Enabling Interactive Music Performance through Web Browsers for non-Programmers

Charles K. Neimog[1*], **Rodolfo Coelho de Souza**

[1]Escola de Comunicação e Artes, University of São Paulo

charlesneimog@outlook.com, rcoelho@usp.br

Abstract. *We introduce* `pd4web`*, software that allows running Pure Data (Pd) patches directly in web browsers without programming expertise. It removes the need for C code or build configuration, enabling Pd-based electroacoustic and live-electronic works on desktop and mobile platforms. Unlike similar tools,* `pd4web` *targets musicians and composers familiar with Pd, allowing them to deploy patches online. By lowering technical barriers, it expands creative possibilities and reduces dependence on specialized hardware or software.* `pd4web` *has been used in compositions such as Improviso I, Cânticos de Silício I, but also in projects like Gaiasenses Web and cartoP5, demonstrating real-time web-based electroacoustic performance with minimal setup. However, due to the Web Audio API and remaining challenges, including latency and performance limits, the need for ongoing improvements to the Web Audio API remains.*

1. Introduction

In contemporary concert music, live electronics represent a distinct field that contrasts with fixed Media. Fixed media involves pre-recorded electronic compositions played back during performances, which can be acousmatic or mixed-media (the performer plays with pre-recorded samples, processes, and others). Live electronics, on the other hand, incorporate real-time sound generation and manipulation during a performance. Within this domain, two main approaches stand out: live electronics and live coding.

Focusing on live electronics, the primary interest is the possibility of enabling dynamic musical interactions between an instrumental performance and a computer program. This approach fosters a responsive, adaptive musical environment where the musical output is shaped by the composer's ability to create interactions between the computer and the instrument player.

In computer music research, numerous software platforms have emerged to bridge the gap between compositional concepts and the technical demands of live musical interaction. Among the most influential contributions is the work of Miller Puckette, particularly his development of Max and Pure Data [1] [2], built on earlier research into real-time music systems, dating back to Mathews' `RTSKED`. Puckette [2] highlights the core objective behind Max and Data: "making software systems that were truly usable by non-computer scientists." Both environments have stood out, largely due to their high-level abstractions, which simplify complex tasks such as FFT-based signal processing, audio manipulation, and interactive musical control. By providing modular frameworks and integrated DSP libraries, these two platforms empower composers to prototype sophisticated signal-processing algorithms and design custom user interfaces — without requiring deep expertise in low-level or math programming.

Pd and Max are extensible by researchers/composers with programming expertise, another strong advantage. It is possible to develop and share external objects (dynamic libraries), expanding the platforms' functionalities. Since these dynamic libraries operate within a minimal set of data structures – primarily numbers, strings, or lists of numbers and strings – objects from different developers remain easily interoperable. This design ensures that composer/research-created extensions can be combined and connected within the visual programming environment, fostering a modular and scalable approach to real-time audio processing and interactive music composition.

Compared to text-based environments such as CSound, SuperCollider, and ChucK, another advantage lies in their graphical programming interface (visual programming environment), which aligns with the principles of Object-Oriented Programming (OOP). In these environments, Pd and Max objects function similarly to classes in OOP, encapsulating both data and methods within a modular structure. Graphically, these objects are represented as rectangular elements that users can connect to form signal-processing chains and interactive musical systems (cf. Figure 1). This approach provides an intuitive, visually structured workflow, reducing the need for traditional text-based coding while maintaining the flexibility and extensibility required for advanced musical applications.

Finally, when comparing Pd and Max, the key distinction lies in their accessibility and development models. Max, as a commercial software, offers a more polished graphical user interface, prioritizing usability and user experience. In contrast, Pure Data, as open-source and free, makes it financially more accessible. Additionally, its open-source nature provides significant advantages, such as fostering open community-driven contributions, enabling continuous development, and allowing for better flexibility in customization and extensibility (key for the development of `pd4web`).

But, in both environments, its external extensibility is a great advantage, also introducing challenges that,

[*]Supported by Funding CAPES.

while manageable for composers familiar with the platform, can be significant obstacles for *performers* with little or no experience in computer music. These challenges include the need to download and install these external objects, place them in the correct directories for Pd/Max to recognize, and occasionally troubleshoot missing system libraries such as FFTW3 (used in Pd and externals libraries like `partialtrack`, `o.scofo~` and others), compatibility issues between different software versions, or problems related to audio drivers and interfaces. We argue that the accumulation of these technical challenges substantially increases the complexity of executing live-electronics works when compared to the traditional task of reading and interpreting a notated musical score. Consequently, such requirements render the performance of live-electronics compositions a highly specialized domain within the broader field of musical performance.

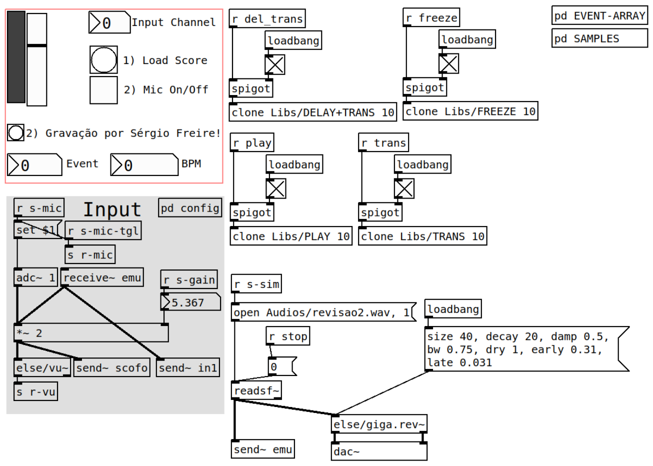


Figure 1: Example of visual programming using Pure Data.

Trying to solve this problem, the initial line of investigation addressed the possibility of executing Pure Data (Pd) directly within web environments. Within this scope, projects such as *webpd*, *hvcc*, and *empd* were identified. *webpd*, developed by Sebastien Piquemal, "converts the audio graph and processing objects from a patch into plain human-readable JavaScript or AssemblyScript." Likewise, *hvcc*, developed by Wasted Audio, is a "Python-based dataflow audio programming language compiler that generates C/C++ code and a variety of specific framework wrappers." A major limitation of both platforms is that they treat the Pd patch as a source from which the audio graph is replicated. As a result, all objects must be reimplemented within the respective platforms, which prevents compatibility with existing externals. In the case of *hvcc*, however, this approach offers the advantage of enabling the deployment of *patches* on devices with limited computational resources, such as the *Bela* platform.

empd, developed by Claude Heiland-Allen [3], does not have this limitation because it corresponds to a modified version of Pd designed to address internal compatibility issues in WebAssembly-based deployment, thereby enabling the complete execution of Pd inside browsers. Through the use of WebAssembly and browser-native technologies, *empd* (now part of Pd after version 0.55) makes it possible to distribute Pd *patches* as fully accessible web applications, eliminating the necessity of installing Pd, managing external objects, or resolving dependency conflicts associated with dynamic libraries. This functionality significantly improves the usability of live-electronics systems because once compiled, the patch will work across all devices that support a browser. Several proposals have been presented using *empd*, like *PdWebParty* [4], *PdXR* ([5]) (designed to be used in Metaverse), and *WebPdL2Ork* ([6]). An overview of these three platforms is presented in Table 1.

Table 1: Comparison of different platforms for live-electronics performance capabilities.

Feature	hvcc	webpd	empd
Multiple Targets	✓	×	×
Compiler Graphical Interface	✓	✓	×
Externals Support (without re-implementation)	×	×	✓

But, despite these advantages, *empd* imposes its own technical barriers. The successful deployment of a system requires compiling Pd, establishing build processes for external objects when personalized externals are required, configuring these builds, designing a graphical interface for the *patch*, and integrating all components into a coherent structure. Such tasks remain highly complex for individuals without deep programming expertise in C/C++ and Web (JavaScript, HTML, and CSS). In this sense, the purpose of `pd4web` is to automate these tasks by building the configuration, providing a good and powerful GUI interface, and, in general, create a cross-platform App from the patch[1]. In basic terms, `pd4web` differs from *PdWebParty*, *PdXR*, and *WebPdL2Ork* in that it is designed as a JavaScript library with straightforward interoperability with other libraries (see section 3). It provides mechanisms for incorporating a wide range of external objects (see section 4.1), supports compatibility with GUI externals developed using *pdlua* (see section 4.2), and offers a simple GUI interface and automatic configuration for the compilation process via a Pd patch.

2. Context and motivation

`pd4web` was developed following the premieres of two pieces, *Eco* and *Moteto* (Curitiba-PR, 2023), both of which involved real-time feedback processes [7]. *Eco*, in particular, revealed significant challenges with Pd during rehearsals and the performance. We encountered issues with external libraries, bugs in externals, and fatal errors that

[1]Note that *pd4web* is designed to be used after the development of the *patch*, to use Pd on Web browsers, there are options like Purr-Data available in `https://charlesneimog.github.io/purr-data/`.

caused Pd to crash. These problems ultimately made the concert unfeasible and illustrate the risks of using Pd with externals created by composers rather than professional programmers.

Moteto was composed as a direct response to the challenges encountered in *Eco*, adopting a radical approach: instead of relying on laptops (with multiple instances of Pd), servers, or other external technical infrastructure, all audio processing was implemented in JavaScript and C (compiled to WebAssembly), leveraging WebAudio technology available in smartphone browsers. The success of this approach was particularly evident in performance contexts: *Moteto* was initially premiered by music students and later performed by an independent ensemble, consisting largely of amateur musicians with no technological background. This ease of execution highlighted the potential of WebAudio as a viable solution to many of the technical barriers associated with live-electronics performance. However, while this approach proved effective, it also underscored a significant limitation: expecting composers to develop a new project from scratch for each piece – including designing a build system and manually implementing fundamental DSP techniques such as FFT and partial tracking in C, as was done in *Moteto* – is neither practical nor sustainable.

Addressing this paradox – balancing development time with ease of use for performers – we proposed `pd4web`. By streamlining the deployment of live electronics through a browser-based solution, we believe that `pd4web` enhances accessibility for both performers and composers, reinforcing WebAudio as a viable and practical medium for contemporary musical creation.

3. Who Is pd4web For?

In the development of `pd4web`, the platform has been designed to accommodate three distinct user profiles: (1) performers, musicians and non-musicians without prior experience in Pure Data (Pd) or programming, (2) composers with some level of proficiency in Pd, and (3) artist-composers with expertise in both Pd, lua and web (JavaScript) programming.

For the first group, `pd4web` primarily targets usability through Web and Progressive Web Applications (PWAs). These users can engage with artistic projects without requiring any prior configuration or technical knowledge. The focus here is on simplicity and providing a traditional website-like experience. All underlying Pd-related processes are abstracted away, allowing users to concentrate exclusively on the musical or performative aspects of the artistic work. PWAs offer a notable advantage by supporting cross-platform deployment without reliance on proprietary distribution channels such as the Apple App Store or Google Play Store. Once deployed via `pd4web`, a Pd patch can function as a standalone application that is installable directly from the browser. Importantly, these applications also have offline capabilities, enabling consistent performance even in environments lacking internet connectivity—an essential requirement for live performances once installed.

For the second group—composers with Pd experience — `pd4web` offers native Pd integration, including support for most external objects. Its main benefit for these users is simplifying the compilation process, removing the need for manual configuration of compilers, build systems, or toolchains. A dedicated Pd object, `pd4web.compiler` (shown in Figure 5), automates these configurations by generating the corresponding web applications. `pd4web` also provides *templates*, which are pre-made websites for pieces or tutorial patches, identified by a single ID number. These templates broaden `pd4web`'s usability without requiring any programming skills. Currently, templates are available for choir pieces, live-electronics setups (score displayed alongside the Pd patch), *p5.js* and *ml5* integrations, tutorial patches, check `https://charlesneimog.github.io/pd4web/patch/templates/` for details.

For the third user group — composers and artists proficient in both Pd and web development — `pd4web` offers a JavaScript/TypeScript API. This API enables the bidirectional exchange of data between Pd patches and web interfaces, allowing developers to both send and receive information in real-time, thereby facilitating the creation of dynamic and interactive web-based artistic systems. We also made examples of these possibilities using *p5js* for a project poem, *ml5* for gesture recognition, and other tools. In this case, `pd4web` is loaded by using the *pd4web.js* file, after that, an instance of `pd4web` can be created using the followed code:

```javascript
var Pd4Web = null;
Pd4WebModule().then((Module) => {
  Pd4Web = new Module.Pd4Web();

  Pd4Web.openPatch("index.pd", {
    canvasId: "Pd4WebCanvas",
    soundToggleId: "Pd4WebAudioSwitch",
    patchZoom: 2,
    projectName: "MyProject",
    channelCountIn: 1,
    channelCountOut: 2,
    sampleRate: 48000,
    renderGui: true,
    requestMidi: false,
    fps: 0, // browser chooses the frame rate
  });
});
```

It is noteworthy that `pd4web` executes within a Web Audio Worklet context and relies on an internal virtual file system, which is instantiated at compile time. Consequently, any resources intended for use within `pd4web` — including Pd patches, audio files, and textual data — must either be incorporated at compile time or dynamically provided to the Web Audio Worklet via the `sendFile` method. For example, the following JavaScript snippet demonstrates how to handle a user-selected file and transmit its contents to `pd4web`:

```javascript
document.getElementById("someAudioInput").
    addEventListener("change", async (e) => {
```

```
    const file = e.target.files[0];
    if (!file) return;
    const arrayBuffer = await file.arrayBuffer();
    Pd4Web.sendFile(arrayBuffer, file.name);
});
```

In this context, the main patch compiled for `pd4web` is automatically saved as `index.pd`, serving as the entry point for the application. Any additional patches, sounds, or anything else should be placed in the `Libs/`, `Audios/`, or `Extras/` directory, from where they are automatically incorporated into the `pd4web` virtual file system.

Once the environment is completely initialized, it is possible to receive data from Pd to the Web (Pd $\rightarrow$ Web) using callbacks, with parameters transmitted through Pd objects such as `[s r-test]`.

```
Pd4Web.onBangReceived("r-test", (r) => {
    console.log("Received bang from ", r);
});
Pd4Web.onFloatReceived("r-test", (r, f) => {
    console.log("Received float: ", f);
});
Pd4Web.onSymbolReceived("r-test", (r, s) => {
    console.log("Received symbol: ", s);
});
Pd4Web.onListReceived("r-test", (r, l) => {
    console.log("Received list: ", l);
});
```

To send data to Pd patch (Web $\rightarrow$ Pd), you use the functions below, and you receive the data using the object `[r s-test]`.

```
Pd4Web.sendBang("s-test");
Pd4Web.sendFloat("s-test", 42);
Pd4Web.sendSymbol("s-test", "test");
Pd4Web.sendList("s-test", [1, "hi", "message"]);
```

These data exchanges can be used to create pieces that incorporate information from GPS, smartphone orientation, illuminance, and other sensors.

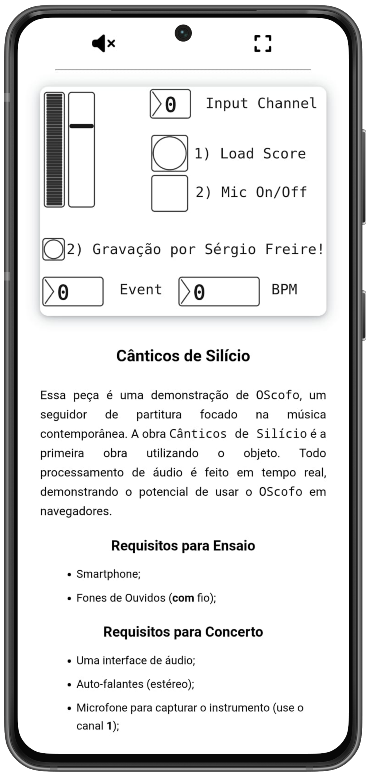
4. Key Achievements

4.1. External Objects Support

Environments like CSound, Max, and Pure Data can extend their native functionality via dynamically loaded libraries, known in Pure Data as *external objects*. A key goal in `pd4web` development was enabling the use of these objects without compromising system stability or restricting workflows for non-programmers. To achieve this, `pd4web` automates the analysis and compilation of external objects directly within the Pure Data environment as will be explained in the next section.

For example, `pd-saf`[2] is a wrapper for Pure Data based on the *Spatial Audio Framework* (SAF) [8], enabling advanced spatial audio rendering in web environments. Updates to the Web Audio API allow full access to the user's hardware channels, supporting high-channel-count spatialization in the browser.

Another example is `o.scofo~`, a Score Follower algorithm based on IRCAM research [9]. It synchronizes adaptively with live performances and can automate parameters like lighting cues, sample triggering, and virtual orchestration. Figure 2 demonstrates its use in `pd4web`, making this tool accessible via websites and smartphones.



Cânticos de Silício

Essa peça é uma demonstração de OScofo, um seguidor de partitura focado na música contemporânea. A obra Cânticos de Silício é a primeira obra utilizando o objeto. Todo processamento de áudio é feito em tempo real, demonstrando o potencial de usar o OScofo em navegadores.

Requisitos para Ensaio

- Smartphone;
- Fones de Ouvidos (**com** fio);

Requisitos para Concerto

- Uma interface de áudio;
- Auto-falantes (estéreo);
- Microfone para capturar o instrumento (use o canal **1**);

Figure 2: Demonstration of *Cânticos de Silício I* using `o.scofo~`.

Additional libraries that have been implemented include `else`, `cyclone`, `pmpd`, `partialtrack`, `fftease`, and `timbreIDLib`, among others. It is noteworthy that most libraries utilize `pd.cmake`[3] can be easily added to `pd4web`.

4.2. `pd-lua`: Graphical Library

In 2023, Timothy Schoen extended the external *pd-lua*, as documented in [10], by integrating a graphical API that internally translates *Lua* calls into the appropriate formats required by various Pure Data (Pd) variants, specifically Pd with *Tcl*, Purr-Data with *Node.js*, and PlugData with *JUCE/OpenGL/NanoVG*. This enhancement enables developers to construct graphical user interfaces (GUIs) for their patches that operate consistently across these Pd variants, as well as within Web-based patches via `pd4web`.[4]

[2]See `https://github.com/charlesneimog/pd-saf`.

[3]See `https://github.com/pure-data/pd.cmake`.

[4]For an example of the capabilities check `https://github.com/ben-wes/pdlua-show_tilde`.

On the backend, *NanoVG* in conjunction with `WebGL` is employed to render the *Lua*-based GUI elements within a web browser, following an approach analogous to that utilized in `PlugData`. In our implementation, *NanoVG* interfaces with *WebGL* to render the GUI elements. Examples of GUI *Lua* objects are presented in Figure 3. These graphical elements have access to mouse and keyboard events (currently implemented only in the `pd4web` version of `pd-lua`), facilitating the development of sophisticated and intuitive interfaces for web applications.

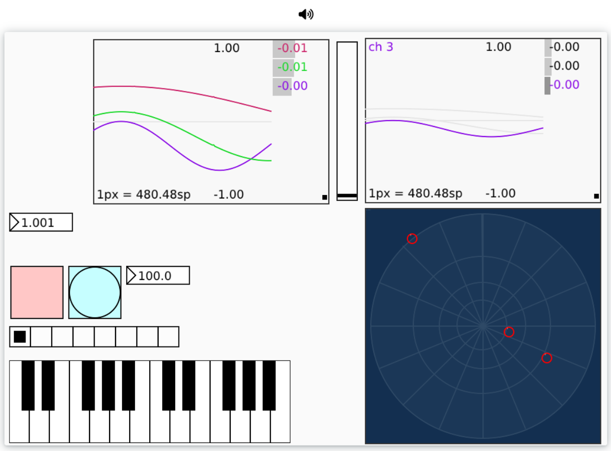


Figure 3: Example of Graphics drawn with `pd-lua`.

4.3. Electroacoustic music preservation

The long-term preservation of both software and compositions is recognized as a critical concern within the electroacoustic music community. This issue has been explicitly addressed by several composers and researchers, as discussed in [11], [12], [13], [14], among others.

Following the preservation framework proposed by [14], which distinguishes *bit-level*, *logical-level*, and *conceptual-level* preservation, we argue that `pd4web` primarily contributes to the *logical-level* of preservation for mixed-media pieces. By organizing all code, dependencies, and function calls of a piece in a single repository, `pd4web` ensures that the structure and functionality of the work are maintained with minimal dependencies possible.

Even in the extreme case where compilation in C becomes impossible, it remains feasible to reconstruct the entire sequence of function calls that generate the piece, including the digital signal processing functions employed. This allows for a faithful recreation of the work from its underlying computational structure. While `pd4web` does not tackle *bit-level* preservation of files or storage media, its capability to consolidate and provide access to the full codebase of a piece marks a significant advance toward sustainable, reproducible preservation.

4.4. Technology Accessibility

As discussed in Section 3, `pd4web` seeks to maximize accessibility along two dimensions particularly relevant in Brazil: technical accessibility for non-programmers and financial accessibility for musicians with limited resources. Technical accessibility is facilitated through a graphical interface and an automated compilation process, providing users without programming expertise a straightforward way to engage creatively with the software. Financial accessibility addresses the high cost of computing devices in Brazil, which constitutes a significant barrier. However, according to the Brazilian Institute of Geography and Statistics (IBGE)[15], nearly 90% of the population owns a smartphone. `pd4web` leverages these widely available devices, enabling students and emerging musicians to experience and experiment with electroacoustic music. Despite smartphone limitations—such as lower-quality microphones—the platform still supports meaningful aesthetic and creative experiences.

5. Technical Overview

`pd4web` is a C++ library designed to analyze the syntax of a Pd patch. This analysis extracts key information about the *patch*, including the external libraries it requires, the necessary git repositories to be cloned, and other dependencies. Since the Pd community is predominantly open-source, these repositories can be seamlessly integrated into an artist's project during compilation. Once the analysis is complete, `pd4web` compiles the code along with the required Pd and library files using Emscripten (as illustrated in Figure 4).

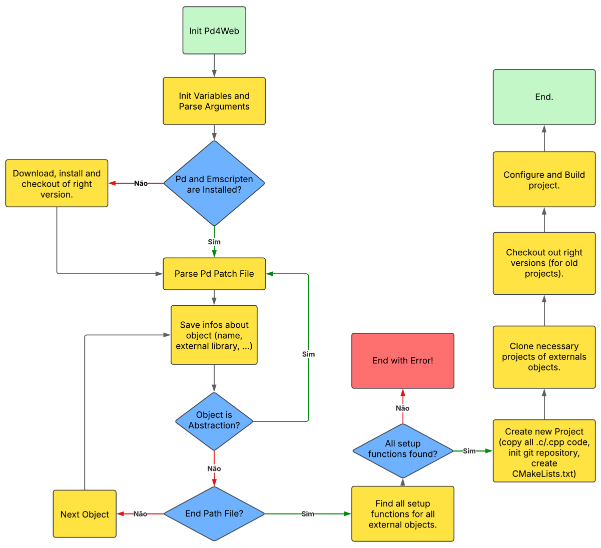

Init Pd4Web
Init Variables and Parse Arguments
Pd and Emscripten are Installed?
Não
Download, install and checkout of right version.
Sim
Parse Pd Patch File
Save infos about object (name, external library, ...)
Object is Abstraction?
Sim
Não
End Path File?
Não
Next Object
Sim
Find all setup functions for all external objects.
All setup functions found?
Não
End with Error!
Sim
Create new Project (copy all .c/.cpp code, init git repository, create CMakeLists.txt)
Clone necessary projects of externals objects.
Checkout out right versions (for old projects).
Configure and Build project.
End.

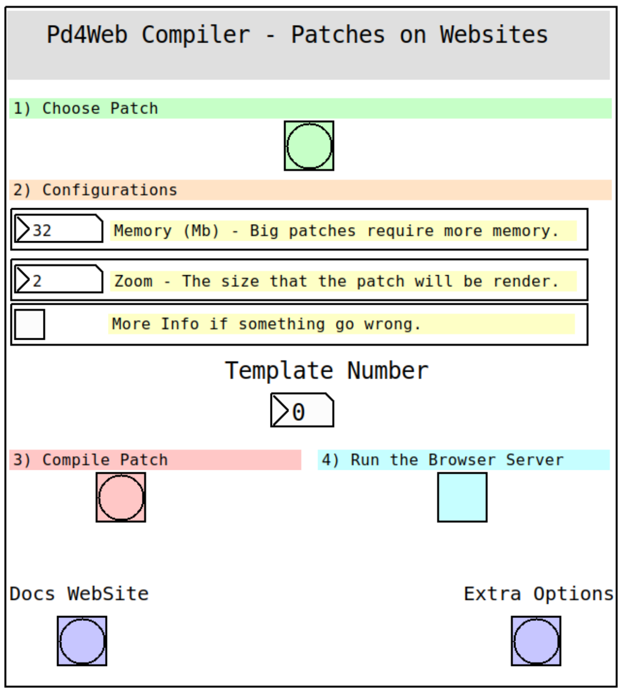
Figure 4: Demonstrations of the `pd4web` compiler patch.

Different from the implementation of Pd, in `pd4web` all external objects are configured, compiled, and loaded as internal objects. In other words, all the objects in `pd4web` behave similarly to native Pd objects. We chose this implementation primarily to avoid the complexity of supporting runtime dynamic linking via *.wasm* binaries using functions like `dlopen` and `dlsys`, which under Emscripten require careful thread synchronization and management of function pointer tables, especially when pthreads are involved[5]

As a solution, based on the patch analysis, pd4web can identify the external objects required by a project, compile them offline, and include them as static libraries in the final binary. This effectively converts all necessary external objects into internal Pd objects, including only those actually used. The approach reduces program size by compiling strictly what is needed, eliminates the need for runtime downloads, simplifies object loading in the browser, and we believe that it can improve long-term maintainability (see Section 4.3). Additionally, it enables deployment on free hosting platforms like GitHub Pages, where file size limits (e.g., 100MB) would otherwise restrict uploading entire libraries.

To enable this approach, pd4web identifies the object setup functions and generates a C++ file (*externals.cpp*) that calls these setup functions, ensuring that the required objects are available when the patch is loaded. Due to this internalization, minor modifications to the patch are also necessary—for example, removing path prefixes, which are handled automatically.

Finally, all these steps are done with the front-end interface that allows the compilation of patches within a Pd environment, a patch that compiles patches, and a Python module. This approach ensures that users of Pd can work within a familiar graphical interface without interacting with complex build processes or command lines. The front-end interface is illustrated in Figure 5.


Pd4Web Compiler - Patches on Websites

1) Choose Patch

2) Configurations

32 Memory (Mb) - Big patches require more memory.

2 Zoom - The size that the patch will be render.

More Info if something go wrong.

Template Number

0

3) Compile Patch

4) Run the Browser Server

Docs WebSite

Extra Options

Figure 5: Demonstrations of the pd4web compiler patch.

[5]More details in https://emscripten.org/docs/compiling/Dynamic-Linking.html#pthreads-support.

6. Technical Constraints of WebAudio for live electronics

Users of pd4web may encounter limitations due to the inherent constraints of the WebAudio API, with latency on devices representing the most significant challenge. Modern browsers attempt to select an optimal block size based on system capabilities; because of that, in practice, latency can theoretically range from 128 samples (2.66 ms) to 2048 samples (42.6 ms) or more, with precise control proving particularly difficult, especially on smartphones. On desktop systems, at least in Firefox, block size can be adjusted[6], yet this remains far from the goal of simplifying Pd patch deployment for non-programmers or users without advanced technical experience.

Another limitation is performance. The comparison between native C/C++ code and code compiled to WebAssembly is a controversial topic. While some microbenchmarks suggest that WebAssembly can approach the performance of native code, comprehensive evaluations have consistently demonstrated that WebAssembly tends to be slower than native C due to additional abstraction layers, sandboxing overhead, and runtime safety checks. While these layers improve security and portability, they also add CPU overhead that becomes especially noticeable in demanding patches. As a result, some patches that perform well on the artist's local machine may encounter issues in the browser, leading to sound artifacts caused by insufficient performance.

To improve performance, we use the switch~ object in Pd to selectively disable audio processing in parts of a patch. This is especially useful for computationally intensive operations, such as FFT analysis or spectral processing, which do not need to run continuously. By enabling these processes only when needed, we reduce computational load and ensure smoother execution in the WebAssembly environment.

7. Use Cases and Examples

The first piece to utilize pd4web is *Improviso I*, an artistic project centered around improvisation with a poem projection on stage. Performers interact with the algorithm by playing specific notes indicated in the projection, while the web-based patch responds by applying sound granulation to a pre-recorded poem or freezing/granulating a small buffer of the ongoing performance. The piece requires only a computer and a projector, and despite its real-time audio processing, the built-in microphone of the computer is sufficient for performance. Additionally, for rehearsals, performers can use a home TV as a substitute for the projector, eliminating the need for an audio interface, microphones, or any other technical audio equipment. The website for the piece is available at https://charlesneimog.github.io/Improviso-I/ under *Poem Algorithm*.

[6]See https://github.com/charlesneimog/web-audio-latency for a tutorial.

Cânticos de Silício (2024-5) is another piece that utilizes `pd4web`. Written for Soprano Saxophone, it showcases the potential of `pd4web` in two key areas: 1) Its ease of configuration, allowing the piece to be performed by a saxophonist with a wired headphone (due to latency) and a smartphone, without the need for an audio interface, computers, microphones, or any other technical audio equipment. 2) The integration of a score follower object called `o.scofo~`, which, despite its complex mathematics and algorithms, can be seamlessly incorporated into the `pd4web` environment. This enables the performer to read a traditional score while the algorithm tracks the player's progress on the score and triggers the electronic processing accordingly with note events detected. *Cânticos de Silício I* is available in `https://charlesneimog.github.io/Canticos-de-Silicio-I/`.

Artists and researchers are starting to use `pd4web`, in our knowledge, projects like:

- *Gaiasenses Web* by Felipe Mammoli (`https://gaiasenses-web.vercel.app/`), an online application that generates real-time animations and soundscapes from climate and meteorological data.
- *cartoP5* by Jean-Yves Gratius (`https://jyg.github.io/carto_p5/webpatch/`), an interactive web-based work for sonic improvisation.
- Research from the *Núcleo Música Nova*, mainly the project *BiA*, a library for the study of musical acoustics and sound synthesis, and *Cartilha*, a library for teaching concepts of modular synthesis.

In conclusion, the adoption of `pd4web` demonstrates its versatility and accessibility for both artistic and research contexts. From stage improvisation and score-following compositions to data-driven soundscapes and educational tools, the platform enables complex real-time audio processing with minimal technical requirements. These examples highlight its potential to broaden creative possibilities while lowering technological barriers for performers, composers, and researchers alike.

8. Future Perspectives

We aim to expand the adoption of `pd4web`, not only as a platform for real-time audio applications but also as a demonstration tool to advocate for improvements in the WebAudio specification. Specifically, `pd4web` highlights the need for a standardized, secure method to request low-latency audio in browsers. Achieving this would enable a wider range of real-time musical applications, particularly on smartphones, where low-latency audio processing is currently limited or inconsistent.

In addition, we are exploring methods for enabling device-to-device communication within a local network without relying on external servers. This approach allows interactive compositions in which devices – such as smartphones – can communicate directly when connected to the same Wi-Fi network. By eliminating the need for centralized infrastructure, this model supports real-time data exchange among participants, fostering spontaneous, decentralized, and collaborative musical interactions.

9. Conclusion

We believe that the development of `pd4web` represents a significant step forward in the technical/financial accessibility and usability of live-electronics systems for performers, composers, and musicians in general. By leveraging the power of WebAudio and the versatility of Pd, `pd4web` eliminates many of the technical barriers traditionally associated with live-electronics performance, such as complex software installations and configuration of external libraries. This simplification allows musicians with no programming background to integrate real-time sound processing into their performances, promoting a more inclusive and efficient creative environment.

Despite its considerable advantages, `pd4web` still faces technical limitations, particularly regarding WebAudio's latency and performance. The system can be used for electroacoustic music rehearsals and concerts, including smartphones, although latency issues require attention in concert performances. Ongoing optimization efforts, along with potential future advances in browser-based audio technologies, are expected to improve the user experience and expand the applications of live electronics in contemporary concert music.

References

[1] Miller Puckette. Combining event and signal processing in the max graphical programming environment. *Computer Music Journal*, 15(3):68, 1991. Accessed: 2024-12-09.

[2] Miller Puckette. Pure data: Another integrated computer music environment. In *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, Tachikawa, Japan, 1996.

[3] Claude Heiland-Allen. empd. `https://mathr.co.uk/empd/`, n.d. Accessed: 2024-12-09.

[4] Zack Lee. Pdwebparty: Run pure data patches in a web browser and share them via link. `https://github.com/cuinjune/PdWebParty`, 2020. Accessed: 2024-12-09.

[5] Damian Dziwis. Pdxr - the evolution of pure data into the metaverse. In *Proceeding of the International Computer Music Conference (ICMC)*, pages 1–8.

[6] William Furgerson, Ivica Bukvic, Justin Kerobo, and Bradley Davis. WebPdL2Ork: Bringing the linux laptop orchestra to the browser. In *Proceedings of the 19th Linux Audio Conference*.

[7] Ricardo De Oliveira Thomasi. *Estudos Ecos e vislumbres de uma performance musical ecossistêmica: uma investigação sobre estratégias de controle de estruturas emergentes em ecossistemas audíveis*. Doutorado em processos de criação musical, Universidade de São Paulo, São Paulo, June 2022. Accessed: 2024-12-02.

[8] Leo McCormack and Archontis Politis. Sparta & compass: Real-time implementations of linear and parametric spatial audio reproduction and processing methods. In *Audio Engineering Society Conference: 2019 AES International Conference on Immersive and Interactive Audio*. Audio Engineering Society, 2019.

[9] Arshia Cont. A coupled duration-focused architecture for real-time music-to-score alignment. 32(6):974–987.

[10] Albert Gräf, Timothy Schoen, and Benjamin Wesch. Pd-lua signals and graphics. In *Proceedings of the 19th Linux Audio Conference*.

[11] Bruce W. Pennycook and Bruce W. Pennycook. Who will turn the knobs when i die. 13(3):199–208.

[12] Isabel Pires, Filipa Magalhães, and Andreia Nogueira. Preservation and technological obsolescence: Portuguese contemporary musical heritage in perspective. 47(4):355–364.

[13] Joel Chadabe. Preserving Performances of Electronic Music. 30(4):303–305.

[14] Guillaume Boutard. Is there a digital archivist in the room? The preservation of musique mixte.

[15] Instituto Brasileiro de Geografia e Estatística (IBGE). No brasil, 88,9% da população de 10 anos ou mais tinha celular em 2024. `https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-noticias/noticias/44032-no-brasil-88-9-da-populacao-de-10-anos-ou-mais-tinha-celular-em-2024`, 2025. Accessed: August 16, 2025.