TISMIR

Correya, A., et al. (2021). Audio and Music Analysis on the Web using Essentia.js. *Transactions of the International Society for Music Information Retrieval,* 4(1), pp. 167–181. DOI: https://doi.org/10.5334/tismir.111

RESEARCH

Audio and Music Analysis on the Web using Essentia.js

Albin Correya*,†, Jorge Marcos-Fernández*, Luis Joglar-Ongay*,‡, Pablo Alonso-Jiménez*, Xavier Serra* and Dmitry Bogdanov*

Open-source software libraries have a significant impact on the development of Audio Signal Processing and Music Information Retrieval (MIR) systems. Despite the abundance of such tools, there is a lack of an extensive and easy-to-use reference library for audio feature extraction on Web clients.

In this article, we present *Essentia.js*, an open-source JavaScript (JS) library for audio and music analysis on both web clients and JS engines. Along with the Web Audio API, it can be used for both offline and real-time audio feature extraction on 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 cross-compiled from the Essentia C++ library, which facilitates a JS interface to a wide range of low-level and high-level audio features, including signal processing MIR algorithms as well as pre-trained TensorFlow.js machine learning models. 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 and the Node.js engine, and four devices, including mobile Android and iOS, comparing it to the native performance of Essentia and the Meyda JS library.

Keywords: Software; web audio; audio analysis; music signal processing; music audio classification; deep learning

1. Introduction

The Web has become one of the most used computing platforms with billions of web pages served daily, and JavaScript (JS) is an essential part of building modern web applications. Using HTML, CSS, and JS, developers can make dynamic and interactive applications that run in a web browser by implementing custom client-side scripts. At the same time, developers can also use cross-platform run-time engines like Node.js to write server-side code in JS. The flexibility of being able to use 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 (Stack Overflow, 2021). JS is also widely used as an entry-level programming language by 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 (Adenot and Choi, 2021), 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 as long as the web client has computational resources for the required processing. The Web Audio API provides a JS interface to various predefined audio graph nodes for processing, synthesis, and analysis. Even though these nodes' capabilities are limited, the API also includes an interface allowing developers to add custom JS code for audio processing.

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, there are a few existing libraries written in JS (e.g., Meyda, JS-Xtract, lfo, and MMLL) offering music audio analysis (Fiala et al., 2015; Jillings et al., 2016; Matuszewski and Schnell, 2017; Collins and Knotts, 2019) but they implement only a very limited set of MIR audio feature extraction algorithms. Other attempts at bringing audio processing and feature extraction onto the web client side have also been made by using Emscripten to cross-compile tools written in other languages to JS via WebAssembly (e.g., Piper, Faust, and CsoundEmscripten) (Thompson et al., 2017; Letz et al., 2017; Bernardo et al., 2019; Lazzarini et al., 2014; 2015).

* Music Technology Group, Universitat Pompeu Fabra, Barcelona, ES

† Currently at Moodagent A/S, Copenhagen, DK

‡ SonoSuite S.L., Barcelona, ES

Corresponding author: Dmitry Bogdanov (dmitry.bogdanov@upf.edu)

Still, these tools are also limited in the number of audio analysis features available out of the box, especially for MIR tasks. **Table 1** gives an overview of the most relevant existing libraries that include MIR functionality in terms of type of platform, number of MIR algorithms, different applications covered, and last time it was updated. To the best of our knowledge, these libraries are not popular among MIR researchers for their typical tasks and some of them are not actively maintained.

Currently, there is a lack of more extensive, more configurable alternatives focused on MIR needs. This is partially due to the fact that writing a new JS audio analysis library from scratch or manually porting native tools requires a lot of effort. It was not until recently that audio analysis applications became possible on web clients due to the recent development of features in browsers. In addition, the growth of computational power of mobile devices made it feasible in a large variety of contexts. So far, for these reasons, MIR researchers and developers have often relied on server-side solutions using existing native tools when creating web applications.

In this article,[1] we present *Essentia.js*,[2] an open-source JS library for audio and music analysis on the web, released under the AGPLv3 license. It allows audio analysis and MIR applications to be built for web browsers and JS engines such as Node.js. It provides straightforward integration with the latest W3C Web Audio API specification allowing for real-time audio feature extraction on web browsers. Web applications written using the proposed library can also be cross-compiled to native device targets such as for PCs, smartphones, and IoT devices using JS frameworks such as Electron.[3] In addition, we also present a collection of TensorFlow.js audio machine learning (ML) models for music processing along with a high-level add-on JS module *essentia.js-model* integrated into the *Essentia.js* library. This module allows developers to do end-to-end processing from audio input to the models' prediction results with a simple JS API. Although the library is still under development, we expect it to be useful for research, industrial and creative applications related to MIR and audio analysis in general.

The rest of the article is organized as follows. Section 2 overviews recent web developments that allow porting some of the existing native audio and music analysis libraries and machine learning models to web clients. Section 3 outlines the design choices, software architecture and various components of *Essentia.js*, and in Section 4 we briefly demonstrate various approaches to using the library in offline and real-time scenarios. In Section 5 we discuss using *Essentia.js* for machine learning inference and present the pre-trained models available in Essentia, which we have ported to TensorFlow.js. Section 6 discusses possible applications and use cases of the proposed library for audio analysis and MIR on the web. In Section 7 we provide detailed benchmarking of *Essentia.js* across various platforms and against one alternative JS library. Finally, we conclude and discuss future work in Section 8.

2. Motivation

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 (Bogdanov et al., 2013), Librosa (McFee et al., 2015), Madmom (Böck et al., 2016), Yaafe (Mathieu et al., 2010) and Aubio (Brossier, 2006), have been especially popular within the MIR community due to rapid prototyping needs and a large collection of available tools for scientific computation. Meanwhile, the libraries with a native C/C++ implementation offered faster analysis (Moffat et al., 2015) and were often required for industrial audio applications. Various web applications for audio processing and analysis have been developed using these tools. Spotify API[5] (formerly Echonest), the Freesound API (Font et al., 2013) and AcousticBrainz (Porter et al., 2015) are examples of web services providing precomputed audio features to the end users via a REST API. In addition, numerous web applications were built for aiding tasks such as crowdsourcing audio annotations (Fonseca et al., 2017; Cartwright et al., 2017), audio listening tests (Schoeffler et al., 2015; Jillings et al., 2015), and music

Table 1: Overview of libraries for audio analysis and MIR on web clients compared to *Essentia.js,* including libraries written purely in JS or cross-compiled for Wasm, in terms of their target applications and the number algorithms suitable for MIR out of the box. *Csound and Faust are very extensive programming languages for audio DSP which require cross-compilation.

Name	Implementation	MIR algorithms	Applications	Last updated
CsoundEmscripten (Lazzarini et al., 2014)	asm.js[4]	*	processing, synthesis	2021
Meyda (Fiala et al., 2015)	plain JS	~20	analysis	2021
JS-xtract (Jillings et al., 2016)	plain JS	~70	analysis	2021
Piper (Thompson et al., 2017)	Wasm	~20	analysis, processing	2018
Faust (Letz et al., 2017)	Wasm	*	processing, synthesis	2021
lfo (Matuszewski and Schnell, 2017)	plain JS	~15	analysis, processing	2017
MMLL (Collins and Knotts, 2019)	plain JS	~15	analysis	2020
Essentia.js	Wasm	~200	analysis, processing, synthesis	2021

education platforms (MTG UPF, 2021; Mahadevan et al., 2015) to mention a few. All of these services manage their audio analysis on the server, which may require a significant effort and resources to scale to many users. Within the MIR community, there have been previous initiatives for online accessibility of MIR algorithms and audio data for collaborative research (West et al., 2010).

With the recent arrival of WebAssembly (Wasm) support on most modern web browsers (Haas et al., 2017), one can efficiently port the existing C/C++ audio processing and analysis code into the Web Audio ecosystem using compiler toolchains such as Emscripten.[6] Wasm is a low-level assembly-like language with a compact binary format that runs with near-native performance on modern web browsers or any WebAssembly-based stacks without compromising security, portability or load time. Wasm code is comparatively faster than JS code (Herrera et al., 2018) because it avoids just-in-time compilation and has less garbage collection overhead. It can run within *AudioWorkletProcessor*.[7] 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 (Kleimola and Larkin, 2015). However, taking full advantage of all these features can be challenging because it requires 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. Therefore, an ideal JS MIR software library should encapsulate and abstract all these steps through automated scripts 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 in MIR-based projects, such as plotting audio features on a web page, ready-to-use feature extractors, and possible integration with web-based machine learning frameworks, which the existing JS libraries generally lack.

Considering native software tools, Moffat et al. (2015) evaluated a wide range of MIR software libraries in terms of coverage, effort, presentation, and time lag and found Essentia[8] (Bogdanov et al., 2013) to be an overall best performer with respect to these criteria. Essentia is an open-source library for audio and music analysis available under the AGPLv3 license[9] 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 feature 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 choice for porting to a Wasm target for the web platform.

2.1 Machine learning on the Web

ML methods, especially deep learning for audio and music processing, allow for innovative approaches that greatly complement the traditional signal processing methods but are not yet well-represented on the web compared to other domains such as text and image processing. Web ML frameworks like TensorFlow.js[10] and ONNX.js[11] have enabled the use of pre-trained ML models in typical web software development workflows, which has helped application developers to leverage this new set of AI technologies.

Currently, TensorFlow Hub[12] provides many pre-trained models ready for deployment in JS applications, but is lacking models for most common audio-related tasks. This is not surprising considering that many ML audio models require a spectral representation derived from the waveform as an input (except for a few models that operate on raw audio). When using the model for inference, the input representation has to be computed the same way as in the training phase to produce valid results.

Essentia has recently released a collection of pre-trained TensorFlow models for audio and music related tasks (Alonso-Jiménez et al., 2020a, b). These models are optimised for production use and are trained with the audio representations computed using Essentia itself, which makes them a potential choice to be ported to TensorFlow.js models for the web platform. However, using pre-trained models via ML libraries like TensorFlow.js directly can be cumbersome for many users since it demands some ML domain expertise. In order to avoid this overhead and facilitate the usability and inclusivity of these tools, new JS abstraction libraries and tools were created with user-friendly APIs (Roberts et al., 2018; ITP NYU, 2018; Bernardo et al., 2019). Similarly, an MIR JS library would benefit from having easy interfaces with ML libraries, such as TensorFlow.js.

3. Essentia.js

Essentia.js is 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 easy-to-use MIR library for the Web. In this section, we discuss our design choices, the architecture, and various components of *Essentia.js*. **Figure 1** shows an overview of these components.

3.1 Design and functionality

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

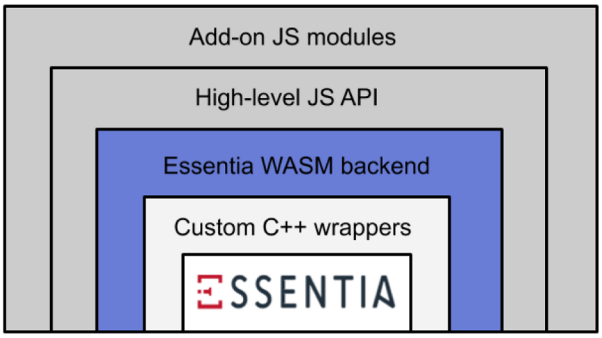


Figure 1: Overview of the *Essentia.js* library in terms of its abstraction levels.

- **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 that can provide 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 and documentation are automatically generated from the existing upstream Essentia C++ code and documentation using code templates and scripting as described in Sections 3.2 and 3.3. We also provide add-on modules with helper classes for feature extraction, visualisation and machine learning inference that can be used for rapid prototyping of web applications.
- **Modularity and extensibility**. Just like Essentia itself, the *Essentia.js* codebase is modular by design. It contains a large number of configurable algorithms that can be arranged into the desired audio processing chains. The add-on utility modules shipped with the library leverage this 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[13] and the Web Audio API. *Essentia.js* relies 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 and Wasm. In addition, we wrote the API using TypeScript for static typing, and to be able to transpile the library for new and old JS targets (for forward and backward compatibility) as the ECMAScript (ES)[14] specification continues to evolve.
- **Lightweight and with few dependencies**. It is important for a JS library to be lightweight since the load times of JS code can directly impact the user experience and performance of web applications. This includes having few external dependencies, which also makes the library much more maintainable. Taking this into account, the *Essentia Wasm back end* is built without any of Essentia's third-party software dependencies, except for Kiss FFT.[15] It includes the majority of the algorithms in Essentia,[16] while the few excluded algorithms can be still integrated into the Wasm back end by compiling and linking with the required third-party dependencies using our build tools (Section 3.5). All the JS code in the library is passed through a code minimization process to achieve lightweight builds for the web browsers. With all these efforts, we were able to achieve builds of *Essentia.js* with complete functionality (including the Wasm back end and the JS API) as small as 2.5MB, and about 3MB with add-on modules.
  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 3.5).
- **Reproducibility**. Using the Wasm back end of Essentia ensures identical computation pipeline and consistent numerical analysis results across various devices and native platforms on which Essentia has been previously used and tested. Therefore, *Essentia.js* technically 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 ML models, enabling their use in web applications (Section 5.3).
- **Easy installation**. *Essentia.js* is easy to install and integrate with new or existing web projects. It is available both as a package on the Node Package Manager (NPM) registry[17] and as static builds on our public GitHub repository. In addition, we also provide a 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.[18] The documentation is built automatically using JSdoc[19] and the algorithm reference is generated from the upstream Essentia C++ documentation using Python scripts.

3.2 Essentia Wasm back end

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

First, the Essentia C++ library is compiled to LLVM assembly[20] using Emscripten. Emscripten (Zakai, 2011) is an 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 including Wasm. Second, we need a custom C++ interface to generate the corresponding JS bindings, which allows us to access the algorithms in Essentia from JS. We used *Embind* (Austin, 2014) for generating this C++ interface that binds native Essentia code to JS.

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

Essentia Wasm back end provides compact Wasm binary files along with the JS bindings to over 200 Essentia algorithms. We provide these binaries and JS glue code for both asynchronous and synchronous imports of the *Essentia Wasm back end*, covering a wide range of use cases. The build for asynchronous import can be instantly loaded into an HTML page. The synchronous-import

build supports the ES6 style class imports characteristic of modern JS libraries. This build can also be seamlessly integrated with AudioWorklets and Web Workers for better performance in demanding web applications.

3.3 High-level JS API

Even though it is possible to use the *Essentia Wasm back end* with its bindings directly, they have limitations due to the specifics of using *Embind* with Essentia: users need to manually specify all parameter values for the algorithms because default values are not supported. To overcome this issue, we developed a high-level JS API written using TypeScript (Bierman et al., 2014). 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 static error checking and 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 back end. 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 algorithm methods are configurable in a manner similar to Essentia's C++ and Python APIs. **Listing 1** shows an example of using the high-level API of *Essentia.js*.

3.4 Add-on utility modules

Essentia.js ships 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 three modules:

- **essentia.js-extractor** contains configurable feature extractors for selected MIR features such as harmonic pitch class profiles (HPCP) and mel-frequency cepstral coefficients (MFCCs). Each extractor implements the entire processing chain starting from audio signal input and outputs the resulting track-level or frame-level features.
- **essentia.js-model** provides abstractions to use pre-trained ML models distributed with *Essentia.js.* It combines both input feature extraction using *Essentia.js* and model inference using TensorFlow.js. See Section 5.3 for more details.
- **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*[21] data visualization library, which has a design layout and functionalities like Matplotlib,[22] and is easily configurable. Currently, we provide abstract classes for plotting basic MIR features like melody/pitch contours, spectrograms, HPCPs, MFCCs, etc.

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

```
// Imports Essentia Wasm back end
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.

3.5 Build and packaging tools

We provide tools for custom builds and packaging of *Essentia.js:*

- **Command-line interface**. We provide customised scripts for building *Essentia.js.*
- **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. It allows users to select a specific set of algorithms and build settings to create and download a smaller build.

  The official *Essentia.js* builds are bundled using Rollup[23] and then packaged and hosted using NPM.

4. Usage

In this section, we outline several use examples and application scenarios for getting started with *Essentia.js.* The library can be imported into a web application using different methods which allow the library to be integrated into any modern JS framework, including HTML <script> tag and ES6 class imports. It can be instantly served from online sources like NPM and third-party JS CDNs. We refer the reader to the online documentation for further details.[24]

4.1 Offline analysis

Many MIR use cases rely on non-real-time (offline) audio and music analysis. **Listing 1** shows a simple JS example for offline analysis of pitch and onsets.

For features computed on overlapping frames, *Essentia.js* provides the *FrameGenerator* method, similar 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 by not blocking the main UI thread.

4.2 Real-time analysis

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

- **ScriptProcessorNode**[25] 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 developers because of its cross-browser support. Note that as *ScriptProcessorNode* runs the JS audio processing code on the main user interface (UI) thread, it may result in unreliable performance and audio glitches (W3C TAG, 2013).
- **AudioWorklet** design pattern (Choi, 2018) 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 not supported by all browsers.[27]

5. Machine Learning Inference

In recent years, ML techniques, especially deep learning, have been used in a wide range of MIR tasks. Following recent developments, modern web browsers are also capable of running ML applications, in particular with the support of WebGL[28] and Wasm which enable faster performance than plain JS. In this section, we introduce our collection of TensorFlow models created to be used within Essentia and explain the process of porting those models to the TensorFlow.js format. Finally, we present the machine learning inference functionality of *Essentia.js*, implemented as a module of the library. It interfaces with TensorFlow.js and is able to use our pre-trained models.

Essentia.js can be easily integrated with popular JS ML frameworks such as TensorFlow.js (Smilkov et al., 2019) and ONNX.js (Ning, 2020) for inference. Pre-trained audio ML models using common audio features as input (e.g., mel-spectrogram, Constant-Q transform, or chroma) can be easily ported and used for inference in web browsers. Essentia itself now ships with TensorFlow support, via its C API, and a collection of pre-trained models for music auto-tagging and classification (Alonso-Jiménez et al., 2020a) among other MIR tasks. However, using this API would require custom Wasm TensorFlow builds which are potentially laborious to maintain and optimize for the existing Web GPU back ends. Instead, we decided to rely on TensorFlow.js. Conveniently, it provides tools that allow easy conversion of the models shipped with Essentia into the required format. Our *essentia.js-model* add-on module provides extractors for computing the input features and converting them into the data format expected by TensorFlow.js.

5.1 Essentia TensorFlow models

Essentia contains a repository of pre-trained ML models publicly available under the Creative Commons BY-NC-ND 4.0 license.[29] Many of those models were trained by different researchers. We converted them to the appropriate format and implemented algorithms in Essentia to compute the spectral representations required as input.

For *Essentia.js*, we focused on the following models for the tasks of auto-tagging (Pons and Serra, 2019), tempo estimation (Schreiber and Müller, 2019), and classification based on transfer learning (Alonso-Jiménez et al., 2020a; b):

- Two auto-tagging models trained on the Million Song Dataset (MSD) (Bertin-Mahieux et al., 2011) and MagnaTagATune (MTT) (Law et al., 2009) with activations for the top-50 tags in each taxonomy. The tags contain information related to the genre, instrumentation, mood or era of the music (e.g., rock, pop, alternative, indie, electronic, female, vocalists, dance, 00s, alternative rock, and jazz).
- Three tempo models that estimate the tempo of music, ranging from 30 to 286 BPM. We included a variety of CNN architectures with different model sizes.
- Various classifiers for genre, mood, and other semantic categories trained using transfer learning. This technique allows leveraging the knowledge acquired on the larger models for more specific classification tasks. **Table 2** lists the classes available in each task.

Table 2: Transfer learning classifiers.

	Task	Classes
genre	dortmund	alternative, blues, electronic, folkcountry, funksoulrnb, jazz, pop, raphiphop, rock
	gtzan	blues, classic, country, disco, hip hop, jazz, metal, pop, reggae, rock
	rosamerica	classic, dance, hip hop, jazz, pop, rhythm and blues, rock, speech
mood	acoustic	acoustic, non acoustic
	aggressive	aggressive, non aggressive
	electronic	electronic, non electronic
	happy	happy, non happy
	party	party, non party
	relaxed	relaxed, non relaxed
	sad	sad, non sad
misc.	danceability	danceable, non danceable
	voice/instrum.	voice, instrumental
	gender	male, female
	tonal/atonal	atonal, tonal
	urbansound8k	air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, street music
	fs-loop-ds	bass, chords, fx, melody, percussion

Table 3: The Essentia models. RF: Receptive field, AT: Auto-tagging, TL: Transfer learning.

Model	RF (s)	Params.	Size (MB)	Purpose
MusiCNN	3	787K	3.1	AT/TL
VGG	3	605K	2.4	AT/TL
VGGish	1	62M	276	TL
TempoCNN	12	[27K–1.2M]	[0.1–4.7]	Tempo

From the proposed models, we only controlled the training process of the transfer learning classifiers, covered in more detail by Alonso-Jiménez et al. (2020a, b). We followed a well-known transfer learning approach consisting of taking the penultimate layer of a large pre-trained model as a feature (embedding) to train a smaller model in a related downstream task with fewer data availability. This approach is known to improve the performance on small datasets, such as the ones we had available for the tasks in **Table 2**.

We used the pre-trained auto-tagging models as feature extractors and a simple Multi-Layer Perceptron (MLP) with two layers as a downstream model. For deployment, the final model combines the embedding extractor and the MLP, with the former accounting for most of the complexity in terms of model parameters and inference time.

Table 3 compares the architectures employed in the provided models in terms of the receptive field (the audio duration required to perform a prediction in seconds), the number of parameters, size in megabytes, and purpose. Note that we only account for the feature extractor part of the transfer learning models, as the fully-connected classifiers are negligible in size. We considered a wide variety of model capabilities in terms of parameters, so it is not expected that all the models are suitable for web deployment on computationally-weak devices.

Figure 2 shows the activations produced by all the auto-tagging and classification taxonomies on the song *Bohemian Rhapsody* by the rock band *Queen*. It can be seen how some classes can be useful to describe the structure of the song. Note that the transfer learning classifiers activate an output even when none of the choices seem appropriate. Therefore, we can find some inconsistencies such as the label *ambient* from the *mood electronic* classifier. Even if it does not seem an adequate label, the classifier does not contain better choices.

5.2 Porting models to TensorFlow.js

The ability to deploy client-side deep learning models is a feature with a growing support by ML frameworks. At the time we started this work, the main tools being actively developed were TensorFlow.js and ONNX.js. We identified the following advantages of using TensorFlow.js for our case:

- It is the most actively maintained project with extensive documentation and example projects.
- It is part of the TensorFlow ecosystem, the same deep learning library used in Essentia.

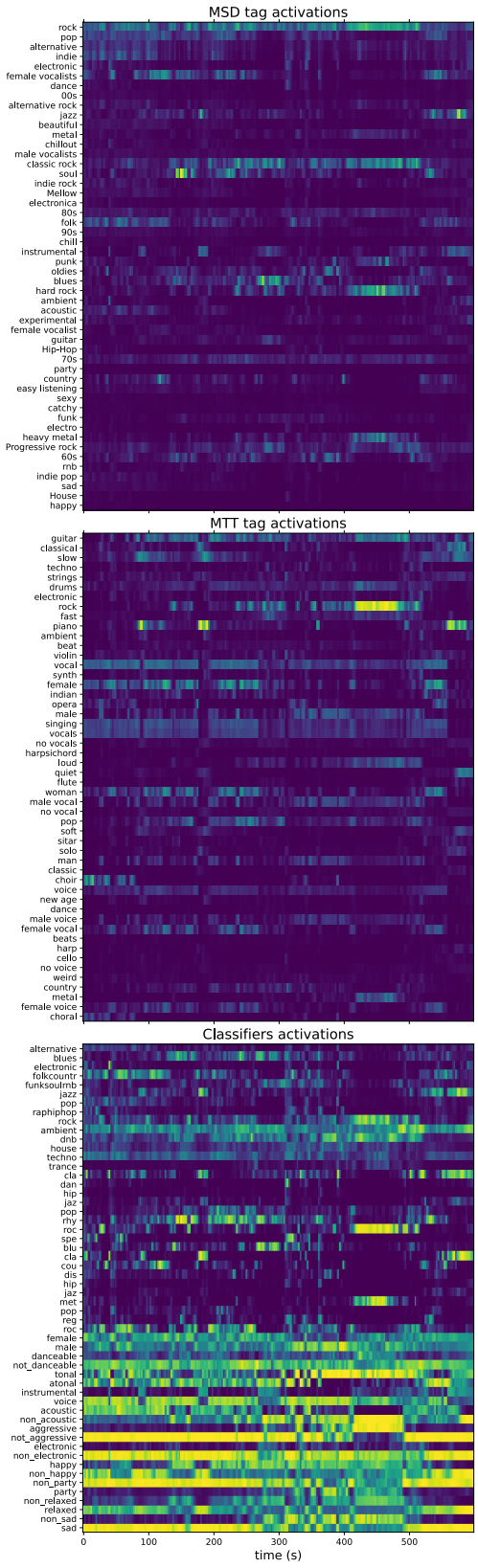


Figure 2: Activations for the MSD and MTT auto-tagging taxonomies and all the transfer learning classifiers for *Bohemian Rhapsody* by *Queen*.

- It supports multiple back end options such as WebGL and Wasm for inference on browsers or Node.js, which provides flexibility for future scenarios.
- It provides a tool to convert the format of existing Essentia models to its required input format.

We used the converter provided by TensorFlow.js to port the models. While all our models were stored as TensorFlow v1 frozen protocol buffers, this tool also supports conversion from TensorFlow v2 SavedModels, and Keras HDF5 files. Additionally, PyTorch models can be exported in ONNX format and converted to TensorFlow v2 Saved models with the official tools.[30] Covering the two major machine learning frameworks means that the vast majority of the models developed for research are suitable for web deployment.

The only additional requirement for the model files is to know the name of the inputs and outputs, which can be done with tools such as Netron.[31]

In the frozen format, the topology and weights are contained in a single binary file. TensorFlow.js models are defined in two files: a human-readable JSON file containing the topology and a binary file with the model weights. None of the weight quantization options offered by the converter were applied. The models are approximately the same size after conversion.

We compared the activations generated by both the original and the converted models finding minimal numerical differences in the range of $1e^5$. We have also seen similar differences when testing the original models under different computer architectures or TensorFlow versions. After a further inspection of prediction outcomes, we conclude that they are too small to alter the sense of the predictions.

All the converted models are available for download on the Essentia website.[32] They can be used for inference on a wide variety of devices, similar to TensorFlow.js.

5.3 Essentia.js-model

To use our pre-trained models in TensorFlow.js, one would have to implement the exact audio representations needed by the models as an input, which requires some development effort and domain knowledge. Models based on different CNN architectures expect different types and resolutions of input spectrogram representations for inference. To facilitate the models' usability, we developed *essentia.js-model*, an add-on JS module that implements the required settings for each of the models we provide and is able to compute the required input format automatically without the developer needing to know the specific details. It combines both feature extraction using *Essentia.js* and model inference using TensorFlow.js. The APIs for achieving both of these processes are decoupled to allow more complex use-cases (for example, doing feature extraction and inference sessions in separate web workers). The detailed API documentation of the module is available online.

We outline the two main use cases of *essentia.js-model* below:

- **Input feature extraction**. The module provides an interface for feature extraction via the *EssentiaTFInputExtractor* class. This class helps the user ensure the correct type and size of input audio feature representation matching the desired models of choice. Its constructor is called by passing the *EssentiaWASM* import from the *Essentia Wasm back end* and choosing the target extractor type. The *compute* method of the class returns the feature representation for a given audio frame. **Listing 2** shows an example of using the class for an offline feature extraction task with the MusiCNN-based models.
- **Inference**. This module provides its model inference functionalities through the classes *TensorflowMusiCNN*, *TensorflowVGGish* and *TensorflowTempoCNN* for models with the MusiCNN, VGGish and TempoCNN architectures, respectively. Each of these classes' constructors is called by passing a global import of the TensorFlow.js package and a path to where the pre-trained model is stored (both can be a URL or a local file path). The *predict* method returns the output of the inference session as a JS promise for a given input feature representation which is pre-computed by *EssentiaTFInputExtractor*. **Listing 3** shows an example of using *TensorflowMusiCNN*.

  Currently, we are not providing access to the models via a CDN server, and it is up to the user to host the models required for their applications.

```
// Import Essentia Wasm back end
import {EssentiaWASM} from 'essentia.js';
import {EssentiaTFInputExtractor} from
'./essentia.js-model.es.js';

// Instantiate feature extractor for MusiCNN-based
     models
const extractor = new EssentiaTFInputExtractor(
     EssentiaWASM, 'musicnn');

// Load a mono audio file from a given URL using Web
     Audio API
const audioURL = "https://freesound.org/data/previews
     /328/328857_230356-lq.mp3";
const audioContext = new AudioContext();
const audioBuffer = await extractor.
     getAudioBufferFromURL(audioURL, audioContext);

// Downsample audio to required sample rate
const audio = extractor.downsampleAudioBuffer(
     audioBuffer);

// Compute mel-spectrogram
let inputFeature = extractor.computeFrameWise(audio);
```

Listing 2: Example of offline audio feature extraction for the MusiCNN-based models using *essentia.js-model* via ES6 style imports.

```
// Import essentia.js-model and tensorflow.js
import {TensorflowMusiCNN} from './essentia.js-model.es.
    js';
import * as tf from "@tensorflow/tfjs";

// Path where the tfjs models are stored
const modelURL = "./autotagging/msd/msd-musicnn-1/model.
     json";

// Create an instance of EssentiaTensorflowJSModel
const musiCNN = new TensorflowMusiCNN(tf, modelURL);

// Promise for initializing the model
await musiCNN.initialize();

// Run model inference on the given feature input
let prediction = await musiCNN.predict(inputFeature);
```

Listing 3: Example of inference of MusiCNN-based models from the feature input computed in Listing 2 using *essentia.js-model* via ES6 style imports.

6. Applications

There are many 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 as well as higher-level semantic annotation. Similar to Essentia, it is suitable for onset detection, beat tracking and tempo estimation, pitch and melody extraction, key and chord estimation, cover song similarity, loudness metering and audio problem detection, sound and music classification, music auto-tagging, and genre and mood identification, among other tasks. It is possible to extract feature embeddings with the provided deep learning models, which can then be used for sound and music similarity or transfer learning tasks (Alonso-Jiménez et al., 2020b). Further updates to the algorithms and models in the Essentia library will be included into the future versions of *Essentia.js*.

We provide starter code and a collection of analysis examples on our website.[33] **Figure 3** shows some examples of real-time use of the library. These real-time demo applications access the user's microphone and analyse the input signal in real time. The analysis results from *Essentia.js* are then visualized. Mel-spectrogram and pitch analysis are displayed as animated plots, with the sound level mapped to color intensity. HPCP and auto-tagging use pitch-class or tag activations mapped to color brightness and transparency, respectively. These examples use *AudioWorklet* for real-time analysis. Not every algorithm and model we provide is suitable for real-time inference on web browsers. **Figure 3** shows non-real-time mood

Figure 3: *Essentia.js* demo applications: (a) real-time mel-spectrogram (top-left), pitch estimation (top-right), HPCP (bottom-left) and music auto-tagging (bottom-right), (b) five *Essentia.js* transfer learning models for mood classification, (c) industrial application by SonoSuite for audio problem detection.

classification of a song with five transfer learning models, as well as BPM and key estimation. In this particular case, model inference is performed on a separate thread using the *Web Workers* API.

Essentia.js can also be used for JS server applications. TensorFlow has a dedicated wrapper, *tfjs-node*, with direct bindings to the TensorFlow C API, which can be used in Node.js run-time applications.

Essentia.js has already gained attention in the industrial context. For example, SonoSuite S.L.[34] is implementing an application for automatic detection of audio quality issues in music recordings (Alonso-Jiménez et al., 2019; Joglar-Ongay, 2020a, b). Customers are able to upload music to their platform for digital music distribution, and analysis of the music for audio problems is performed in the browser, giving immediate feedback about any issues, displayed in a music player highlighting the areas where each problem is. **Figure 3** shows a prototype of this application, which can be found together with other demos on the *Essentia.js* website.

7. Benchmarking

We measured the execution time of *Essentia.js* in several JS platforms, and we provide comparisons with the native implementation of Essentia and available counterpart algorithms in *Meyda*. We considered Meyda (Fiala et al., 2015) because it is an MIR analysis library implemented in pure JS with an active community.[35]

We built a set of test suites using the benchmark.js[36] library for the JS measurements and pytest-benchmark[37] for the native ones. These libraries repeatedly execute the algorithms under test until they can provide statistically significant measurements.

We considered various common MIR features and tasks and measured the entire processing chain (including auxiliary algorithms where needed) for each of them using a 30-second audio segment as an input. We did not measure the time necessary for loading audio files, preprocessing audio with Web Audio API, or loading TensorFlow.js models for simplicity, as those times can be affected by a number of factors, such as network connectivity. We ensured the equivalence of the implementations for the tested features in *Essentia.js* and native Essentia in Python. We provide the code and website to reproduce these experiments online.[38]

The tests were executed on four different devices, in total covering twelve different environments. **Table 4** shows the device and platform versions used for the JavaScript tests. For the native baseline tests, we used Linux and MacOS machines with Python 3.7.10 and 3.6.7, respectively.

Table 4: Platform versions for each device used in the JavaScript benchmarks.

Device	Chrome	Firefox	Node.js
Linux	89.0.4389.114 (64-bit)	87.0 (64-bit)	14.15.1
macOS	89.0.4389.114 (64 bit)	87.0 (64-bit)	14.13.0
Android	92.04484.6	Nightly 210421	–
iOS	87.0.4280.77	33.0	–

The Linux computer used for all runs is a 2017 DELL XPS-15 with a 2.80 GHz × 8 Intel Core i7-7700HQ processor, 16 GB of RAM, GPU GeForce GTX 1050 running Ubuntu 20.04.2 LTS. The Macintosh machine runs macOS 10.15.7, with a 2.2 GHz 6-core i7 CPU, 16 GB of RAM, and Intel UHD Graphics 630 GPU. The mobile phone is a Xiaomi Redmi Note 7 Pro with a Snapdragon Octa-core 1.7 GHz processor and 6 GB RAM running Android 9 (LineageOS 16). The iOS device is an iPad 6th generation (MR7G2TY/A), 2 GB of RAM, iOS version 14.4.1 and an A10 Fusion 2.3 GHz CPU.

It is important to note that these technologies are in continuous development, and browsers are evolving quickly. During our tests (March and April 2021), the performance of some algorithms improved noticeably without any modification to the *Essentia.js* implementation, thanks to browser updates, specifically Firefox Nightly, which we use in our model benchmarks, in Section 7.2.

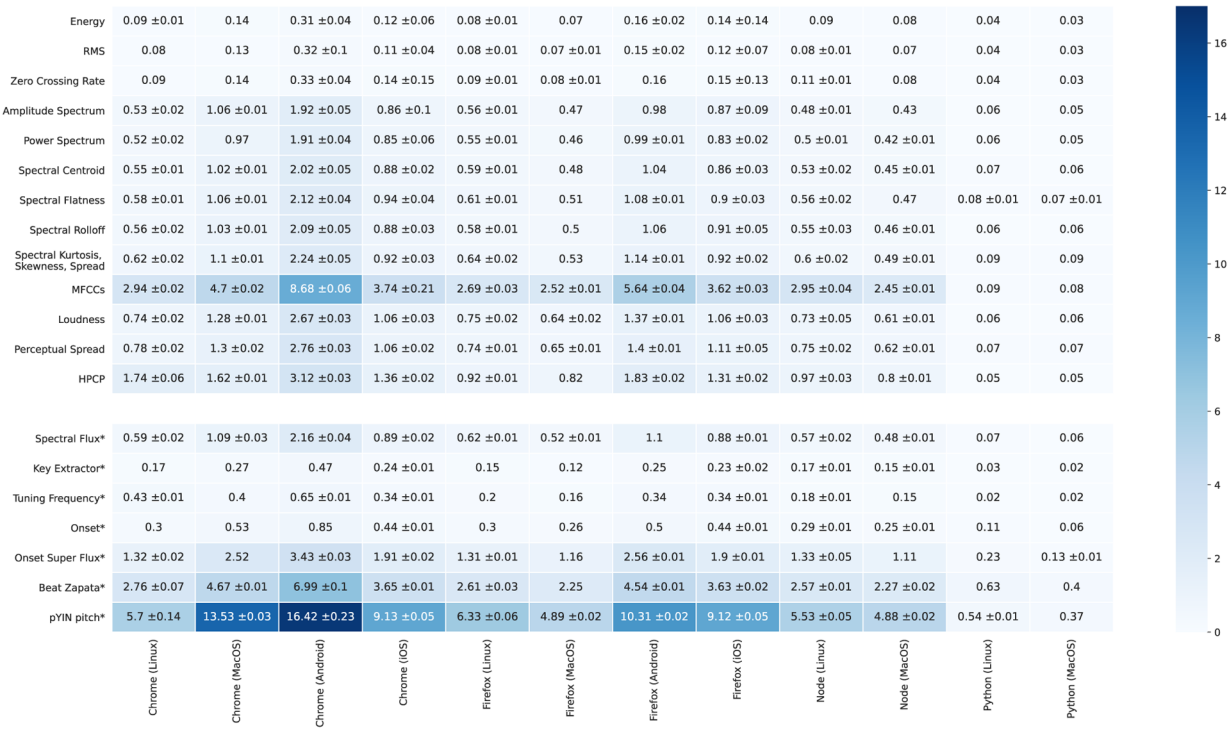
7.1 Essentia algorithms

We tested the performance of *Essentia.js* on a set of audio features, most of which were present in Meyda. The results are presented in **Figure 4**.

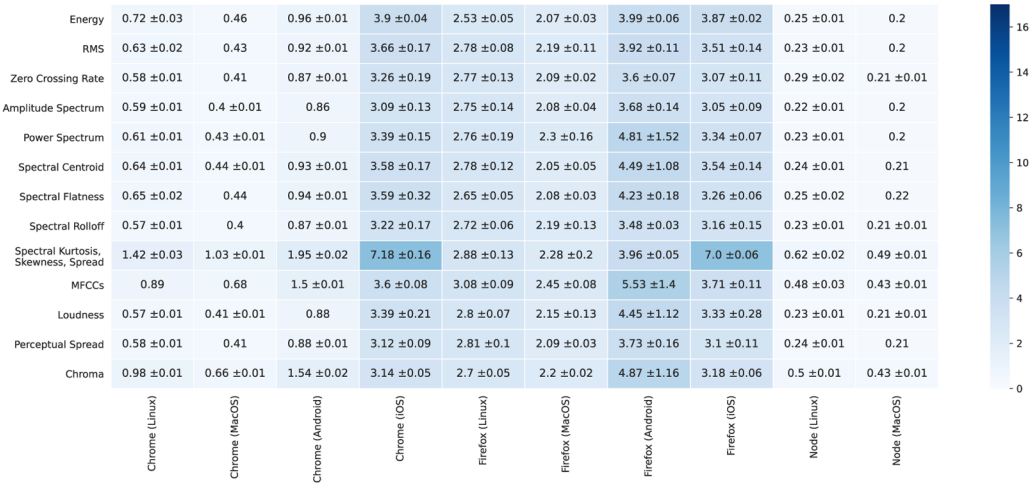
We can see how *Essentia.js* performs faster than Meyda for most algorithms in the browsers with the exception of MFCCs and HPCP (HPCP is a feature similar to Chroma in Meyda). Meyda is faster than *Essentia.js* in Node.js. The browser where Meyda performs worst is Firefox on Android followed closely by both browsers on iOS, while Chrome on Android has a performance close to the one on Linux and MacOS. When it comes to *Essentia.js*, Chrome on Android has the worst performance closely followed by Firefox on Android then Chrome on MacOS and both browsers on iOS. The fastest platforms are both browsers on Linux, Firefox on MacOS and Node.js, all having similar performance. For *Essentia.js*, all the features follow a similar pattern across platforms (with MFCC, pYin pitch, and beat detection being the slowest), while Meyda is less predictable. As expected, the Python configurations with native Essentia were faster for all features.

Computing times for the majority of the *Essentia.js* algorithms in our test set took from 0.46 to 3.48 seconds which is 1.5 to 6.8% of the duration of the input audio segment. We have observed slower behavior for some features such as MFCCs and pYIN pitch which took in the worst case 8.68 and 16.419 seconds (28.9% and 54.7%), respectively. This behavior might be due to possible memory management issues that are yet to be discovered in our future work. There are many proposals for improving Wasm performance which will possibly improve the overall performance of the library.

Finally, we have estimated whether algorithms can run in real-time. We assumed that the execution times of all algorithms were linear with respect to the input audio duration, and we determined how much time it would take to run the algorithms on a single frame (therefore, only frame-wise features were considered). If this time was shorter than the duration of the frame in seconds,



(a) *Essentia.js*



(b) *Meyda*

Figure 4: Mean execution time (in seconds) for common audio features on a 30-second music track. If the standard deviation is smaller than 0.005 it is not printed. The algorithms marked with * in (a) are only available in *Essentia.js*.

we consider the algorithm apt to run in real time. Our estimation on the worst-case platform (Chrome for Android) confirms that it is possible for all the considered algorithms.

7.2 TensorFlow.js models

We tested inference time using the *TensorflowMusiCNN* or *TensorflowVGGish* functions (depending on each model architecture) with the following selected ML models described in Section 5.1:

- auto-tagging MusiCNN
- auto-tagging VGG
- genre rosamerica MusiCNN
- genre rosamerica VGGish
- mood happy MusiCNN
- mood happy VGGish

In addition, we tested the computation time for two auxiliary functions, *TensorflowInputMusiCNN* and *TensorflowInputVGGish*, required to generate input representations for the models.

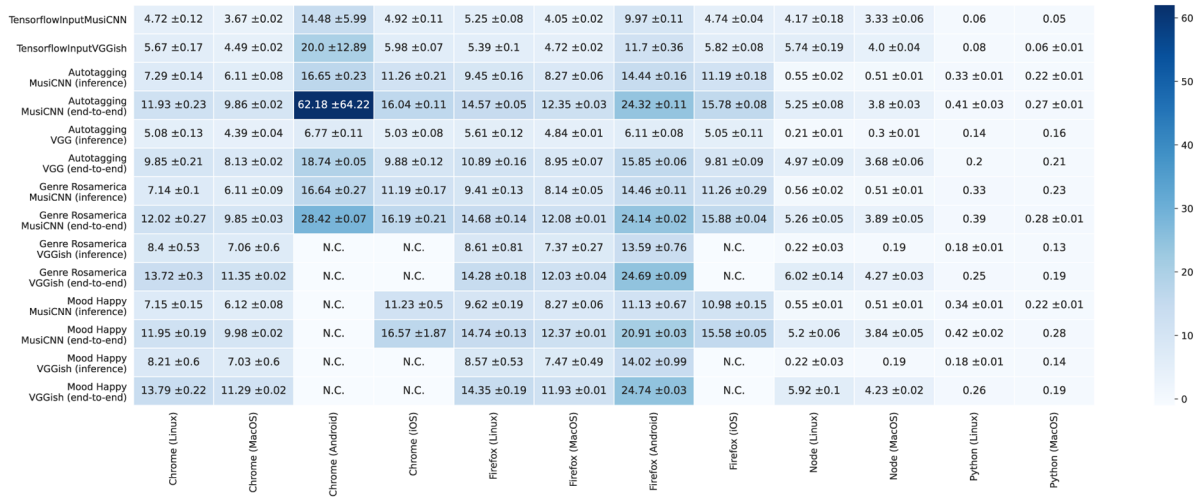
For each model, we ran two benchmarks: time spent on inference and time spent on the entire end-to-end process, including the auxiliary input feature extraction. All tests were performed on the same 30-second audio

segment, re-sampled to 16 kHz, and mixed down to mono. Note that browser tests were performed on the main UI thread and we did not benchmark the models' real-time capabilities.

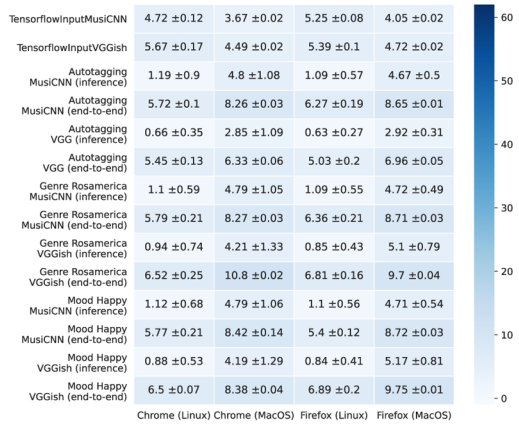
TensorFlow.js provides several back ends for executing models, including WebGL and Wasm which we tested. All browser benchmarks were run using the Wasm back end, which provides CPU acceleration and portable 32-bit precision. This back end is widely supported across browser vendors and is comparable to the other CPU-accelerated platforms, Node.js and Python. The default browser back end, using WebGL, executes tensor operations on the device's GPU, therefore its arithmetic precision is hardware-specific (16-bit on iOS devices).[39] Since not all Android and iOS devices support WebGL or have powerful enough GPUs, only browser benchmarks on Linux and MacOS were done using WebGL (in addition to Wasm). **Figure 5** shows the benchmark results for these two back ends side by side for comparison.

We can see that overall the performance of models with WebGL is faster, as expected. For both back ends, model inference is faster than the entire end-to-end process, which includes the computation of the required input representations. This is because data loading for inference is done on the CPU, but only inference itself is executed on the GPU. Input representation computation is not affected by the TensorFlow.js back end used. The transfer learning classifiers based on VGGish, the most complex model among the ones we provide, are significantly slower. Therefore we cannot expect it to behave well in many use-cases, for example those requiring shorter analysis time like real-time applications.

Missing result data points correspond to the models that we were unable to execute: VGGish (both *genre rosamerica* and *mood happy*) on iOS and Chrome for Android, and *mood happy* MusiCNN on Chrome for Android. This may be due to lack of memory on iOS, since VGGish-based models are the largest in *Essentia.js* (with a size of 288 MB), and the



(a) CPU acceleration (Wasm on browsers)



(b) GPU acceleration with WebGL. Since not all Android and iOS devices support WebGL or have powerful enough GPUs, only browser benchmarks on Linux and MacOS are shown.

Figure 5: Mean execution time (in seconds) for *Essentia.js* model algorithms on a 30-second music track, comparing TensorFlow.js back ends (a) Wasm, and (b) WebGL. If standard deviation is smaller than 0.005 it is not printed. *N.C.* stands for not computed values, where the benchmark suite was unable to complete execution. (a) CPU acceleration (Wasm on browsers). (b) GPU acceleration with WebGL. Since not all Android and iOS devices support WebGL or have powerful enough GPUs, only browser benchmarks on Linux and MacOS are shown.

iOS device only had 2 GB of RAM. In the case of the missing Android models, it may be due to browser timeouts and memory limits established by browser vendors.

Overall, inference on JS takes from 0.19 to 16.65 seconds (0.6 to 55.5% of the input audio duration) and end-to-end on JS takes from 3.68 to 28.4 seconds (12.3 to 94.6% of the input audio duration), with the exception of auto-tagging MusiCNN, which took 62.18 seconds end-to-end. From these results, we assume that most of the provided ML models are potentially suitable for real-time applications. We have successfully deployed the MusiCNN models for auto-tagging in such a scenario in our online demo applications (Section 6).

8. Conclusions

We have presented *Essentia.js*, an open-source JavaScript library for audio and music analysis on the Web. It is based on the Essentia C++ library commonly used in MIR, which we ported to JS via Wasm and made compatible with the latest technologies in the Web Audio ecosystem. In addition, it contains a collection of pre-trained ML models ported for TensorFlow.js and an add-on module for their easy use in Web applications requiring audio and music analysis. These models address some of the common music classification tasks, tempo estimation, and extraction of music feature embeddings, some of which are available for real-time applications.

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, research, industrial, and creative contexts. Detailed information about the library is available at the official web page. It contains the complete documentation, usage examples, and tutorials for getting started. The source code of the library is publicly available in our Github repository.[40] 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 and adding more pre-trained ML models for audio analysis, classification, and synthesis on the web. For better portability, we will consider creating the models in the ONNX format. We also aim to develop web applications that go beyond typical MIR tasks to attract and build a diverse user community.

We will also conduct user tests. As part of our dissemination activities, we have presented the library and models to potential users (web developers and MIR researchers) in a tutorial format at the Web Audio Conference 2021[41] and received very positive feedback. However, a more formal survey process is required for critical user-centered evaluation of the tools' interface, documentation, and overall usability, which is a limitation of our current study.

Notes

[1] http://asmjs.org.

[2] This article is an extension of our conference paper (Correya et al., 2020), with the main novel contributions being the new functionality for machine learning inference, new and more extensive benchmarking experiments, updated library codebase, and new demo examples and applications. The machine learning functionality is partially presented in (Correya et al., 2021).

[3] https://essentia.upf.edu/essentiajs.

[4] https://www.electronjs.org.

[5] https://developer.spotify.com/documentation.

[6] https://emscripten.org.

[7] https://www.w3.org/TR/webaudio/#audioworkletprocessor.

[8] https://essentia.upf.edu.

[9] Also available under a commercial license.

[10] https://www.tensorflow.org/js.

[11] https://github.com/microsoft/onnxjs.

[12] https://tfhub.dev.

[13] https://www.w3.org/html/wiki/Elements/audio.

[14] http://ecma-international.org/ecma-262.

[15] https://github.com/mborgerding/kissfft.

[16] As of April 2021, over 200 algorithms are supported.

[17] https://www.npmjs.com/package/essentia.js.

[18] https://mtg.github.io/essentia.js.

[19] https://jsdoc.app.

[20] https://llvm.org.

[21] https://plotly.com/javascript.

[22] https://matplotlib.org.

[23] https://rollupjs.org.

[24] https://mtg.github.io/essentia.js/docs/api.

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

[26] https://www.w3.org/TR/webaudio/#audioworkletnode.

[27] https://caniuse.com/#feat=mdn-api_audioworklet.

[28] https://www.khronos.org/webgl.

[29] https://essentia.upf.edu/machine_learning.html.

[30] https://github.com/onnx/onnx-tensorflow.

[31] https://netron.app/.

[32] https://essentia.upf.edu/models.

[33] https://mtg.github.io/essentia.js/examples.

[34] https://sonosuite.com/.

[35] As of April 2021, Meyda has 20 MIR algorithms.

[36] https://benchmarkjs.com/.

[37] https://pytest-benchmark.readthedocs.io.

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

[39] https://www.tensorflow.org/js/guide/platform_environment.

[40] https://github.com/MTG/essentia.js.

[41] https://github.com/MTG/essentia.js-tutorial-wac2021.

Acknowledgements

The work on *Essentia.js* has been partially funded by the Ministry of Science and Innovation of the Spanish Government under the grant agreement PID2019-111403GB-I00 (Musical AI). The authors would like to thank Alastair Porter for valuable feedback on the manuscript and Alex Albàs for his assistance in developing the website for benchmarking.

Competing Interests

The authors have no competing interests to declare.

Author Contributions

Albin Correya led the core development and maintenance of *Essentia.js*. Jorge Marcos-Fernández worked on the development of web demos and conducted benchmarking experiments together with Luis Joglar-Ongay, who also contributed an audio problem detection demo. Deep learning models were ported by Pablo Alonso-Jiménez. Dmitry Bogdanov took part in the design of the library and supervised the project and this publication. All authors participated in writing the paper and agreed to the published version of the manuscript.

References

Adenot, P. and **Choi, H.** (2021). Web audio API, W3C candidate recommendation snapshot. Retrieved March 31, 2021, from https://www.w3.org/TR/webaudio.

Alonso-Jiménez, P., Bogdanov, D., Pons, J., and **Serra, X.** (2020a). TensorFlow audio models in Essentia. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*. DOI: https://doi.org/10.1109/ICASSP40776.2020.9054688

Alonso-Jiménez, P., Bogdanov, D., and **Serra, X.** (2020b). Deep embeddings with Essentia models. In *International Society for Music Information Retrieval Conference (ISMIR 2020) Late Breaking Demo*.

Alonso-Jiménez, P., Joglar-Ongay, L., Serra, X., and **Bogdanov, D.** (2019). Automatic detection of audio problems for quality control in digital music distribution. In *Audio Engineering Society Convention 146*.

Austin, C. (2014). CppCon 2014: Embind and Emscripten: Blending C++11, JavaScript, and the Web Browser. Retrieved March 31, 2021, from https://www.youtube.com/watch?v=Dsgws5zJiwk.

Bernardo, F., Kiefer, C., and **Magnusson, T.** (2019). An AudioWorklet-based signal engine for a live coding language ecosystem. In *Web Audio Conference (WAC 2019)*, pages 77–82.

Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., and **Lamere, P.** (2011). The Million Song Dataset. In *International Society for Music Information Retrieval Conference (ISMIR 2011)*.

Bierman, G., Abadi, M., and **Torgersen, M.** (2014). Understanding TypeScript. In *European Conference on Object-Oriented Programming (ECOOP 2014)*. DOI: https://doi.org/10.1007/978-3-662-44202-9_11

Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F., and **Widmer, G.** (2016). madmom: A new Python audio and music signal processing library. In *ACM International Conference on Multimedia (MM 2016)*. DOI: https://doi.org/10.1145/2964284.2973795

Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., Roma, G., Salamon, J., Zapata, J., and **Serra, X.** (2013). Essentia: An audio analysis library for music information retrieval. In *International Society for Music Information Retrieval Conference (ISMIR 2013)*. DOI: https://doi.org/10.1145/2502081.2502229

Brossier, P. M. (2006). The aubio library at MIREX 2006. In *Music Information Retrieval Evaluation Exchange (MIREX 2006)*.

Cartwright, M., Seals, A., Salamon, J., Williams, A., Mikloska, S., MacConnell, D., Law, E., Bello, J. P., and **Nov, O.** (2017). 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. DOI: https://doi.org/10.1145/3134664

Choi, H. (2018). AudioWorklet: The future of web audio. In *International Computer Music Conference Proceedings (ICMC 2018)*.

Collins, N. and **Knotts, S.** (2019). A JavaScript musical machine listening library. In *International Computer Music Conference (ICMC 2019)*.

Correya, A., Alonso-Jiménez, P., Marcos-Fernández, J., Serra, X., and **Bogdanov, D.** (2021). Essentia TensorFlow models for audio and music processing on the web. In *Web Audio Conference (WAC 2021)*.

Correya, A., Bogdanov, D., Joglar-Ongay, L., and **Serra, X.** (2020). Essentia.js: A JavaScript library for music and audio analysis on the web. In *International Society for Music Information Retrieval Conference (ISMIR 2020)*.

Fiala, J., Segal, N., and **Rawlinson, H. A.** (2015). Meyda: an audio feature extraction library for the Web Audio API. In *Web Audio Conference (WAC 2015)*.

Fonseca, E., Pons Puig, J., Favory, X., Font Corbera, F., Bogdanov, D., Ferraro, A., Oramas, S., Porter, A., and **Serra, X.** (2017). Freesound Datasets: A platform for the creation of open audio datasets. In *International Society for Music Information Retrieval Conference (ISMIR 2017)*.

Font, F., Roma, G., and **Serra, X.** (2013). Freesound technical demo. In *ACM International Conference on Multimedia (MM 2013)*. DOI: https://doi.org/10.1145/2502081.2502245

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and **Bastien, J.** (2017). Bringing the web up to speed with WebAssembly. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*. DOI: https://doi.org/10.1145/3062341.3062363

Herrera, D., Chen, H., Lavoie, E., and **Hendren, L.** (2018). Numerical computing on the web: Benchmarking for the future. In *ACM SIGPLAN International Symposium on Dynamic Languages (DLS 2018)*. DOI: https://doi.org/10.1145/3276945.3276968

ITP NYU (2018). ml5.js: Friendly machine learning for the Web. Retrieved March 31, 2021, from https://ml5js.org.

Jillings, N., Bullock, J., and **Stables, R.** (2016). JSXtract: A realtime audio feature extraction library for the Web. In *International Society for Music Information Retrieval Conference (ISMIR 2016) Late Breaking Demo*.

Jillings, N., Moffat, D., De Man, B., and **Reiss, J. D.** (2015). Web Audio Evaluation Tool: A browserbased listening test environment. In *Sound and Music Computing Conference (SMC 2015)*.

Joglar-Ongay, L. (2020a). Applications of Essentia on the web. Master's thesis, Universitat Pompeu Fabra. Master Thesis. DOI: https://doi.org/10.5281/zenodo.4091073.

Joglar-Ongay, L. (2020b). Sónar+D CCCB 2020 Workshop: How to automatically detect quality problems in

your music collection. Retrieved April 15, 2021, from https://www.youtube.com/watch?v=NR9-hVLs4b8.

Kleimola, J. and **Larkin, O.** (2015). Web audio modules. In *Sound and Music Computing Conference (SMC 2015)*.

Law, E., West, K., Mandel, M. I., Bay, M., and **Downie, J. S.** (2009). Evaluation of algorithms using games: The case of music tagging. In *International Society for Music Information Retrieval Conference (ISMIR 2009)*.

Lazzarini, V., Costello, E., Yi, S., and **Fitch, J.** (2014). Csound on the Web. In *Linux Audio Conference (LAC 2014)*.

Lazzarini, V., Costello, E., Yi, S., and **Fitch, J.** (2015). Extending Csound to the Web. In *Web Audio Conference (WAC 2015)*.

Letz, S., Orlarey, Y., and **Fober, D.** (2017). Compiling Faust audio DSP code to WebAssembly. In *Web Audio Conference (WAC 2017)*.

Mahadevan, A., Freeman, J., Magerko, B., and **Martinez, J. C.** (2015). EarSketch: Teaching computational music remixing in an online web audio based learning environment. In *Web Audio Conference (WAC 2015)*. DOI: https://doi.org/10.1145/2676723.2691869

Mathieu, B., Essid, S., Fillon, T., Prado, J., and **Richard, G.** (2010). YAAFE, an easy to use and efficient audio feature extraction software. In *International Society for Music Information Retrieval Conference (ISMIR 2010)*.

Matuszewski, B. and **Schnell, N.** (2017). LFO – a graph-based modular approach to the processing of data streams. In *Web Audio Conference (WAC 2017)*.

McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., and **Nieto, O.** (2015). librosa: Audio and music signal analysis in Python. In *Python in Science Conference (SciPy 2015)*. DOI: https://doi.org/10.25080/Majora-7b98e3ed-003

Moffat, D., Ronan, D., and **Reiss, J. D.** (2015). An evaluation of audio feature extraction toolboxes. In *International Conference on Digital Audio Effects (DAFx 2015)*.

MTG UPF (2021). MusicCritic: An automatic assessment system for musical exercises. Retrieved March 31, 2021, from https://musiccritic.upf.edu.

Ning, E. (2020). ONNX.js – A JavaScript library to run ONNX models in browsers and Node.js. Retrieved March 31, 2021, from https://www.w3.org/2020/06/machine-learning-workshop/talks/onnx_js_a_javascript_library_to_run_onnx_models_in_browsers_and_node_js.html.

Pons, J. and **Serra, X.** (2019). musicnn: Pre-trained convolutional neural networks for music audio tagging. In *International Society for Music Information Retrieval Conference (ISMIR 2019) Late Breaking Demo*.

Porter, A., Bogdanov, D., Kaye, R., Tsukanov, R., and **Serra, X.** (2015). AcousticBrainz: A community platform for gathering music information obtained from audio. In *International Society for Music Information Retrieval Conference (ISMIR 2015)*.

Roberts, A., Hawthorne, C., and **Simon, I.** (2018). Magenta. js: A JavaScript API for augmenting creativity with deep learning. In *Joint Workshop on Machine Learning for Music (ICML)*.

Schoeffler, M., Stöter, F.-R., Edler, B., and **Herre, J.** (2015). 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)*.

Schreiber, H. and **Müller, M.** (2019). Musical tempo and key estimation using convolutional neural networks with directional filters. In *Sound and Music Computing Conference (SMC 2019)*.

Smilkov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Zhang, K., Cai, S., Nielsen, E., Soergel, D., Bileschi, S., Terry, M., Nicholson, C., Gupta, S. N., Sirajuddin, S., Sculley, D., Monga, R., Corrado, G., Viégas, F. B., and **Wattenberg, M.** (2019). TensorFlow. js: Machine learning for the web and beyond. In *Conference on Systems and Machine Learning (SysML 2019)*.

Stack Overflow (2021). Stack Overflow Annual Developer Survey. Retrieved March 31, 2021, from https://insights.stackoverflow.com/survey.

Thompson, L., Cannam, C., and **Sandler, M.** (2017). Piper: Audio feature extraction in browser and mobile applications. In *Web Audio Conference (WAC 2017)*.

W3C TAG (2013). Web Audio API Design Review. Retrieved March 31, 2021, from https://github.com/w3ctag/design-reviews/blob/master/2013/07/WebAudio.md.

West, K., Kumar, A., Shirk, A., Zhu, G., Downie, J. S., Ehmann, A., and **Bay, M.** (2010). The networked environment for music analysis (NEMA). In *IEEE World Congress on Services (SERVICES 2010)*. DOI: https://doi.org/10.1109/SERVICES.2010.113

Zakai, A. (2011). Emscripten: An LLVM-to-JavaScript compiler. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2011)*. DOI: https://doi.org/10.1145/2048147.2048224

How to cite this article: Correya, A., Marcos-Fernández, J., Joglar-Ongay, L., Alonso-Jiménez, P., Serra, X., & Bogdanov, D. (2021). Audio and Music Analysis on the Web using Essentia.js. *Transactions of the International Society for Music Information Retrieval*, 4(1), pp. 167–181. DOI: https://doi.org/10.5334/tismir.111

Submitted: 24 April 2021 **Accepted:** 02 September 2021 **Published:** 22 November 2021

]u[*Transactions of the International Society for Music Information Retrieval* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS