ESSENTIA.JS: A JAVASCRIPT LIBRARY FOR MUSIC AND AUDIO ANALYSIS ON THE WEB

Albin Correya[1] **Dmitry Bogdanov**[1] **Luis Joglar-Ongay**[1,2] **Xavier Serra**[1]
[1] Music Technology Group, Universitat Pompeu Fabra, Barcelona, Spain
[2] SonoSuite, Barcelona, Spain
albin.correya@upf.edu, dmitry.bogdanov@upf.edu

ABSTRACT

Open-source software libraries for audio/music analysis and feature extraction have a significant impact on the development of Audio Signal Processing and Music Information Retrieval (MIR) systems. Despite the abundance of such tools on the native computing platforms, there is a lack of an extensive and easy-to-use reference library for audio feature extraction on the Web. In this paper, we present *Essentia.js*, an open-source JavaScript (JS) library for audio and music analysis on both web clients and JS-based servers. Along with the Web Audio API, it can be used for efficient and robust real-time audio feature extraction on the web browsers. *Essentia.js* is modular, lightweight, and easy-to-use, deploy, maintain and integrate into the existing plethora of JS libraries and Web technologies. It is powered by a WebAssembly back-end of the Essentia C++ library, which facilitates a JS interface to a wide range of low-level and high-level audio features. It also provides a higher-level JS API and add-on MIR utility modules along with extensive documentation, usage examples, and tutorials. We benchmark the proposed library on two popular web browsers, Node.js engine, and Android devices, comparing it to the native performance of Essentia and Meyda JS library.

1. INTRODUCTION

The Web has become one of the most used computing platforms with billions of web pages served daily, and JS being an essential part of building modern web applications. Using HTML, CSS, and JS, developers can make dynamic, interactive, and responsive web pages by implementing custom client-side scripts. At the same time, the developers can also use cross-platform run-time engines like Node.js [1] to write server-side code in JS. The flexibility of using JS on both server and client ends of web applications arguably makes it one of the most used computer programming languages in the recent years [2]. JS is also widely used as an entry-level programming language by the thinkers from design, art, computer graphics, architecture, and spaces in between where audio processing and analysis can be relevant.

With the adoption of both HTML5 and the W3C Web Audio API specifications [14], modern web browsers are capable of audio processing, synthesis, and analysis without any third-party dependencies on proprietary software. This allows developers to move most of the audio processing code from the server to the client and can provide better scalability and deployment, considering that the web-client has computational resources for the required processing. Web Audio API provides a JS interface to various predefined nodes for audio processing, synthesis, and analysis. Even though the provided capabilities are limited, the API includes the *ScriptProcessorNode* for developers to add custom JS code for audio processing. [2] The design of this node has been criticized by the audio developer community since it runs the JS audio processing code on the main UI thread, which can result in unreliable performance and audio glitching [15]. As an alternative, *AudioWorklet* [10] has been introduced to mitigate this design issue to a great extent by allowing running custom audio processing code on a dedicated audio thread. It also allows bi-directional communication between the audio thread and the control thread of Web Audio API asynchronously using the web message ports.

Despite all of these recent developments of Web Audio technologies, the Audio Signal Processing and MIR communities still lack reliable and modular software tools and libraries that could be easily used for building audio and music analysis applications for web browsers and JS runtime engines. To the best of our knowledge, Meyda [11] and JS-Xtract [18] are the few available JS libraries that are ready-to-use and have implementations of a limited set of MIR audio features. [3] The lack of more extensive alternatives is not surprising, given that writing a new JS audio analysis library from scratch would require a lot of effort. Also, JS code for audio processing are prone to performance issues due to the *just-in-time* (JIT) compilation and garbage collection overhead, which can be critical for

[1] https://nodejs.org

[2] https://www.w3.org/TR/webaudio/#scriptprocessornode

[3] As of May 2020, Meyda only has 20 MIR algorithms.

real-time audio and music analysis tasks. Due to these reasons, researchers and developers often rely on server-side audio processing solutions using the existing native MIR tools for writing the required web applications.

Over the last two decades, the existing software tools for audio analysis have been mostly written in C/C++, Java and Python and delivered as standalone applications, host application plug-ins, or as software library packages. Software libraries with a Python API, such as Essentia [7], Librosa [23], Madmom [6], Yaafe [22] and Aubio [8], have been especially popular within the MIR community due to rapid prototyping and rich environment for scientific computations. Meanwhile, the libraries with a native C/C++ back-end offered faster analysis [24] and were often required for writing industrial audio applications. Various web applications for audio processing and analysis have been developed using these tools. Spotify API [4] (formerly Echonest), Freesound API [13] and AcousticBrainz [25] are examples of web services providing precomputed audio features to the end users via a REST API. Besides, numerous web applications were built for aiding tasks such as crowd sourcing audio annotations [9, 12], audio listening tests [19, 26] and music education platforms [1, 21] to mention a few. All these services manage their audio analysis on the server, which may require a significant effort and resources to scale to many users.

With the recent arrival of WebAssembly (WASM) support on most of the modern web browsers [16], one can safely port the existing C/C++ audio processing and analysis code into the Web Audio ecosystem using compiler toolchains such as Emscripten. [5] WASM is a low-level assembly-like language with a compact binary format that runs with near-native performances on modern web browsers or any WebAssembly-based stacks without compromising security, portability and load time. WASM code was found to be comparatively faster than JS code [17] and generates no garbage from the code and can run within *AudioWorkletProcessor*. [6] This makes it an ideal solution to the problems that were previously hindering us from building efficient and reliable JS MIR libraries for the web platform [20]. However, taking full advantage of all these features can be challenging because it requires understanding concurrent programming wrapped with several JS APIs and dealing with cross-compilation and linking of the native code. Even for experienced developers, compiling native code to WASM targets, generating JS bindings, and integrating them in a regular JS processing code pipeline can be cumbersome. Hence, it is essential that an ideal JS MIR software library should encapsulate and abstract all these steps and be packaged as a compact build which is easy-to-use and extendable using a high-level JS API. Besides the JS API, the potential users might also need utility tools that are often required while building MIR-based projects, such as plotting audio features on an HTML page, ready-to-use feature extractors, and possible integration with web-based machine learning frameworks.

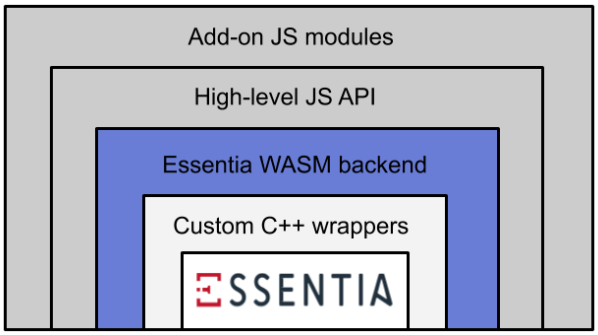


Figure 1: Overview of the *Essentia.js* library in terms of its abstraction levels.

In [24], the authors evaluated a wide range of MIR software libraries in terms of *coverage*, *effort*, *presentation*, *time-lag* and found Essentia [7] [7] to be an overall best performer with respect to these criteria. Essentia is an open-source library for audio and music analysis released under the AGPLv3 license providing a wide range of optimized algorithms (over 250 algorithms) that are successfully used in various academic and industrial large-scale applications. Essentia includes both low-level and high-level audio features, along with some ready-to-use features extractors. And, it provides an object-oriented interface to fine tune each algorithm in detail. Given all these advantages and that the code repository is consistently maintained by its developers, it is a good potential choice for porting into WASM target for the web platform.

In this paper, we present *Essentia.js*, an open-source JS library for audio and music analysis on the web, released under the AGPLv3 license. It allows building audio analysis and MIR applications for web browsers and JS engines such as Node.js. It provides straightforward integration with the latest W3C Web Audio API specification allowing efficient real-time audio feature extraction on the web browsers. Web applications written using the proposed library can also be cross-compiled to native targets such as for PCs, smartphones, and IoT devices using the JS frameworks like Electron [8] and React Native. [9]

The rest of the paper is organized as follows. Section 2 outlines the design choices, software architecture and various components of *Essentia.js*. An overview of potential use-cases and usage examples are detailed in Section 3. A detailed benchmarking of *Essentia.js* across and against various platforms and alternative JS libraries can found in Section 4. Finally, we conclude in Section 5.

2. ESSENTIA.JS

Essentia.js is much more than just JS bindings to the Essentia C++ library. It was developed with coherent design and functional objectives that are necessary for building an efficient and easy-to-use MIR library for the Web. In this

[4] https://developer.spotify.com/documentation
[5] https://emscripten.org
[6] https://www.w3.org/TR/webaudio/#audioworkletprocessor
[7] https://essentia.upf.edu
[8] https://www.electronjs.org
[9] https://reactnative.dev

section, we discuss our design choices, the architecture, and various components of *Essentia.js*. Figure 1 shows an overview of these components.

2.1 Design and Functionality

We chose the following goals and design decisions for developing the library:

- **User-friendly API and utility tools**. The Web is a ubiquitous computing platform, and an ideal JS MIR library should provide a simple, user-friendly API while being highly configurable for advanced use cases. *Essentia.js* ships with a simple JS API and add-on utility modules with a fast learning curve for new users. The main JS API is composed of a singleton class with methods implementing most of the functionality (each method is an algorithm in Essentia). These methods are automatically generated from the upstream C++ code and documentation using code templates and scripting as described in Sections 2.2 and 2.3. We also provide add-on modules with helper classes for feature extraction and visualisation that can be used for rapid prototyping of web applications. A quick preview of the JS API can be seen in Listing 2.
- **Modularity and extensibility**. Just like Essentia itself, the *Essentia.js* codebase is modular by design. It contains a large amount of configurable algorithms that can be arranged into the desired audio processing chains. The add-on utility modules shipped with the library leverage this functionality to build custom feature extractors.
- **Web standards compliance**.Web browsers provide a standard set of tools for loading and processing audio files using the HTML5 Audio element [10] and the Web Audio API. *Essentia.js* rely on these standard features for loading audio files or for streaming real-time audio from various device sources. It also provides seamless integration with the latest tools in the Web Audio ecosystem such as AudioWorklets, Web Workers, [11] WASM and *SharedArrayBuffer*. In addition, JS conforms to the ECMAScript specification [12] and it is evolving fast with new features added to the language every year. For backward and forward compatibility of our JS code, we wrote our JS API using Typescript (Section 2.3).
- **Lightweight and few dependencies**. It is important for a JS library to be lightweight since the load times of JS code can directly impact the UI/UX and performance of web applications. This includes having fewer dependencies, which also makes the library much more maintainable. Taking this into account, *Essentia WASM backend* is built without any third-party software dependencies of the Essentia library except for Kiss FFT. [13] It includes the majority of the algorithms in Essentia, [14] while the few excluded algorithms can be still integrated into the WASM backend by compiling and linking with the required third-party dependencies using our build tools (Section 2.5). Besides, all the JS code in the library is passed through a code compression process to achieve lightweight builds for the web browsers. With all these efforts we were able to achieve builds of *Essentia.js*, including the WASM backend and the JS API, as small as 2.5MB approximately. We also provide tools for custom lightweight builds of the library that only include a subset of the selected algorithms to further reduce the build size (Section 2.5).
- **Reproducibility**. Using the WASM backend of Essentia ensures identical analysis results across various devices and native platforms on which Essentia has been previously extensively used and tested. Remarkably, *Essentia.js* allows reproducing a large amount of existing code and research based on Essentia as well as, to a certain extent, other libraries. In particular, it is possible to use *Essentia.js* to reproduce common input audio representations for the existing machine learning models, enabling their usage in web applications.
- **Easy installation**. *Essentia.js* is easy to install and integrate with new or existing web projects. It is available both as a package on NPM [15] and as static builds on our public GitHub repository. In addition, we also provide continuous delivery network (CDN) through open source web services.
- **Extensive documentation**. We provide a complete API reference together with the instructions to get started, tutorials, and interactive web application examples. [16] The documentation is built automatically using JSdoc [17] and the algorithm reference is generated from the upstream Essentia C++ documentation using Python scripts.

2.2 Essentia WASM backend

As already mentioned, the core of the library is powered by the *Essentia WASM backend*. It contains a lightweight WASM build of Essentia C++ library along with custom JS bindings for using it in JS. This backend is generated in multiple steps.

Firstly, the Essentia C++ library is compiled to LLVM assembly [18] using Emscripten. Emscripten [28] is a LLVM to JS compiler which provides a wide range of tools for compiling the C/C++ code-base or the intermediate LLVM builds into various targets such as *asm.js* [19] and WASM. Secondly, we need a custom C++ interface in order to generate the corresponding JS bindings which would allow us access the algorithms in Essentia on the JS side. We used

[10] https://www.w3.org/html/wiki/Elements/audio
[11] https://w3c.github.io/workers/
[12] http://ecma-international.org/ecma-262
[13] https://github.com/mborgerding/kissfft
[14] As of May 2020, over 200 algorithms are supported.
[15] https://www.npmjs.com
[16] https://mtg.github.io/essentia.js
[17] https://jsdoc.app
[18] https://llvm.org
[19] http://asmjs.org

Embind [4] for generating this C++ interface that binds Essentia native code to JS.

Writing custom JS bindings for all Essentia algorithms can be tedious considering their large amount. Hence, we created Python scripts to automate the generation of the required C++ code for the C++ wrapper from the upstream library Python bindings. Using this scripts, it is possible to configure which algorithms to include in the bindings by their name and category. Therefore, it is possible to create extremely lightweight builds of the library with only a few specific algorithms required for a particular application. The *Essentia WASM backend* is built by compiling the generated wrapper C++ code and linking with the precompiled Essentia LLVM using Emscripten.

Essentia WASM backend provides compact WASM binary files along with the JS bindings to over 200 Essentia algorithms. We provide these binaries and a JS glue code for both asynchronous and synchronous import of Essentia WASM backend, covering a wide range of use cases. The build for asynchronous import can be instantly loaded into a HTML page. The synchronous-import build supports the new ES6 style class imports characteristic of the modern JS libraries. This build can be also seamlessly integrated with AudioWorklet and Web Workers for better performance demanding web applications.

2.3 High-level JS API

Even though it is possible to use the *Essentia WASM backend* with its bindings directly, they have limitations due to the specifics of using *Embind* with Essentia: a user needs to manually specify all parameter values for the algorithms because the default values are not supported. This is cumbersome and to solve this issue we developed a high-level JS API written using Typescript [5]. Typescript is a typed superset of JS that can be compiled to various ECMA targets of JS. In addition, this gives us the benefit of having a typed JS API which can provide better exception handling. Again we used custom Python scripts and code templates to automatically generate the Typescript wrapper in a similar way to the C++ wrapper for the WASM backend. The high-level JS API of *Essentia.js* provides a singleton class *Essentia* with all the algorithms and helper functions encapsulated as its methods All the algorithm methods are configurable similarly to the Essentia's C++/Python API itself. Listing 1, shows an example of using the high-level API of *Essentia.js*.

2.4 Add-on utility modules

Essentia.js is shipped with a few add-on modules to facilitate common MIR tasks. These add-on modules are written entirely in Typescript using the *Essentia.js* high-level JS API. Currently, we provide two add-on modules:

- **essentia.js-extractor** contains predefined feature extractors for common MIR tasks, computing BPM, beat positions, pitch, predominant melody, key, chords, chroma features, MFCC, etc. Each extractor implements the entire processing chain starting from audio input and outputs the resulting track-level or frame-level features. These extractors are configurable as well.

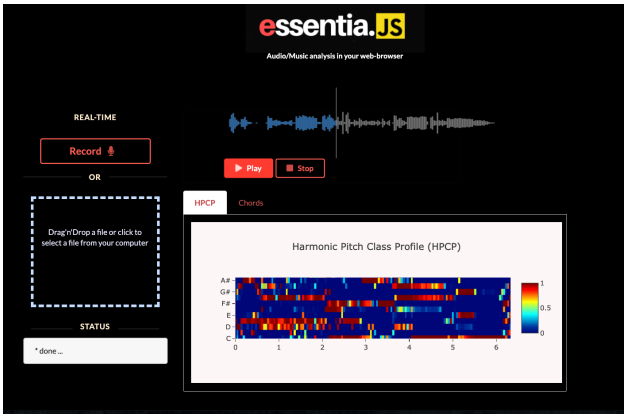


Figure 2: Screenshot of a example web application that use *Essentia.js* and its add-on modules.

- **essentia.js-plot** provides helper functions for visualization of MIR features, allowing both real-time and offline plotting in a given HTML element. It uses the *Plotly.js* data visualization library, which has a design layout and functionalities much alike of Matplotlib, [20] and is easily configurable. Currently, we provide object-oriented classes for plotting basic MIR features like melody/pitch contours, spectrograms, chroma, MFCC, etc. The module is functionally similar to the *display* module in Librosa [23].

A full reference of the modules can be found in the documentation of the library. Both modules can be easily extended with more functionalities as per the demands of the user community.

2.5 Build and packaging tools

We provide tools for custom builds and packaging of *Essentia.js* for the developers and the end-level users:

- **Command-line interface (CLI)**. We provide CLI for building *Essentia.js* using some customised shell scripts.
- **Docker**. We provide a Docker image with static builds of Essentia with Emscripten and other development dependencies required for building *Essentia.js*.
- **Web application**. We also host a website for building custom *Essentia.js* online. [21] It allows users to select specific set algorithms and build settings.

The official *Essentia.js* builds are bundled using Rollup [22] and then packaged and hosted using NPM.

3. GETTING STARTED

In this section, we outline several usage examples and application scenarios for getting started with *Essentia.js*.

The library can be imported into a web application using the following methods:

- **HTML <script> tag**. The most simple way to use *Essentia.js* is by using it with the HTML <*script*> tag.

[20] https://matplotlib.org
[21] https://mtg.github.io/essentia.js-builder
[22] https://rollupjs.org

- **NPM**. *Essentia.js* can be also installed from NPM using the command `npm install essentia.js`.
- **ES6 class imports**. *Essentia.js* can be imported using the ES6 class style imports in JS. This allows to integrate the library into any modern JS framework. Listing 1 shows an example of using ES6 style imports for an offline feature extraction task.
- **CDN**. We also provide CDN links for instantly serving *Essentia.js* online using free third-party web services such as Jsdelivr [23] and Unpkg. [24]

There are a lot of potential web applications that can be built with *Essentia.js*. The library provides algorithms for typical sound and music analysis tasks, including spectral, tonal, and rhythmic characterization. In particular, it is suitable for onset detection, beat tracking and tempo estimation, melody extraction, key and chord estimation, sound and music classification, cover song similarity, loudness metering, and audio problems detection among others. Figure 2 shows the screenshot of an example web application that we include with the library. Below we outline some of the common application use cases of the library. We provide an extensive collection of analysis examples on our website. [25]

3.1 Offline feature extraction

Many MIR use cases rely on an offline audio analysis and feature extraction. Listing 1 shows a simple JS example of using the library for offline analysis of pitch and onsets. For features computed on overlapping frames, *Essentia.js* provides the *FrameGenerator* method similarly to Essentia's Python API. Frames generated by this method can be used as an input to other algorithms in the processing chain. The offline feature extraction can be run inside a Web Worker to improve the efficiency in performance-demanding web applications.

3.2 Real-time feature extraction

Essentia.js can be used for efficient real-time audio/music analysis in web browsers along with the Web Audio API. This can be done by using the *ScriptProcessorNode* or the newly introduced *AudioWorklet* in the Web Audio API:

- **ScriptProcessorNode** allows users to provide custom JS code for audio feature extraction in its `onprocess` callback. Even though the *ScriptProcessorNode* is deprecated according to the current W3C Web Audio API specifications, it is still widely used by the developers because of its cross-browser support.
- **AudioWorklet** design pattern [10] allows users to write high-performance real-time audio analysis on a dedicated audio thread. Users can write custom analysis code by extending the *AudioWorkletProcessor* and further abstract it as a node in the Web Audio API graph using *AudioWorkletNode*. [26] Currently, the only limitation is that it is only supports in the latest Firefox and Chromium-based web browsers.

```
// Imports Essentia WASM backend
import {EssentiaWASM} from 'essentia-wasm.module.js';
// Imports Essentia.js core API
import Essentia from 'essentia.js-core.es.js';

// Creates Essentia.js instance
const essentia = new Essentia(EssentiaWASM);

// Instance of Web Audio API AudioContext
const audioContext = new AudioContext();
// URL of an audio file
let audioURL = "https://freesound.org/data/previews
    /328/328857_230356-lq.mp3";

// Decode audio file as Float32 typed array
const audioData = await essentia.
    getAudioChannelDataFromURL(audioURL, audioContext,
     0); // audioContext, channel number

// Convert audioData array into vector
const audioVector = essentia.arrayToVector(audioData);

// Onset detection with SuperFluxExtractor algorithm
let bt = essentia.SuperFluxExtractor(audioVector);
console.log(bt.onsets);

// Pitch estimation with PitchYinProbabilistic
    algorithm
let pyYin = essentia.PitchYinProbabilistic(audioVector,
     4096, 256); // frameSize, hopSize

console.log(pyYin.pitch);

// Shutdown Essentia.js instance and free memory
essentia.shutdown();
essentia.delete();
```

Listing 1: A simple example of offline audio feature extraction using *Essentia.js* via ES6 style imports.

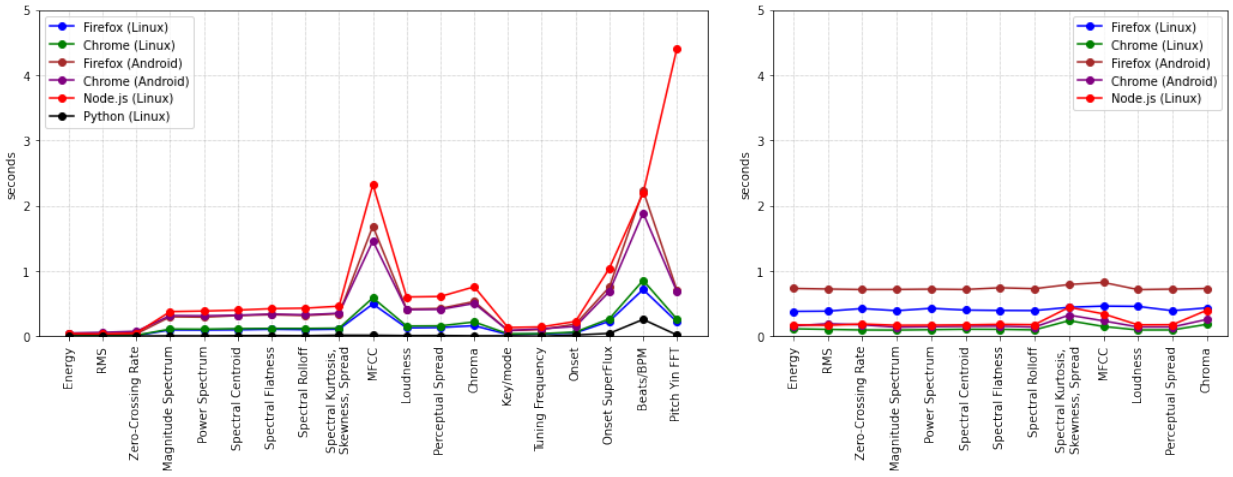
3.3 Machine learning applications

In the recent years, machine learning (ML) techniques, especially deep learning have been widely used in a wide range of MIR tasks. With the support of WebGL and WASM, modern web browsers are also capable of running ML applications. *Essentia.js* can be easily integrated with popular JS ML frameworks such as *TensorFlow.js* [27] and *Onnx.js* [27] for training and inference. Pre-trained audio ML models using features computed with Essentia as an input (e.g., mel-spectrograms, Constant-Q transform, or chroma) can be easily ported and used for inference in web browsers. In particular, Essentia now ships with a collection of pre-trained TensorFlow models for music audio tagging and classification [3]. These models can be run for inference using *Essentia.js* and *TensorFlow.js* libraries. Our *essentia.js-extractor* add-on module provides a mel-spectrogram extractor for computing the inputs to these models.

4. BENCHMARK

We tested the performance of *Essentia.js* in terms of the analysis time for common MIR audio features on various

[23] `https://www.jsdelivr.com`
[24] `https://unpkg.com`
[25] `https://mtg.github.io/essentia.js/examples`
[26] `https://www.w3.org/TR/webaudio/#audioworkletnode`
[27] `https://github.com/Microsoft/onnxjs`



(a) *Essentia.js*

(b) *Meyda*

Figure 3: Average analysis times (in seconds) for common audio features on a 5-second music clip. "Python (Linux)" corresponds to the analysis baseline using native Essentia with Python bindings.

platforms, and compared it to the native Essentia library. In addition, we measured the analysis times for features available in *Meyda* and compared them to their *Essentia.js* counterparts. To this end, we built a set of test suites using the JS library *benchmark.js* and implemented the equivalent features using both libraries. In our benchmark we measure the time it takes for the entire processing chain to compute a feature given a 5-second audio segment as an input. The code used by *Essentia.js* is equivalent to the one for *Essentia* used in Python. The benchmarking of Python implementation was done using the library *pytest* with the *benchmark extension*. We provide the code and website to reproduce these experiments online. [28]

The results are reported in Figure 3. They include tests on five different environments:

- Linux with Chrome 84.0.4147.89 run with disabled extensions.
- Linux with Firefox 78.0.2 in private browsing mode.
- Android 9 (LineageOS 16) with Chrome 84.0.4147.89 in incognito mode.
- Android 9 (LineageOS 16) with Firefox Nightly 200727 06:00
- Linux with Node.js v.13.13.0.

The Linux computer used for all runs is a 2017 DELL XPS-15 with a 2.80GHz x 8 Intel Core i7-7700HQ processor, 16GB of RAM and Ubuntu 19.04 as OS. The mobile phone is a Xiaomi Redmi Note 7 Pro with a Snapdragon Octa-core 1.7 GHz processor and 6GB RAM. All the tests were done with the same 5 seconds audio file.

As we can see from Figure 3, the results shows that the performance of *Essentia.js* algorithms were considerably slower when running on Node.js and Firefox and Chrome on Android compared to Firefox and Chrome on Linux. Interestingly, Node.js performed worse than Firefox and Chrome on Android, which was not expected. This is probably because different vendors have slightly different implementations of WASM support in their platforms or due to some other reasons yet to be found. In addition, WASM is a relatively new technology in active development. [29] Many proposals for improving its performance such as SIMD optimizations and multi-thread support are under way. We also aim to do detailed benchmarking of real-time use cases and using the Web Audio API Audio Worklets in our future work.

5. CONCLUSIONS

We have presented *Essentia.js*, an open-source JavaScript library for music/audio analysis on the Web. It is based on the Essentia C++ library which is commonly used in MIR, ported to JS via WASM, and compatible with the latest technologies in the Web Audio ecosystem. To the best of our knowledge, this is the most comprehensive library for audio analysis and MIR, which can be run on web browsers as well as JS server applications. We hope that the library will contribute to the creation of new online music technology tools in educational, industrial, and creative contexts. The source code of the library is publicly available in our Github repository. [30] Everyone is encouraged to contribute to the library.

In our future work, we will focus on improving the performance of the library along with expanding the add-on modules, particularly on providing pre-trained audio ML models for audio analysis, classification, and synthesis on the web client. We also aim to develop interesting web applications that go beyond typical MIR tasks to attract and build a diverse user community. The detailed information about the library is available at the official web page. [31] It contains the complete documentation, usage examples and tutorials needed for one to get started.

[28] `https://mtg.github.io/essentia.js-benchmarks`

[29] `https://webassembly.org/roadmap/`

[30] `https://github.com/MTG/essentia.js`

[31] `https://essentia.upf.edu/essentiajs`

6. REFERENCES

[1] MusicCritic: An automatic assessment system for musical exercises. https://musiccritic.upf.edu. Accessed: 2020-09-04.

[2] Stack Overflow Annual Developer Survey. https://insights.stackoverflow.com/survey. Accessed: 2020-15-04.

[3] Pablo Alonso-Jiménez, Dmitry Bogdanov, Jordi Pons, and Xavier Serra. Tensorflow audio models in Essentia. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, pages 266–270, 2020.

[4] Chad Austin. CppCon 2014: Embind and Emscripten: Blending C++11, JavaScript, and the Web Browser. https://www.youtube.com/watch?v=Dsgws5zJiwk. Accessed: 2020-15-04.

[5] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In *European Conference on Object-Oriented Programming (ECOOP 2014)*, pages 257–281. Springer, 2014.

[6] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. Madmom: A new python audio and music signal processing library. In *ACM International Conference on Multimedia (MM 2016)*, pages 1174–1178, 2016.

[7] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin Salamon, José Zapata, and Xavier Serra. Essentia: An audio analysis library for music information retrieval. In *International Society for Music Information Retrieval Conference (ISMIR 2013)*, pages 493–498, 2013.

[8] Paul M Brossier. The aubio library at MIREX 2006. In *Music Information Retrieval Evaluation Exchange (MIREX 2006)*, 2006.

[9] Mark Cartwright, Ayanna Seals, Justin Salamon, Alex Williams, Stefanie Mikloska, Duncan MacConnell, Edith Law, Juan P Bello, and Oded Nov. Seeing sound: Investigating the effects of visualizations and complexity on crowdsourced audio annotations. *Proceedings of the ACM on Human-Computer Interaction*, 1(CSCW):1–21, 2017.

[10] Hongchan Choi. *AudioWorklet: The future of web audio*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2018.

[11] Jakub Fiala, Nevo Segal, and Hugh A. Rawlinson. Meyda: an audio feature extraction library for the Web Audio API. In *Web Audio Conference (WAC 2015)*, pages 1–6, 2015.

[12] Eduardo Fonseca, Jordi Pons Puig, Xavier Favory, Frederic Font Corbera, Dmitry Bogdanov, Andres Ferraro, Sergio Oramas, Alastair Porter, and Xavier Serra. Freesound datasets: a platform for the creation of open audio datasets. In *International Society for Music Information Retrieval Conference (ISMIR 2017)*, pages 486–93. International Society for Music Information Retrieval (ISMIR), 2017.

[13] Frederic Font, Gerard Roma, and Xavier Serra. Freesound technical demo. In *ACM International Conference on Multimedia (MM 2013)*, page 411–412, New York, NY, USA, 2013.

[14] W3C Audio Working Group. W3C Web Audio API specifications. https://www.w3.org/TR/webaudio. Accessed: 2020-15-04.

[15] W3C Technical Architecture Group. Web Audio API Design Review. https://github.com/w3ctag/design-reviews/blob/master/2013/07/WebAudio.md. Accessed: 2020-05-04.

[16] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*, pages 185–200, 2017.

[17] David Herrera, Hanfeng Chen, Erick Lavoie, and Laurie Hendren. Numerical computing on the web: Benchmarking for the future. In *ACM SIGPLAN International Symposium on Dynamic Languages (DLS 2018)*, pages 88–100, 2018.

[18] Nicholas Jillings, Jamie Bullock, and Ryan Stables. JSXtract: A realtime audio feature extraction library for the Web. In *International Society for Music Information Retrieval Conference (ISMIR 2016) Late Breaking Demo*, 2016.

[19] Nicholas Jillings, David Moffat, Brecht De Man, and Joshua D. Reiss. Web Audio Evaluation Tool: A browser-based listening test environment. In *Sound and Music Computing Conference (SMC 2015)*, 2015.

[20] Jari Kleimola and Oliver Larkin. Web audio modules. In *Sound and Music Computing Conference (SMC 2015)*, 2015.

[21] Anand Mahadevan, Jason Freeman, Brian Magerko, and Juan Carlos Martinez. Earsketch: Teaching computational music remixing in an online web audio based learning environment. In *Web Audio Conference (WAC 2015)*, 2015.

[22] Benoit Mathieu, Slim Essid, Thomas Fillon, Jacques Prado, and Gaël Richard. Yaafe, an easy to use and efficient audio feature extraction software. In *International Society for Music Information Retrieval Conference (ISMIR 2010)*, pages 441–446, 2010.

[23] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Python in Science Conference (SciPy 2015)*, 2015.

[24] David Moffat, David Ronan, and Joshua D. Reiss. An evaluation of audio feature extraction toolboxes. In *International Conference on Digital Audio Effects (DAFx 2015)*, pages 1–7, 2015.

[25] Alastair Porter, Dmitry Bogdanov, Robert Kaye, Roman Tsukanov, and Xavier Serra. Acousticbrainz: a community platform for gathering music information obtained from audio. In *International Society for Music Information Retrieval Conference (ISMIR 2015)*, 2015.

[26] Michael Schoeffler, Fabian-Robert Stöter, Bernd Edler, and Jürgen Herre. Towards the next generation of web-based experiments: A case study assessing basic audio quality following the ITU-R recommendation BS. 1534 (MUSHRA). In *Web Audio Conference (WAC 2015)*, pages 1–6, 2015.

[27] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, et al. Tensorflow. js: Machine learning for the web and beyond. In *Conference on Systems and Machine Learning (SysML 2019)*, 2019.

[28] Alon Zakai. Emscripten: an llvm-to-javascript compiler. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2011)*, pages 301–312, 2011.