it.*

Tip: If a character stretches out a vowel too much at the end, you can append an END at the end of your sentence.

Adjustable Inference Parameters

There are a few inference-time parameters that can be adjusted. All of them are directly fed to the model. **Note that those are only available in FastSpeech2 models**.

- **Energy** is what the FastSpeech architecture defines as energy. It's very hard to explain, so let's leave that adjusting it will lead to subtle or strong changes in tone from the speaker
- **Speed** is self-explanatory, although dramatic changes may make the voice sound unnatural
- **F0** is the fundamental frequency prediction. This can make quite noticeable changes to the voice; for example, high F0 may make the speaker take on a more assertive-sounding tone.

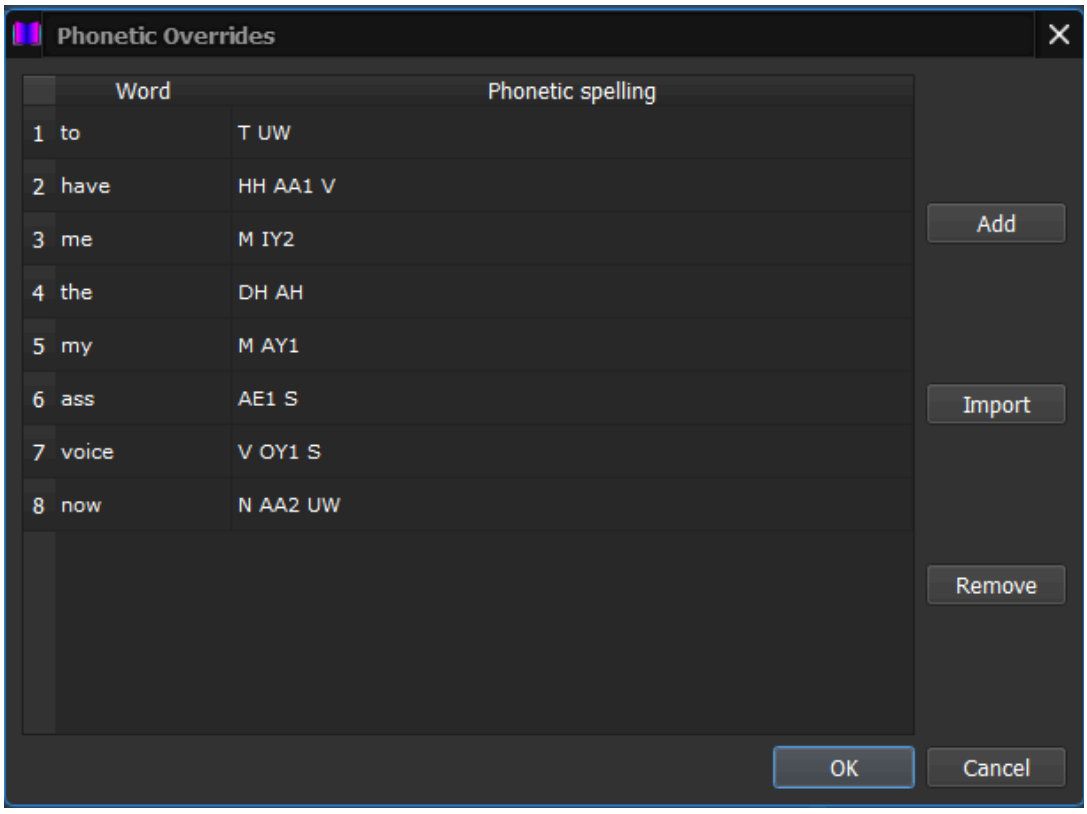
For **Bidirectional padding**, it's a modification to the inference process. Basically, to prevent unnatural sudden cutoffs, the program appends a *SIL* token to the end of each utterance. Normally, it's only at the end, but some models may have problems spelling their

first word, what Bidirectional padding does is that it also adds a silence token to the beginning of each sentence, which can alleviate those problems.

(go next page)

Phonetic Overrides

In the Tools menu you can find the **Overrides** option, which opens the phonetic dictionary.



An example of a window

The purpose of this function is to provide overrides for pronunciations, when doing inference, it finds every instance of a word in the dictionary and replaces it with the spelling. *Why?* Because sometimes the text to phoneme converter doesn't get it right, or even the correct pronunciation according to conventional dictionaries like the CMU one make the model spell words wrong, or maybe we just want more fine control

Therefore, there is a certain workflow for detecting and correcting bad pronunciations: experiment with alternatives, then add it to the dictionary.

The overrides are saved in the same folder as the exe file with the name *dict.phd* (*phd* stands for **ph**onetic **d**ictionary). You can distribute your dictionary and have other people use the import function so they can add entries to theirs. It is saved every time you click OK in the dialog.

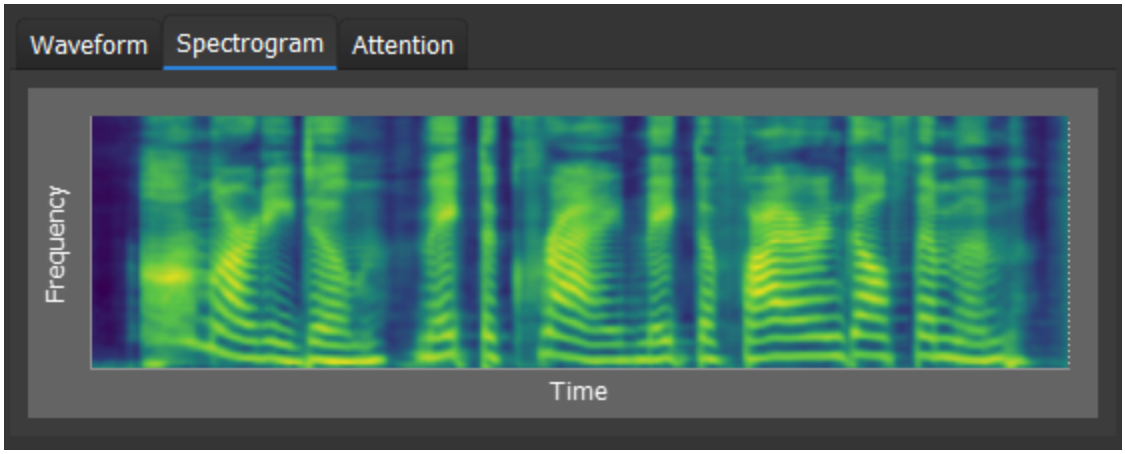
Metrics

When you infer a sentence, a bottom pane will pop up with various options that allow you to investigate your outputs more closely:

Waveform

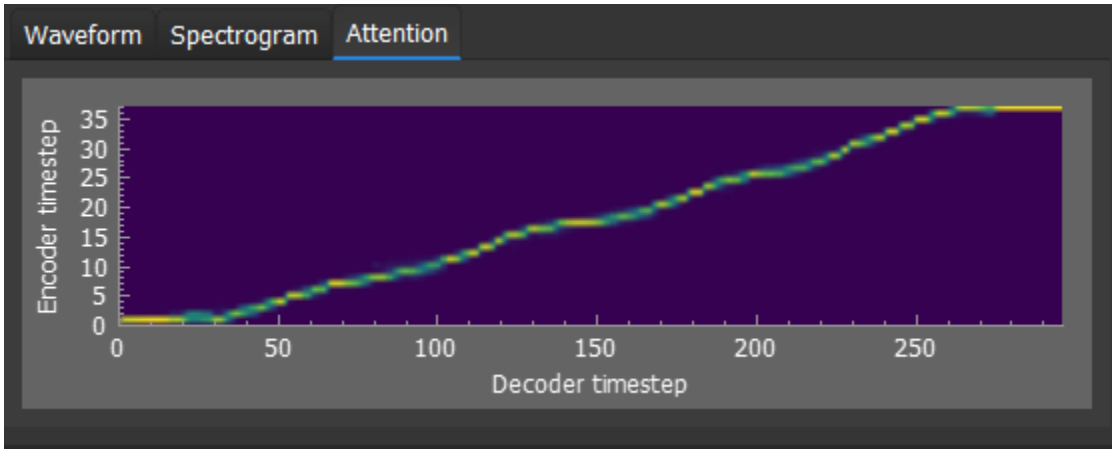
This one's pretty self-explanatory, an Audacity-like representation of your sound

Spectrogram



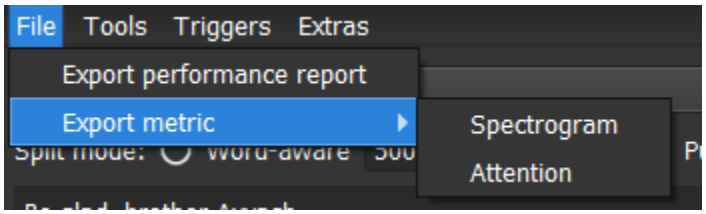
Direct from the Text2Mel model's outputs, the spectrogram is a data-dense representation of sound as perceived by the human ear, and more detailed.

Attention



Only available in Tacotron2, the attention plot shows how the model managed to align the generation. The closer to a nice, diagonal line, the better.

You can export two of these as a standalone image file in the File menu



The resolution of the exported images are determined by the size of the plots themselves as they appear in the interface, with the attention one being 2x its widget size.