

`pedalboard` is a Python library for working with audio: reading, writing, rendering, adding effects, and more. It supports most popular audio file formats and a number of common audio effects out of the box, and also allows the use of VST3® and Audio Unit formats for loading third-party software instruments and effects.

`pedalboard` was built by Spotify's Audio Intelligence Lab to enable using studio-quality audio effects from within Python and TensorFlow. Internally at Spotify, `pedalboard` is used for data augmentation to improve machine learning models and to help power features like Spotify's AI DJ and AI Voice Translation. `pedalboard` also helps in the process of content creation, making it possible to add effects to audio without using a Digital Audio Workstation.



## Features

- Built-in audio I/O utilities (`pedalboard.io`)
  - Support for reading and writing AIFF, FLAC, MP3, OGG, and WAV files on all platforms with no dependencies
  - Additional support for reading AAC, AC3, WMA, and other formats depending on platform
  - Support for on-the-fly resampling of audio files and streams with 0 (1) memory usage
  - Live audio effects via `AudioStream`
- Built-in support for a number of basic audio transformations, including:
  - Guitar-style effects: `Chorus`, `Distortion`, `Phaser`, `Clipping`
  - Loudness and dynamic range effects: `Compressor`, `Gain`, `Limiter`
  - Equalizers and filters: `HighpassFilter`, `LadderFilter`, `LowpassFilter`
  - Spatial effects: `Convolution`, `Delay`, `Reverb`
  - Pitch effects: `PitchShift`
  - Lossy compression: `GSMFullRateCompressor`, `MP3Compressor`

- 
- Quality reduction: [Resample](#), [Bitcrush](#)
  - Supports VST3® instrument and effect plugins on macOS, Windows, and Linux ([pedalboard.load\\_plugin](#))
  - Supports instrument and effect Audio Units on macOS
  - Strong thread-safety, memory usage, and speed guarantees
    - Releases Python’s Global Interpreter Lock (GIL) to allow use of multiple CPU cores
      - \* No need to use [multiprocessing](#)!
    - Even when only using one thread:
      - \* Processes audio up to **300x** faster than pySoX for single transforms, and 2-5x faster than SoxBindings (via iCorv)
      - \* Reads audio files up to **4x** faster than librosa.load (in many cases)
  - Tested compatibility with TensorFlow - can be used in [tf.data](#) pipelines!

## Installation

[pedalboard](#) is available via PyPI (via Platform Wheels):

```
1 pip install pedalboard # That's it! No other dependencies required.
```

If you are new to Python, follow [INSTALLATION.md](#) for a robust guide.

## Compatibility

[pedalboard](#) is thoroughly tested with Python 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, and 3.12 as well as experimental support for PyPy 3.7, 3.8, and 3.9.

- Linux
  - Tested heavily in production use cases at Spotify
  - Tested automatically on GitHub with VSTs
  - Platform [manylinux](#) and [musllinux](#) wheels built for [x86\\_64](#) (Intel/AMD) and [aarch64](#) (ARM/Apple Silicon)
  - Most Linux VSTs require a relatively modern Linux installation (with glibc > 2.27)
- macOS
  - Tested manually with VSTs and Audio Units
  - Tested automatically on GitHub with VSTs
  - Platform wheels available for both Intel and Apple Silicon

- 
- Compatible with a wide range of VSTs and Audio Units
  - Windows
    - Tested automatically on GitHub with VSTs
    - Platform wheels available for [amd64](#) (x86-64, Intel/AMD)

## Examples

**Note:** If you'd rather watch a video instead of reading examples or documentation, **watch *Working with Audio in Python (feat. Pedalboard)* on YouTube.**

## Quick start

```
1 from pedalboard import Pedalboard, Chorus, Reverb
2 from pedalboard.io import AudioFile
3
4 # Make a Pedalboard object, containing multiple audio plugins:
5 board = Pedalboard([Chorus(), Reverb(room_size=0.25)])
6
7 # Open an audio file for reading, just like a regular file:
8 with AudioFile('some-file.wav') as f:
9
10     # Open an audio file to write to:
11     with AudioFile('output.wav', 'w', f.samplerate, f.num_channels) as o:
12
13         # Read one second of audio at a time, until the file is empty:
14         while f.tell() < f.frames:
15             chunk = f.read(f.samplerate)
16
17             # Run the audio through our pedalboard:
18             effected = board(chunk, f.samplerate, reset=False)
19
20             # Write the output to our output file:
21             o.write(effected)
```

**Note:** For more information about how to process audio through Pedalboard plugins, including how the `reset` parameter works, see the documentation for `pedalboard.Plugin.process`.

## Making a guitar-style pedalboard

```
1 # Don't do import *! (It just makes this example smaller)
2 from pedalboard import *
```

---

```

3 from pedalboard.io import AudioFile
4
5 # Read in a whole file, resampling to our desired sample rate:
6 samplerate = 44100.0
7 with AudioFile('guitar-input.wav').resampled_to(samplerate) as f:
8     audio = f.read(f.frames)
9
10 # Make a pretty interesting sounding guitar pedalboard:
11 board = Pedalboard([
12     Compressor(threshold_db=-50, ratio=25),
13     Gain(gain_db=30),
14     Chorus(),
15     LadderFilter(mode=LadderFilter.Mode.HPF12, cutoff_hz=900),
16     Phaser(),
17     Convolution("./guitar_amp.wav", 1.0),
18     Reverb(room_size=0.25),
19 ])
20
21 # Pedalboard objects behave like lists, so you can add plugins:
22 board.append(Compressor(threshold_db=-25, ratio=10))
23 board.append(Gain(gain_db=10))
24 board.append(Limiter())
25
26 # ... or change parameters easily:
27 board[0].threshold_db = -40
28
29 # Run the audio through this pedalboard!
30 effected = board(audio, samplerate)
31
32 # Write the audio back as a wav file:
33 with AudioFile('processed-output.wav', 'w', samplerate, effected.shape
34               [0]) as f:
35     f.write(effected)

```

## Using VST3® or Audio Unit instrument and effect plugins

```

1 from pedalboard import Pedalboard, Reverb, load_plugin
2 from pedalboard.io import AudioFile
3 from mido import Message # not part of Pedalboard, but convenient!
4
5 # Load a VST3 or Audio Unit plugin from a known path on disk:
6 instrument = load_plugin("./VSTs/Magical8BitPlug2.vst3")
7 effect = load_plugin("./VSTs/RoughRider3.vst3")
8
9 print(effect.parameters.keys())
10 # dict_keys([
11 #     'sc_hpf_hz', 'input_lvl_db', 'sensitivity_db',
12 #     'ratio', 'attack_ms', 'release_ms', 'makeup_db',
13 #     'mix', 'output_lvl_db', 'sc_active',

```

---

```

14 # 'full_bandwidth', 'bypass', 'program',
15 # ])
16
17 # Set the "ratio" parameter to 15
18 effect.ratio = 15
19
20 # Render some audio by passing MIDI to an instrument:
21 sample_rate = 44100
22 audio = instrument(
23     [Message("note_on", note=60), Message("note_off", note=60, time=5)],
24     duration=5, # seconds
25     sample_rate=sample_rate,
26 )
27
28 # Apply effects to this audio:
29 effected = effect(audio, sample_rate)
30
31 # ...or put the effect into a chain with other plugins:
32 board = Pedalboard([effect, Reverb()])
33 # ...and run that pedalboard with the same VST instance!
34 effected = board(audio, sample_rate)

```

## Creating parallel effects chains

This example creates a delayed pitch-shift effect by running multiple Pedalboards in parallel on the same audio. `Pedalboard` objects are themselves `Plugin` objects, so you can nest them as much as you like:

```

1 from pedalboard import Pedalboard, Compressor, Delay, Distortion, Gain,
  PitchShift, Reverb, Mix
2
3 passthrough = Gain(gain_db=0)
4
5 delay_and_pitch_shift = Pedalboard([
6     Delay(delay_seconds=0.25, mix=1.0),
7     PitchShift(semitones=7),
8     Gain(gain_db=-3),
9 ])
10
11 delay_longer_and_more_pitch_shift = Pedalboard([
12     Delay(delay_seconds=0.5, mix=1.0),
13     PitchShift(semitones=12),
14     Gain(gain_db=-6),
15 ])
16
17 board = Pedalboard([
18     # Put a compressor at the front of the chain:
19     Compressor(),

```

---

```
20 # Run all of these pedalboards simultaneously with the Mix plugin:
21 Mix([
22     passthrough,
23     delay_and_pitch_shift,
24     delay_longer_and_more_pitch_shift,
25 ]),
26 # Add a reverb on the final mix:
27 Reverb()
28 ])
```

## Running Pedalboard on Live Audio

On macOS or Windows, Pedalboard supports streaming live audio through an `AudioStream` object, allowing for real-time manipulation of audio by adding effects in Python.

```
1 from pedalboard import Pedalboard, Chorus, Compressor, Delay, Gain,
  Reverb, Phaser
2 from pedalboard.io import AudioStream
3
4 # Open up an audio stream:
5 with AudioStream(
6     input_device_name="Apogee Jam+", # Guitar interface
7     output_device_name="MacBook Pro Speakers"
8 ) as stream:
9     # Audio is now streaming through this pedalboard and out of your
      speakers!
10    stream.plugins = Pedalboard([
11        Compressor(threshold_db=-50, ratio=25),
12        Gain(gain_db=30),
13        Chorus(),
14        Phaser(),
15        Convolution("./guitar_amp.wav", 1.0),
16        Reverb(room_size=0.25),
17    ])
18    input("Press enter to stop streaming...")
19
20 # The live AudioStream is now closed, and audio has stopped.
```

## Using Pedalboard in tf.data Pipelines

```
1 import tensorflow as tf
2
3 sr = 48000
4
5 # Put whatever plugins you like in here:
6 plugins = pedalboard.Pedalboard([pedalboard.Gain(), pedalboard.Reverb()
  ])
```

---

```

7
8 # Make a dataset containing random noise:
9 # NOTE: for real training, here's where you'd want to load your audio
   somehow:
10 ds = tf.data.Dataset.from_tensor_slices([np.random.rand(sr)])
11
12 # Apply our Pedalboard instance to the tf.data Pipeline:
13 ds = ds.map(lambda audio: tf.numpy_function(plugins.process, [audio, sr
   ], tf.float32))
14
15 # Create and train a (dummy) ML model on this audio:
16 model = tf.keras.models.Sequential([tf.keras.layers.InputLayer(
   input_shape=(sr,)), tf.keras.layers.Dense(1)])
17 model.compile(loss="mse")
18 model.fit(ds.map(lambda effected: (effected, 1)).batch(1), epochs=10)


```

For more examples, see: - the “examples” folder of this repository - the “Pedalboard Demo” Colab notebook - *Working with Audio in Python (feat. Pedalboard)* by Peter Sobot at EuroPython 2022 - an interactive web demo on Hugging Face Spaces and Gradio (via @AK391)

## Contributing

Contributions to `pedalboard` are welcomed! See CONTRIBUTING.md for details.

## Citing

To cite `pedalboard` in academic work, use its entry on Zenodo:  DOI 10.5281/zenodo.7817838

To cite via BibTeX:

```

1 @software{sobot_peter_2023_7817838,
2   author      = {Sobot, Peter},
3   title       = {Pedalboard},
4   month       = jul,
5   year        = 2021,
6   publisher   = {Zenodo},
7   doi         = {10.5281/zenodo.7817838},
8   url        = {https://doi.org/10.5281/zenodo.7817838}
9 }

```

## License

`pedalboard` is Copyright 2021-2023 Spotify AB.

---

`pedalboard` is licensed under the GNU General Public License v3. `pedalboard` includes a number of libraries that are statically compiled, and which carry the following licenses:

- The core audio processing code is pulled from JUCE 6, which is dual-licensed under a commercial license and the GPLv3.
- The VST3 SDK, bundled with JUCE, is owned by Steinberg® Media Technologies GmbH and licensed under the GPLv3.
- The `PitchShift` plugin uses the Rubber Band Library, which is dual-licensed under a commercial license and the GPLv2 (or newer).
- The `MP3Compressor` plugin uses libmp3lame from the LAME project, which is licensed under the LGPLv2 and upgraded to the GPLv3 for inclusion in this project (as permitted by the LGPLv2).
- The `GSMFullRateCompressor` plugin uses libgsm, which is licensed under the ISC license and compatible with the GPLv3.

*VST is a registered trademark of Steinberg Media Technologies GmbH.*