Audio Engineering Society
Convention Paper 10483
Presented at the 150th Convention
2021 May 25-28, Online

pyloudnorm: A simple yet flexible loudness meter in Python

Christian J. Steinmetz and Joshua D. Reiss

Centre for Digital Music, Queen Mary University of London, London, UK

Correspondence should be addressed to Christian J. Steinmetz (`c.j.steinmetz@qmul.ac.uk`)

ABSTRACT

The ITU-R BS.1770 recommendation for measuring the perceived loudness of audio signals has seen widespread adoption in broadcasting. Due to its simplicity, this algorithm has now found applications across audio signal processing. Here we describe **pyloudnorm**, a Python package that enables the measurement of integrated loudness following the recommendation. While a number of implementations are available, ours provides an easy-to-install package, a simple interface, and the ability to adjust the algorithm parameters, a feature that others neglect. We outline the design of pyloudnorm and discuss a set of modifications based upon recent literature that improve the robustness of loudness measurements. We perform an evaluation comparing accuracy and runtime with six other implementations, demonstrating that pyloudnorm is both fully compliant and one of the fastest options.

1 Introduction

The nonlinear nature of the human auditory system makes measurement of the perceived loudness of sound challenging [1]. While subjective loudness has been an active area of research in psychoacoustics over the last half century [2–5], these models are often complex and not applicable to measuring the loudness of streaming or recorded audio. For this reason, there has been a longstanding interest within the broadcast industry in simple loudness models, as they enable the ability to monitor and control the listener experience [6–11]. Concurrently, there has been interest in methods for measuring the loudness of music such as Vickers' loudness [12] and ReplayGain [13]. In an effort to standardize, simplify, and improve upon previous approaches, the ITU-R BS.1770 recommendation was proposed [14], and has now seen widespread adoption [15, 16]. The proposed metering algorithm was later included in the EBU R 128 recommendation, which dictates loudness for broadcast material [17].

The ITU-R BS.1770 recommendation proposes a straightforward algorithm consisting of frequency-weighting filters and gated energy measurements. The algorithm has been shown to correlate well with the perceived loudness of broadband content, is computationally efficient, and relatively easy to implement. For all these reasons it has now seen widespread adoption in the broadcast industry, and with the rise of online streaming platforms, interest in content normalization with the recommendation has sustained [18, 19].

While this recommendation has been found to correlate well with broadband content, further listening studies have discovered that this is not always the case, especially for narrowband content [20–23]. These investigations have led to a series of proposed modifications to the original recommendation, which generally consists of adjustments to the algorithm parameters for the frequency-weighting filters and gating block sizes. While these modifications are relatively straightforward to incorporate, there has been limited adoption thus far.

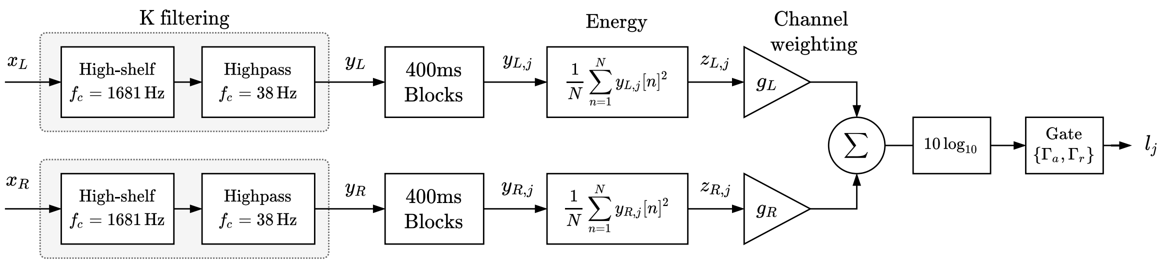
K filtering
Energy
Channel weighting
x_L
High-shelf $f_c = 1681\,\text{Hz}$
Highpass $f_c = 38\,\text{Hz}$
y_L
400ms Blocks
$y_{L,j}$
$\frac{1}{N}\sum_{n=1}^{N} y_{L,j}[n]^2$
$z_{L,j}$
g_L
x_R
y_R
$y_{R,j}$
$\frac{1}{N}\sum_{n=1}^{N} y_{R,j}[n]^2$
$z_{R,j}$
g_R
Σ
$10\log_{10}$
Gate $\{\Gamma_a, \Gamma_r\}$
l_j

Fig. 1: Measurement of integrated loudness following the ITU-R BS.1770 recommendation [14].

While originally intended for broadcast scenarios, due to its simplicity and efficacy, the ITU-R BS.1770 recommendation has now found applications across audio signal processing [23–30]. In order to meet the demand for applications outside its original scope, a number of implementations now exist that provide a programmatic interface. Unfortunately, most of these implementations are either difficult to install, provide an interface that is not efficiently accessed from Python, or do not allow for modifications of the algorithm parameters. For these reasons, we built **pyloudnorm**[1], a Python package that is easily installed and integrated into existing projects, while also providing the ability to adjust the underlying algorithm parameters.

The simplicity and flexibility of this package has lead to an interest in pyloudnorm, which a variety of works now utilize. Current applications include pre-processing for audio machine learning datasets, such as the CLEAR dataset for acoustic question answering [31], a dataset of multichannel hearing aid recordings [32], the LibriMix dataset which generated mixtures of speech for source separation [33], the creation of music mixtures in the Slakh dataset [34], and the OrchideaSOL dataset [35], which features orchestral recordings. In addition to dataset pre-processing, there have also been applications in feature extraction for machine learning with the Surfboard library [36], as well as in data augmentation in Scaper [37], and as a final post-processing step in voice conversion [38].

The structure of the paper is as follows. In Section 2 we describe the algorithm and introduce recently proposed modifications. Then in Section 3 we introduce pyloudnorm, along with other existing implementations. Section 4 presents an evaluation of these implementations using the compliance material provided by the recommendation, as well as our own collection of examples. We finally present conclusions in Section 5.

[1]`https://github.com/csteinmetz1/pyloudnorm`

2 Algorithm

The proposed algorithm is outlined for the stereo case at a high level in Fig. 1. First, the "K"-frequency weighting consists of a high-shelf filter that aims to mimic the response of the head, followed by a highpass filter that reduces the influence of low frequencies. Then we take the filtered signal of each channel y_i, and split this into overlapping blocks of 400 ms, with an overlap of 75%. We then compute the energy of each block j in each channel i

$$z_{i,j} = \frac{1}{N}\sum_{n=1}^{N} y_{i,j}[n]^2,$$

where N is the number of samples in each block, and n is the sample index within the block. The loudness of each block is then given by

$$l_j = -0.691 + 10\log_{10}\sum_i g_i \cdot z_{i,j},$$

where $g_i = [1, 1, 1, 1.41, 1.41]$, for the left, right, centre, left surround, and right surround, respectively.

The final step involves applying a gate in order to reduce the influence of blocks with low energy. An absolute threshold is given by $\Gamma_a = -70\,\text{dB LUFS}$, along with a second relative threshold Γ_r, which is determined by first measuring the loudness of all the blocks above the absolute threshold and subtracting 10

$$\Gamma_r = -0.691 + 10\log_{10}\sum_i g_i\left(\frac{1}{|J_g|}\sum_{J_g} z_{i,j}\right) - 10,$$

where $J_g = \{j : l_j > \Gamma_a\}$, and $|J_g|$ is the number of blocks above the threshold. This enables us to compute the final integrated loudness in the same way by summing only blocks that fall above both thresholds

$$L_{KG} = -0.691 + 10\log_{10}\sum_i g_i\left(\frac{1}{|J_g|}\sum_{J_g} z_{i,j}\right),$$

this time where $J_g = \{j : l_j > \Gamma_a \text{ and } l_j > \Gamma_r\}$.

2.1 Proposed modifications

The recommendation makes clear that loudness measurements correlate well with perception only when the signal being measured is broadband in nature.

> *It should be noted that while this algorithm has been shown to be effective for use on audio programmes that are typical of broadcast content, the algorithm is not, in general, suitable for use to estimate the subjective loudness of pure tones.* [14]

While this is not often an issue when employed in broadcast scenarios, the use of this recommendation in other applications has continued to increase, potentially leading to inaccurate measurements. For example, when measuring the loudness of narrowband or percussive content, such as isolated instruments, sound effects, impulse responses, as well as other recordings.

The degree to which measurements produced by the recommendation deviate from perception has been studied. A number of modifications have been proposed that aim to improve the performance and robustness of measurements. Cabrera et al. [20] proposed some of the first adjustments, informed by a number of listening studies. This involved raising the cutoff frequency of the highpass filter to 149 Hz, and the replacement of the high-shelf filter by a notch filter centered at 1 kHz. Pestana and Barbosa [39] first identified potential shortcomings of the recommendation for common multitrack sources, and later suggested adopting a smaller gating block size of 280 ms in combination with a +10 dB gain on the high-shelf filter [21].

More recently, Fenton and Lee [22] provided two alternative frequency-weighting filters. The first adjustment proposed boosting the gain of the high-shelf filter by +5 dB and changing the cutoff of the highpass filter to 130 Hz, as well as the addition of a peaking filter with a center frequency of 500 Hz. They also investigated a more complex modification that replaced this peaking filter with a higher order variant. Fenton [23] later extended these results with a further listening study, providing optimized filter and gating block size parameters for different instrument types, which they found improved the quality of equal loudness-based mixes. De Man [40] noted that the original recommendation only provided filter coefficients at 48 kHz, so they reverse-engineered the filter specification from the original recommendation and provided filter prototypes that enable fully compliant filters at any sample rate.

3 Implementation

We designed pyloudnorm with simplicity in mind. This means that users can measure the loudness of an array of audio samples using just three lines of code, as shown in Listing 1. This includes importing the package, instantiating a meter with the appropriate sample rate, and then passing an array of audio samples to measure. Calling the integrated loudness method will return the integrated loudness in dB LUFS of an array of audio samples, as measured by a meter following the original recommendation.

```python
import pyloudnorm as pyln
# create BS.1770 meter
meter = pyln.Meter(rate)
# measure loudness of signal x
loudness = meter.integrated_loudness(x)
```

Listing 1: Using pyloudnorm in Python.

This enables users to carry out loudness measurements following the recommendation without any underlying knowledge of the algorithm and its parameters. But, as outlined in the previous section, a number of modifications have been proposed, which improve performance for some use cases. To facilitate the integration of these modifications, pyloudnorm additionally exposes the underlying algorithm parameters to users if desired. To enable use of the proposed modifications we include pre-defined filter specifications. This facilitates the use of these modifications by simply passing a corresponding string while instantiating the meter. We hope that this will enable users to make more accurate measurements in the growing and diverse applications of the loudness recommendation.

In addition to the modifications that have been proposed thus far, pyloudnorm also includes the ability to easily adopt new modifications. This includes any adjustments to the gating block size and frequency-weighting filters. This even enables users to define any cascade of arbitrary second-order filters, providing a high level of flexibility, if desired. To facilitate this, we include an IIR filter class that implements not only the highpass and high-shelf filter prototypes used in the original recommendation, but also low-shelf, lowpass, peaking, and notch filter prototypes. This functionality enables easy adoption of future modifications, as well as provides a platform for experimenting with new modifications to the recommendation.

4 Evaluation

Beyond pyloudnorm, a wide variety of implementations of the loudness recommendation have been made available, both in streaming and offline formats. In this section we compare pyloudnorm to four open source offline implementations, as well as two commercial implementations that provide a graphical user interface.

4.1 Other implementations

Essentia [41] is a popular C++ library with a Python interface for music information retrieval feature extraction. In addition to many other common audio features, Essentia includes an implementation of the loudness recommendation. While Essentia provides an extensive collection of audio features, it is large and somewhat cumbersome to install, making it less attractive in the case where only loudness measurements are desired.

Another popular alternative is ffmpeg[2], a cross-platform tool for processing audio and video. Results from the meter can be accessed in Python by making calls to the command-line, or optionally through a wrapper, at the cost of some additional overhead compared to a pure Python implementation.

libebur128[3] provides an implementation in C, and addresses some of the limitations of ffmpeg, namely that it provides a more efficient interface since it has less overhead than the far more powerful ffmpeg tool. Nevertheless, it still suffers from a similar problem in that it does not feature a direct implementation in Python.

A more recent implementation, loudness.py [40], addresses many of these limitations by providing a Python implementation utilizing NumPy [42], making it the closest to pyloudnorm. Additionally, loudness.py provides a more precise implementation of the frequency-weighting filters based upon reverse-engineering the filter specifications in the recommendation. The major limitation is that it provides a standalone Python function, and therefore cannot easily be installed and imported within a Python project, as pyloudnorm can.

We also consider two commercial implementations. These include the meter in Adobe Audition[4], a popular digital audio workstation, and youlean[5], a web-based implementation. This implementation allows users to upload audio files to a web page for them to be analyzed with measurements reported back to the user.

[2]https://ffmpeg.org/
[3]https://github.com/jiixyj/libebur128
[4]https://www.adobe.com/products/audition
[5]https://youlean.co/file-loudness-meter/

4.2 Compliance material

While no reference meter exists, the ITU-R BS.1770 recommendation provides compliance material in order to evaluate meter implementations [43]. The provided compliance material consists of mono, stereo, and multichannel audio files at 48 kHz sample rate. The recommendation states that a compliant meter should measure within $\pm$ 0.1 dB LUFS of the target value.

As a first step, we compare the measurements of the compliance material from the different implementations. Since pyloudnorm incorporates the filter specification from loudness.py proposed by De Man [40], we perform measurements with pyloudnorm both in its default configuration, as well as with the modified filters. We report the full results of these measurements in Table 1. While we find that pyloudnorm, libebur128, and Audition provide measurements that are fully compliant, surprisingly, we find that some implementations do not produce the correct measurements. Code to reproduce these results along with the compliance material is made available online[6].

loudness.py

The loudness.py implementation achieves compliance on all test material except for `AbsGateTest`, which has a significant error of -1.96 dB LUFS. This is likely due to an issue when the input signal contains multiple blocks of total silence. Interestingly, the result of a computation with a block of silence results in a loudness for the block of $-\infty$ dB LUFS, which conceptually agrees with our understanding, yet this causes a problem when summing the loudness of the blocks. In pyloudnorm, we handle this by converting $-\infty$ values to the smallest value supported by the platform (generally $-1.8\mathrm{e}308$ for float64) before the summation of the blocks.

ffmpeg

Similar to loudness.py, ffmpeg produces mostly compliant measurements with the exception of a single test case. We measured a deviation of +0.4 dB LUFS from the target value in `RelGateTest` . It is unclear what causes this deviation in the measurement. Interestingly, on many of the other measurements, particularly the sine tone examples, the ffmpeg meter produces readings that are exactly +0.1 dB LUFS above the target, which is at the exact limit for compliance.

[6]https://github.com/csteinmetz1/pyloudnorm-eval

		Implementation							
File	Target	pyloudnorm		loudness.py	ffmpeg	libebur128	Essentia	Audition	youlean
		Default	De Man						
`FrequencySweep`	-18.0	-18.03	-17.99	-17.99	-18.00	-18.00	-18.18	-18.03	-18.02
`25Hz_2ch`	-23.0	-23.00	-22.99	-22.99	-23.10	-23.00	**-26.37**	-23.04	-23.02
`100Hz_2ch`	-23.0	-23.03	-22.99	-22.99	-23.10	-23.00	**-22.86**	-23.04	-23.02
`500Hz_2ch`	-23.0	-23.04	-22.99	-22.99	-23.10	-23.00	-22.99	-23.04	-23.02
`1000Hz_2ch`	-23.0	-23.03	-22.99	-22.99	-23.10	-23.00	-23.00	-23.04	-23.02
`2000Hz_2ch`	-23.0	-23.03	-22.99	-22.99	-23.10	-23.00	-23.00	-23.04	-23.02
`10000Hz_2ch`	-23.0	-23.04	-22.99	-22.99	-23.10	-23.00	-23.00	-23.04	-23.02
`25Hz_2ch`	-24.0	-24.00	-23.99	-23.99	-24.10	-24.00	**-27.21**	-24.04	-24.02
`100Hz_2ch`	-24.0	-24.03	-23.99	-23.99	-24.10	-24.00	-23.92	-24.04	-24.02
`500Hz_2ch`	-24.0	-24.04	-23.99	-23.99	-24.10	-24.00	-23.99	-24.04	-24.02
`1000Hz_2ch`	-24.0	-24.04	-23.99	-23.99	-24.10	-24.00	-24.00	-24.04	-24.02
`2000Hz_2ch`	-24.0	-24.04	-23.99	-23.99	-24.10	-24.00	-24.00	-24.04	-24.02
`10000Hz_2ch`	-24.0	-24.04	-23.99	-23.99	-24.10	-24.00	-24.00	-24.04	-24.02
`RelGateTest`	-10.0	-10.07	-10.03	-10.03	**-9.60**	-10.00	-10.03	-10.07	**-10.15**
`AbsGateTest`	-69.5	-69.49	-69.45	**-71.46**	-69.50	-69.50	-69.45	-69.49	-69.55
`Mono_Voice+Music`	-23.0	-23.03	-22.99	-22.99	-23.10	-23.00	-22.97	-23.03	-22.98
`Mono_Voice+Music`	-24.0	-24.03	-23.99	-23.99	-24.10	-24.00	-23.97	-24.04	-23.98
`Stereo_VinL+R`	-23.0	-23.03	-22.98	-22.98	-23.10	-23.00	-22.97	-23.02	-22.99
`Stereo_VinL+R`	-24.0	-24.02	-23.98	-23.98	-24.10	-24.00	-23.97	-24.02	-23.98

Table 1: Comparison of loudness algorithm implementations with provided compliance material [43]. Measurements that are not within the ± 0.1 dB LUFS tolerance for compliance are marked in **boldface**.

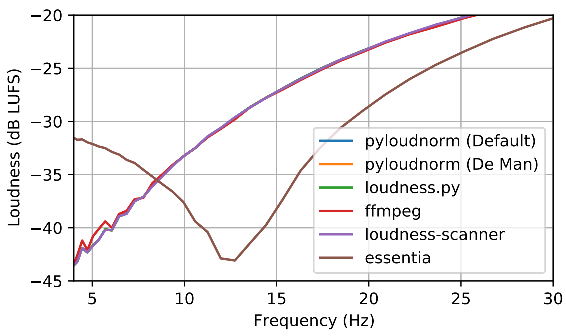

Fig. 2: Measured loudness of -6 dB sinusoidal tones.

Essentia

We find that Essentia underestimates the loudness of very low frequency content, as is evident in the `25Hz_2ch` test case, where we measure a deviation of -3.37 dB LUFS. This leads us to believe that Essentia has a subtle difference in the shape of the filter in the low-end of the frequency range. We measured the loudness of a number of low frequency sine tones with a peak of -6 dB, as shown in Fig. 2. This clearly demonstrates a deviation in the shape of the filtering in the low frequencies, yet the cause of this is not clear.

4.3 Challenging material

As a further investigation, we collected a set of material aimed at challenging the implementations. This includes a number of pure tones, noise, as well as instrument recordings. Since there exists no target loudness, we look for agreement among the implementations. We compute the mean measurement across all implementations and identify any that appear to disagree with the mean by a significant margin (≥ 0.5 dB LUFS). We report measurements on these signals in Table 2.

Notably, we find that ffmpeg produces results that disagree with the other implementations for a number of the test signals, in some cases producing a deviation of over 1 dB LUFS. Essentia agrees for most of the test material, but measures a deviation of nearly -5 dB LUFS for the 16 Hz sine tone, as we would expect based on our earlier findings. loudness.py agrees with the other implementations, and the issue we observed previously does not appear. Finally, it appears that youlean deviates significantly, with a moderate difference in six of the thirteen test signals. As before, pyloudnorm, libebur128, and Audition tend to produce measurements that closely agree with each another.

		Implementation							
File	Mean	pyloudnorm		loudness.py	ffmpeg	libebur128	Essentia	Audition	youlean
		Default	De Man						
sine_16Hz	-24.08	-23.44	-23.44	-23.36	-23.50	-23.40	**-28.48**	-23.48	-23.51
sine_1000Hz	-3.13	-3.05	-3.01	-3.01	-3.00	-3.00	-3.01	-3.05	**-3.88**
sine_1000Hz_pad	-4.18	-4.19	-4.15	-4.15	-4.20	-4.10	-4.15	-4.19	-4.32
sine_16000Hz	-19.77	-19.69	-19.64	-19.64	-19.70	-19.60	-19.64	-19.69	-20.52
sine_19000Hz	-19.78	-19.69	-19.64	-19.64	-19.80	-19.60	-19.64	-19.69	**-20.52**
multi-sines	-10.65	-10.67	-10.62	-10.62	-10.60	-10.60	-10.64	-10.67	-10.79
hf-noise	-9.34	-9.21	-9.16	-9.15	-9.60	-9.20	-9.16	-9.21	**-10.04**
chirp-150-190	-6.69	-6.55	-6.50	-6.52	-6.50	-6.50	-6.51	-6.55	**-7.88**
our_gating_test	-3.37	-3.37	-3.33	-3.33	-3.30	-3.30	-3.33	-3.37	-3.61
piano-D6	-25.12	-25.02	-24.98	-24.98	**-28.20**	-25.00	-24.98	-25.03	**-22.73**
soprano-E4	-29.74	-29.82	-29.77	-29.57	-29.60	-29.60	-29.78	-29.61	**-30.15**
vibraphone-C6	-17.29	-16.95	-16.90	-16.90	**-17.90**	-16.90	-19.60	-16.95	**-16.23**
violin-B3	-12.78	-12.82	-12.78	-12.69	-12.70	-12.70	-12.78	-12.74	-13.00

Table 2: Comparison of loudness algorithm implementations with alternative material.
Measurements that disagree with others significantly ($\geq$ 0.5 dB LUFS) are marked in **boldface**.

Overall, it appears that many of the trends from the original compliance material hold, namely that pyloudnorm, along with loudness.py, libebur128, and Adobe Audition produce measurements that tend to agree with each other. Nevertheless, these findings underlie an important realization that not all of these implementations produce equivalent results. Therefore care should be taken when utilizing their measurements, especially in cases where measurements will be compared across the different implementations, as this could lead to audible differences in normalization.

4.4 Runtime

As the final component of our evaluation, we investigate the runtime performance of the Python implementations. While this evaluation does not directly measure the runtime of each implementation, it provides insight into the amount of time required when performing these measurements from Python. We measured the average execution time for each audio file in the set of compliance material over 10 runs. We then report the average real-time factor (RTF) for each implementation in Table 3, with the real-time factor indicating how much faster the execution takes with respect to the duration of the audio file.

We find that ffmpeg is clearly the slowest, and this is likely due to the overhead required in the processing pipeline, which must be invoked in order to measure the loudness. This is followed by Essentia, which provides its own interface for reading audio data from disk. It performs about 3x faster than ffmpeg, but still lags behind the others. libebur128 achieves a 4x speedup compared to ffmpeg, likely due to having less overhead.

Implementation	RTF	Audio Loader
ffmpeg	26x	ffmpeg
Essentia	88x	Essentia
libebur128	114x	ffmpeg
loudness.py	421x	pysoundfile
pyln (Default)	338x	pysoundfile
pyln (De Man)	455x	pysoundfile

Table 3: Mean real-time factor.

In the case of the native Python implementations, which include pyloudnorm and loudness.py, we utilize the soundfile package[7] to read the audio data from disk. We count the time to load the audio data in the timings to provide a more fair comparison with the previous methods that require system calls and utilize their own data loading process. We see a significant speedup with loudness.py, which achieves a RTF that is 16x greater than ffmepg. These timings are comparable to pyloudnorm, which has similar overhead. These results appear to agree with our intuition that simpler implementations with less overhead tend to perform more efficiently.

[7]https://github.com/bastibe/python-soundfile

5 Conclusion

In this work, we presented pyloudnorm, an easy-to-install Python package that implements the ITU-R BS.1770 recommendation for measuring the perceived loudness of audio signals. We outlined the flexible design of pyloudnorm, and discussed the optional modifications that we incorporate based on recent literature. These modifications aim to improve the performance of measurements for more diverse content, and have been neglected in previous implementations. We compared pyloudnorm to a number of open source and commercial implementations, and performed an evaluation of their measurements on the original compliance material, as well as our own test signals. We found that while most meters are compliant, a few appear to produce unexpected measurements, and may not be fully compliant. We demonstrated that pyloudnorm is both fully compliant, and tends to agree with other meters on a set of test material meant to stress these implementations. Finally, we compared the runtime of these implementations and found that pyloudnorm is among one of the fastest options.

Acknowledgements

This work is supported by the EPSRC UKRI Centre for Doctoral Training in Artificial Intelligence and Music (EP/S022694/1).

References

[1] Stevens, S. S., "The measurement of loudness," *JASA*, 27(5), pp. 815–829, 1955.

[2] Stevens, S. S., "The direct estimation of sensory magnitudes: Loudness," *The American journal of psychology*, 69(1), pp. 1–25, 1956.

[3] Zwicker, E. and Scharf, B., "A model of loudness summation." *Psychological review*, 72(1), p. 3, 1965.

[4] Moore, B. C. and Glasberg, B. R., "A revision of Zwicker's loudness model," *Acta Acustica united with Acustica*, 82(2), pp. 335–345, 1996.

[5] Moore, B. C. J., "Development and Current Status of the "Cambridge" Loudness Models," *Trends in Hearing*, 18, 2014.

[6] Bauer, B., Torick, E., Rosenheck, A., and Allen, R., "A loudness-level monitor for broadcasting," *IEEE Transactions on Audio and Electroacoustics*, 15(4), pp. 177–182, 1967.

[7] Jones, B. L. and Torick, E. L., "A new loudness indicator for use in broadcasting," *SMPTE Journal*, 90(9), pp. 772–777, 1981.

[8] Bauer, B. and Torick, E., "Researches in loudness measurement," *IEEE Transactions on Audio and Electroacoustics*, 14(3), pp. 141–151, 1966.

[9] Skovenborg, E. and Nielsen, S. H., "Evaluation of different loudness models with music and speech material," in *116th AES Convention*, 2004.

[10] Soulodre, G. A., "Evaluation of objective loudness meters," in *116th AES Convention*, 2004.

[11] Lund, T., "Control of Loudness in Digital TV," in *NAB Convention*, 2006.

[12] Vickers, E., "Automatic long-term loudness and dynamics matching," in *111th AES Convention*, 2001.

[13] Robinson, D. J. M., *Perceptual model for assessment of coded audio*, Ph.D. thesis, University of Essex, 2002.

[14] ITU-R BS.1770-4, "Algorithms to Measure Audio Programme Loudness and True-peak Audio Level," Recommendation, International Telecommunications Union, 2015.

[15] Lund, T., "ITU-R BS. 1770 Revisited," in *NAB Convention*, 2011.

[16] Lund, T., "The CALM Act and cross-platform broadcast," in *NAB Convention*, 2012.

[17] EBU R 128, "Loudness normalisation and permitted maximum level of audio signals," Recommendation, European Broadcasting Union, 2012.

[18] Katz, B., "Sound Board: Can We Stop the Loudness War in Streaming?" *JAES*, 63(11), pp. 939–940, 2015.

[19] Grimm, E., "Analyzing Loudness Aspects of 4.2 Million Musical Albums in Search of an Optimal Loudness Target for Music Streaming," in *147th AES Convention*, 2019.

[20] Cabrera, D., Dash, I., and Miranda, L., "Multi-channel loudness listening test," in *124th AES Convention*, 2008.

[21] Pestana, P. D., Reiss, J. D., and Barbosa, A., "Loudness measurement of multitrack audio content using modifications of ITU-R BS. 1770," in *134th AES Convention*, 2013.

[22] Fenton, S. and Lee, H., "Alternative Weighting Filters for Multi-Track Program Loudness Measurement," in *143rd AES Convention*, 2017.

[23] Fenton, S., "Automatic mixing of multitrack material using modified loudness models," in *145th AES Convention*, 2018.

[24] Olive, S. E., Welti, T., and McMullin, E., "A virtual headphone listening test methodology," in *AES Conference: 51st International Conference: Loudspeakers and Headphones*, 2013.

[25] Jillings, N., Moffat, D., De Man, B., and Reiss, J. D., "Web Audio Evaluation Tool: A browser-based listening test environment," in *12th Sound and Music Computing Conference*, 2015.

[26] Friberg, A. et al., "Using listener-based perceptual features as intermediate representations in music information retrieval," *JASA*, 136(4), pp. 1951–1963, 2014.

[27] Schoeffler, M., Stöter, F.-R., Bayerlein, H., Edler, B., and Herre, J., "An Experiment about Estimating the Number of Instruments in Polyphonic Music: A Comparison Between Internet and Laboratory Results." in *ISMIR*, pp. 389–394, 2013.

[28] Ward, D., Reiss, J. D., and Athwal, C., "Multi-track mixing using a model of loudness and partial loudness," in *133rd AES Convention*, 2012.

[29] Mansbridge, S., Finn, S., and Reiss, J. D., "Implementation and evaluation of autonomous multi-track fader control," in *132nd AES Convention*, 2012.

[30] Ward, D. and Reiss, J. D., "Loudness algorithms for automatic mixing," in *AES Workshop on Intelligent Music Production*, 2016.

[31] Abdelnour, J., Salvi, G., and Rouat, J., "CLEAR: A Dataset for Compositional Language and Elementary Acoustic Reasoning," in *ViGIL Workshop at NeurIPS 2020*, 2018.

[32] Fischer, T., Caversaccio, M., and Wimmer, W., "Multichannel acoustic source and image dataset for the cocktail party effect in hearing aid and implant users," *Scientific data*, 7(1), pp. 1–13, 2020.

[33] Cosentino, J., Pariente, M., Cornell, S., Deleforge, A., and Vincent, E., "LibriMix: An Open-Source Dataset for Generalizable Speech Separation," *arXiv:2005.11262*, 2020.

[34] Manilow, E. et al., "Cutting Music Source Separation Some Slakh: A Dataset to Study the Impact of Training Data Quality and Quantity," in *WASPAA*, IEEE, 2019.

[35] Cella, C. E. et al., "OrchideaSOL: a dataset of extended instrumental techniques for computer-aided orchestration," *arXiv:2007.00763*, 2020.

[36] Lenain, R., Weston, J., Shivkumar, A., and Fristed, E., "Surfboard: Audio Feature Extraction for Modern Machine Learning," *arXiv:2005.08848*, 2020.

[37] Salamon, J., MacConnell, D., Cartwright, M., Li, P., and Bello, J. P., "Scaper: A library for soundscape synthesis and augmentation," in *WASPAA*, pp. 344–348, 2017.

[38] Chen, M., Shi, Y., and Hain, T., "Towards Low-Resource StarGAN Voice Conversion using Weight Adaptive Instance Normalization," *arXiv:2010.11646*, 2020.

[39] Pestana, P. D. and Barbosa, Á., "Accuracy of ITU-R BS. 1770 Algorithm in Evaluating Multitrack Material," in *133rd AES Convention*, 2012.

[40] De Man, B., "Evaluation of Implementations of the EBU R128 Loudness Measurement," in *145th AES Convention*, 2018.

[41] Bogdanov, D. et al., "Essentia: An audio analysis library for music information retrieval," in *ISMIR*, 2013.

[42] Harris, C. R. et al., "Array programming with NumPy," *Nature*, 585(7825), pp. 357–362, 2020.

[43] ITU-R BS.2217, "Compliance material for Recommendation ITU-R BS.1770," Recommendation, International Telecommunications Union, 2011.