Euterpe: A Web Framework for Interactive Music Systems

YONGYI ZANG * CHRISTODOULOS BENETATOS * AND ZHIYAO DUAN
(yongyi.zang@rochester.edu) (c.benetatos@rochester.edu) (zhiyao.duan@rochester.edu)

Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY, USA

DRAFT

We present Euterpe, a prototyping web framework designed to facilitate the deployment of interactive music systems on the web. Utilizing the web's natural cross-platform compatibility, Euterpe enables widespread accessibility to these systems, potentially maximizing their impact. One of our main goals is to reduce the burden on developers by providing support in handling the JavaScript aspects of implementation. While developers still need to write JavaScript for their core algorithms, Euterpe assumes the responsibilities of receiving both audio and MIDI real-time input streams, synchronizing them, and sending them to the core algorithm in a structured fashion. Additionally, we offer pre-built functionalities for input and output data visualization. To showcase the capabilities of Euterpe, we conduct case studies on the deployment of "BachDuet" and "JazzImprov", two neural network music improvisation algorithms that were previously inaccessible to the general public. Through these case studies, we gather valuable feedback from both end-users who interacted with BachDuet and the independent developer who created JazzImprov. Euterpe is open-sourced at `https://github.com/yongyizang/Euterpe`.

0 INTRODUCTION

The domain of "Interactive music systems", a term coined by Rowe [1], has a long history of research and development [2, 3]. Such systems can serve a wide range of roles, spanning from augmented computer instruments to the complete embodiment of autonomous musical agents [1]. These systems offer enhanced musical engagement, promoting active participation and creativity among users, enabling them to experience music in a more immersive way than through passive listening [4].

As such, the continued research and development of interactive music systems is very important. However, it is equally important to ensure that these systems are widely and easily accessible to end users. Unfortunately, this is often not the case, as research efforts focused on such systems often conclude with open-sourcing the core algorithms without fully developing a functional prototype or creating one that is easily accessible to users without requiring significant effort to install it [5]. A recent survey [2] on musical agents highlights the scarcity of accessible prototypes for end users. Out of the 78 surveyed agents, only five had playable prototype systems accessible through open-source code, while only one of them had a web demo that was easily accessible to the end users. This scarcity of accessible interactive music systems restricts the potential impact and wider adoption among the general public.

One possible solution to address this issue is to encourage the development of interactive music systems for the web environment. By focusing on web-based implementations, developers can leverage the inherent advantages of the web platform, such as its widespread accessibility and cross-platform compatibility. JavaScript is the primary programming language for web development and has gained significant traction in the realm of music applications since the introduction of the Web Audio API [6, 7].

However, transitioning from research code to a final product that is accessible to end users can be a challenging task [8]. This is particularly true for interactive music systems, as their concurrent nature often necessitates special run-time support, such as threading as well as complex scheduling and synchronization of multiple streams of information [9]. These systems also demand a wide range of user interface components to display the various input and output information associated with the musical interaction.

In this work, we introduce "Euterpe", a prototyping framework that aims to alleviate most of the challenges faced by developers in the implementation of interactive music systems on the web. We specifically focus on systems that accept musical input in the form of audio and

*Equal contribution.

MIDI, and produce output in audio and symbolic music data formats.

We aim to support two commonly used interaction paradigms within Euterpe: the “simultaneous” and “call & response” playing modes, depending on whether the two participants take turns or play at the same time. Additionally, we support the two most common types of symbolic music representation in regards to time granularity, namely “event-based” and “grid-based” [10], as well as frame-level processing for audio.

Euterpe takes on several crucial responsibilities in the context of interactive music systems. It handles the reception of real-time audio and MIDI input streams, ensuring their synchronization and structured transmission to the core algorithm. By assuming these tasks, Euterpe relieves developers from the need of managing these input sources, allowing them to focus on the core functionality of their systems.

Additionally, in order to promote the creation of user-friendly applications, we provide a list of visualization components called “widgets”. These widgets serve as pre-built tools that developers can utilize to display various types of information on the screen such as audio, MIDI, text, and numerical parameters. Euterpe also includes customizable settings and audio mixer windows that allow users to control various parameters of the interaction. These parameters include settings related to the core algorithm as well as the audio levels of all the instruments involved in the interaction.

Finally, to demonstrate Euterpe’s capabilities, we present two case studies. First, we present our prior work during the development of Euterpe named “BachDuet”, which is a neural-network-based musical agent for real-time human-AI simultaneous improvisation of duets in Baroque style in the MIDI format. Second, we invite an independent researcher in the field of interactive music systems, to use Euterpe to create and deploy a prototype of her music interaction algorithm “JazzImprov”.

The work of Euterpe development and the two case studies are guided by two research questions: 1) How can we design and build a prototyping framework with the flexibility to meet diverse developmental needs? 2) Can implementing such a framework to assist researchers in deploying their systems more effectively? We believe that we have positive answers to both questions, as detailed in the following sections: We first review related work (Sec. 1), then delve into the design goals and requirements of Euterpe (Sec. 2). Next, we present implementation details, including the high-level system architecture and individual components (Sec. 3). We then present the two case studies and the feedback we received from the external developer and the end users (Sec. 4). Finally, we explore potential improvements and future work (Sec. 5).

1 RELATED WORK

1.1 Real-Time Interactive Music Systems

In this section, we review several real-time interactive music systems. The design of these systems guided us in determining the types of musical interactions Euterpe aims to support.

“Voyager” [11] is an interactive music system that allows a musician to improvise simultaneously in “free time” with an artificial intelligence agent that responds to user input in real time. It supports both MIDI and audio inputs. “GenJam” [12] is a system that can improvise jazz music in real time in the call & response mode where the musician and the system take turns to play for a predefined number of measures. “BachDuet” [13] is a musical agent that is able to improvise duet counterpoints in real-time with a human player in Baroque style. The improvisation happens in a simultaneous fashion where both the human and the agent play a monophonic voice in the symbolic music format. User inputs are quantized to the sixteenth-note time grid, while at the same time the agent generates its voice aligned to the same grid.

“RL-Duet” [14] is musical agent with the same interaction characteristics as BachDuet, but with the difference that it only predicts new notes on some of the slots of the time grid, depending on the duration of the last note it generated. It is also an example of a musical agent that has never been deployed and tested in a real human-computer interaction scenario. “Piano Genie” [15, 16] is a neural-network-based musical instrument that allows users to play melodies on a full 88-key piano using a small 8-key keyboard. It supports polyphonic input and output and operates in free time. This is achieved by an event-based operation where every new MIDI event received from the user instantly triggers a response from the core algorithm. “AI Duet” [17] is a musical agent that engages in a free-time, monophonic, call & response interaction with the user by generating continuations of the user’s input on a MIDI keyboard.

Finally, “Continuator” [18] is capable of analyzing a musician’s MIDI input in real time and generating new musical MIDI phrases that continue the style and structure of the input in a call & response fashion. Continuator also has some interesting interaction characteristics. During the user’s “call,” the input stream of MIDI events is pushed to a buffer and sent to the core algorithm at variable time intervals asynchronously. When the agent responds, notes are generated step by step, while at the same time, the agent keeps receiving input from the user (if any).

1.2 Audio and Music Analysis in JavaScript

The audio and music analysis ecosystem in JavaScript has seen significant growth and development over the past decade, with the introduction of several libraries and tools. The Web Audio and Web MIDI APIs serve as low-level tools for directly processing audio and MIDI within web browsers. They play a pivotal role in driving the growing utilization of JavaScript in music applications, as well as in the field of the Internet of Musical Things [19].

Based on Web Audio APIs, `Tone.js` provide a simple and intuitive interface for generating and manipulating sounds. For manipulating symbolic music data `Tonal.js` and `Music21j` are music theory libraries with support for a variety of note representations, including MIDI. `Essentia.js` and `Meyda.js` are libraries that offer tools for audio analysis and processing, much like what `librosa` library does for Python. Finally, `Magenta.js` is a library that includes a collection of symbolic music generation models developed by Google's Magenta group, built to run entirely in the browser. All these libraries have significantly expanded the range of music-related applications that can be built entirely in the browser. Euterpe provides a platform for developing interactive music systems using the resources mentioned above.

1.3 Web Frameworks for Hosting Systems

There are several frameworks available that aim to facilitate the deployment and accessibility of machine learning systems on the web.

"Gradio" [20] is an open-source Python library that allows developers to build simple GUIs for their machine-learning models and deploy them on the web to share with others. It is designed for hosting those models on servers, allowing users to connect remotely, instead of downloading them to local machines.

"HuggingFace" [21] is a web framework for hosting neural-network-based models. The framework allows developers to deploy their models as web services that can be accessed via RESTful APIs, making it easy for others to use and integrate these models into their applications. HuggingFace can also automatically generate widgets based on the model type (e.g., text classification, translation) to let users interact with the models.

Compared to Euterpe, HuggingFace and Gradio require only knowledge of Python, which is the most commonly used programming language for prototyping and developing machine-learning models [22]. However, these frameworks are particularly suitable for deploying models that support more static interaction modes, such as classification or text-to-image tasks, which involve limited user input compared to a typical real-time music interaction task. Models in those frameworks are intended to be deployed on remote servers, making them unsuitable for simultaneous interaction due to speed and stability constraints of network communication. Additionally, the scaling requirements of server-based systems may impose financial burdens on researchers and developers as the demand and resource needs increase.

In contrast, systems based on Euterpe can be deployed as single page applications (SPA) that run entirely on the user's local browser environment, thereby reducing latency and allowing the deployment of real-time music systems with higher interactivity. This local deployment also allows for more efficient scaling, as the system can be easily accessed by multiple users without the need for additional server resources. Nevertheless, this functionality requires the developer to implement the system's core algorithm in JavaScript and also imposes memory and time complexity constraints for the algorithm.

1.4 Music Prototyping Languages

Numerous specialized programming languages for computer music have been developed over the years [6], specifically designed for audio synthesis and processing as well as for rapid prototyping of interactive music systems.

One such popular language is Max/MSP [23], which is commonly used for prototyping interactive music systems [24, 3]. Max/MSP is a visual programming language that allows users to visually connect modules and create real-time audio and MIDI processing systems. It provides out-of-the-box graphical components for visualizing data. However, creating easily accessible and shareable systems using Max/MSP can be challenging. While it is possible to export Max/MSP patches as executable files, the process can vary between platforms, making the resulting standalone applications platform-specific. Additionally, Max/MSP does not offer native support for Linux operating systems, further limiting its accessibility and portability across different platforms.

Another notable language in the field is Faust [25], which is specifically designed for executing audio signal computations. Faust is specialized and focused on audio processing tasks and operates at the audio sample level [9]. Faust programs can be compiled to various languages including WebAssembly, enabling them to run directly on web browsers. However, due to its focus on audio signal processing, handling higher-level musical structures is challenging [9].

2 DESIGN

Euterpe is designed as a starting point to facilitate quick prototyping of various types of interactive music systems. To achieve that, we first try to identify the aspects in which these systems can significantly differ as well as the common core elements that they share. Then we incorporate support for those features that could ensure a wide coverage of interactive music systems.

Textual or verbal interaction between two entities always happens in a call & response fashion; however, as discussed in Sec. 1.1, a musical interaction can happen in call & response (e.g., Continuator), simultaneous (e.g., BachDuet) or any combination of the two (e.g., Voyager). Regarding temporal granularity, musical interaction can happen based on a strict or a more fluid time grid. In other scenarios, such as "free time" (e.g., Voyager), there might not even be a time grid, with participants follow their intuition and take actions at completely irregular points in time. In regards to modalities, human-computer musical interaction may involve multiple input and output modalities, such as audio, visual or symbolic data. Additionally, depending on these aforementioned interaction scenarios, interactive music systems often use similar ways to visualize the interaction, such as waveform and spectrogram views for audio, or piano roll and score for symbolic music. Finally,

we observe the diverse deployment environments in which interactive music systems are made accessible. These systems may take the form of standalone applications compatible with various operating systems, mobile applications specifically designed for smartphones and tablets, or even specialized hardware devices. Furthermore, it is also important to recognize that end users have varying degrees of familiarity with different environments, and that certain systems may not be readily available in environments they are familiar with.

Therefore, we aim to achieve the following design objectives:

1. Support a variety of musical interaction modes (Sec. 3)
2. Support both audio and symbolic music modalities (Sec. 3.1)
3. Provide various options for information visualization (Sec. 3.6)
4. Achieve cross-platform compatibility and ease of access for end users (Sec. 3.8)

An overview of Euterpe's architecture is shown in Figure 1. The "Scheduler"(Sec 3.4) is Euterpe's brain, and it functions as an intermediary between the user's input and the music interaction core algorithm, the "Agent"(Sec. 3.2). The Scheduler is responsible for synchronizing all different audio and note events and triggering the corresponding visual and audio components in the GUI. Each of these components, is treated as an independent module, capable of operating concurrently with the others. This modular design is informed by the fact that interactive music systems are inherently concurrent [9], i.e., different components of the system may be excecuted out-of-order or in partial order. For communication between these elements, we design a music interaction communication protocol (MICP) (Sec. 3.3). This modular design also allows developers to focus on the core algorithm, while Euterpe undertakes the task of interpreting the user input, forming structured data to send to the algorithm, and managing the presentation of the user's input and the Agent's output.

3 IMPLEMENTATION

For the implementation of Euterpe, we utilize Vue.js, a JavaScript framework for building web applications. Vue.js provides reactivity and offers a component-based architecture, which suits well the modular and concurrent designing approach outlined in Sec. 2. Additionally, we make use of the WebAudio and WebMidi APIs, which provide robust support for handling audio and MIDI input/output within web browsers.

To further improve Euterpe's performance, we also utilize parallelism. While JavaScript itself does not natively support multiple threads, we leverage the use of the Web Workers API [26] to execute resource-intensive computations in separate processes. Specifically, we utilize a Web Worker for the Agent module, and two AudioWorklets (a specialized form of Web Worker) for recording from the microphone and audio playback. All the rest of the modules responsible for processing the user input and rendering the interface are running on the main thread. Communication between the main thread and the modules running on separate Web Workers is achieved through a combination of message passing (i.e., `postMessage` method) and the utilization of SharedArrayBuffers [27], which enable efficient data sharing for time-critical data such as audio samples.

3.1 User Input

Euterpe can be used for symbolic and audio-based music interaction algorithms by supporting both note event and audio inputs. For note event input, Euterpe implements an on-screen touch-enabled piano keyboard, and supports the computer keyboard as well as external MIDI devices (facilitated by the Web MIDI API). In terms of audio input, the system employs the Web Audio protocol to directly extract audio buffers from the user's audio input device.

3.2 Agent

Euterpe implements the Agent using the Web Worker API. A Web Worker is a JavaScript object that runs a given JavaScript source code as a separate process. Web Workers do not share memory with the main thread, and thus, any communication with the main thread happens through message passing. Web Workers allow us to run demanding tasks without blocking the main thread, ensuring a smooth experience for end users.

The agent in Euterpe follows a modular approach by organizing its logic into separate steps commonly found in interactive music algorithms. These steps are encapsulated as callback functions called "hooks". We provide a set of default hooks that cover various aspects of the Agent's functionality:

- The **`loadConfig`** hook is triggered when the configuration object is received from the main thread. The agent can keep a copy in its thread, set any necessary parameters, and do configuration-related computations.
- The **`loadAlgorithm`** hook is triggered when the main thread is ready to initialize the algorithm. Within this hook, the Agent can perform any algorithm-specific initialization tasks such as loading pre-trained models, setting up data structures, or configuring internal variables. After that it sends a success status message to the main thread to indicate that it is ready to proceed with the interaction.
- The **`processNoteEvent`** hook is triggered whenever the main thread receives a note event from the user (i.e., event-based operation).
- The **`processClockEvent`** hook is triggered at every clock tick, and receives the raw as well as the quantized note events since the last tick. It enables synchronization and music-time level processing (e.g., beat-level).
- The **`processAudioBuffer`** hook is triggered when a new audio buffer is ready for processing. The frequency

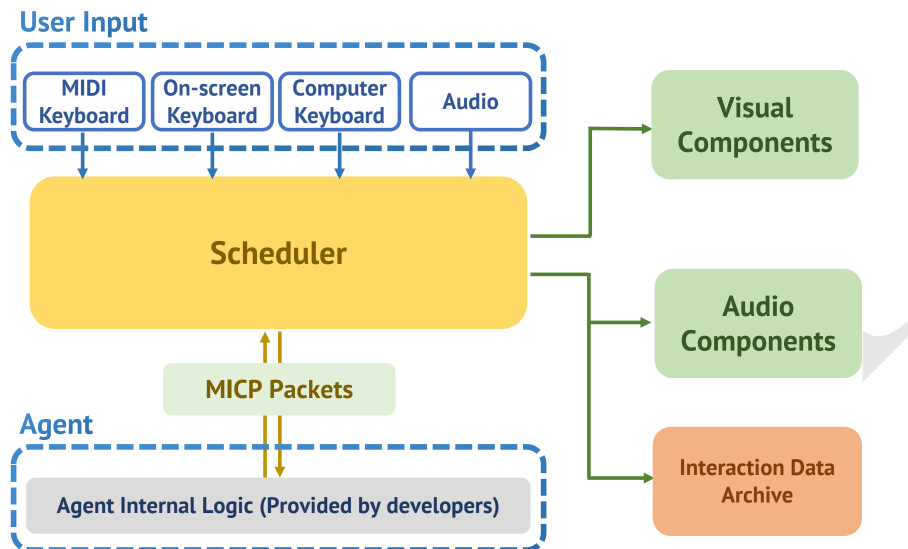


Fig. 1: Overview of the Euterpe architecture

at which this hook is triggered depends on the window and hop size settings chosen by the developer.

- The **processVariableUpdate** hook is triggered when the user interacts with the system's GUI and changes a UI element that corresponds to a variable or hyper-parameter of the Agent.

These hooks are automatically triggered from the main thread and offer developers the flexibility to implement various types of interaction logic within them. It is noted that developers can also write agent logic outside these hooks or even in external files that can be imported into the Agent's script.

3.3 Music Interaction Communication Protocol (MICP)

The modular design of Euterpe (Sec. 2) can benefit from designing a communication protocol between the Agent and the main thread. By formalizing the communication process, we aim to minimize errors by establishing a clear understanding of the expected inputs and outputs. This helps developers and researchers to easily plug in their music interaction algorithms to Euterpe. We refer to this protocol as the Music Interaction Communication Protocol (MICP), and we use the term "packet" to refer to an object that is transferred following the MICP.

A MICP packet is an object that contains two fields: the *hookType* and the *messages*. The *hookType* field corresponds to the hooks discussed earlier in the Agent section. When a packet is sent from the main thread to the Agent, it specifies which hook should process the packet. Conversely, when a packet is sent from the Agent to the main thread, it indicates which hook generated the packet.

The *messages* field contains a list of messages that convey various types of information from and to the Agent. The reason for sending multiple messages within the same packet (whenever possible) is to reduce the communication overhead between the Agent and the main thread (see also Sec. 3.4.2). Each message within a MICP packet consists of two fields: *messageType* and *content*. The *messageType* field indicates the type of the message and helps determine how the content should be interpreted. In our implementation, we have defined several message types, including:

- **status**: It is used by the Agent to convey status information to the main thread, based on which further actions can be triggered. For instance, when the Agent's *loadAlgorithm* hook successfully loads the music interaction algorithm, it sends a "success" message to indicate that the initialization process is completed without errors. After receiving such message, the main thread may then start running the algorithm.
- **quantized_notes**: It is sent from the main thread to the Agent on regular time intervals defined by a time grid. It contains user note events quantized to the time grid.
- **note_list**: It is used by both the main thread and the Agent to transfer un-quantized note events.
- **vector**: It is used by the Agent to send results in the form of a 1-d vector. The values of this vector will be displayed using the vector widget (Sec. 3.6.6).
- **label**: It is used by the Agent to send short textual results. These will be displayed in the label widget (Sec. 3.6.6).

More details on how the main thread utilizes the `quantized_notes` and `note_list` message types is illustrated in Figure 2 and explained in details in Sec. 3.4

3.3.1 *NoteEvent Object*

As mentioned earlier, when the message is of type `note_list` or `quantized_notes`, it contains a list of note event objects. These note event objects are designed as a generalization of MIDI note events, with additional fields that are specifically useful for Euterpe's internal operations:

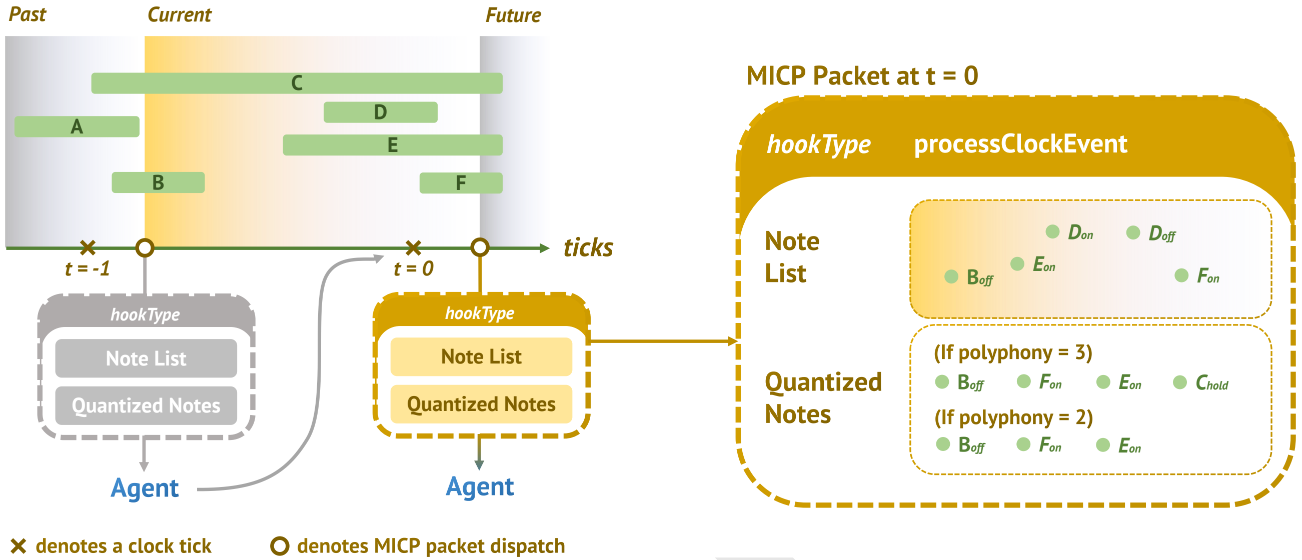


Fig. 2: An example of how the Scheduler processes the user's input, on grid-based mode.

- **player**: the player associated with the note event (e.g., Human or agent)
- **instrument**: the sampler instrument to be used to play this note
- **eventSource**: the source of the note event (Sec. 3.1)
- **name**: the name of the note event, if available (e.g., "C4")
- **type**: the articulation of the note event which can be *On*, *Off* or *Hold*
- **midi**: the MIDI value of the note event
- **chroma**: the chroma value of the note event
- **channel**: the MIDI channel of the note event
- **velocity**: the velocity value of the note event
- **createdAt**: the creation timestamp of the note event
- **playAfter**: the timestamp indicating when the note event should be played after
- **duration**: the duration value of the note event, used only by the Agent, for notes that their duration is known

It is important to note that all the time-related fields (`createdAt`, `playAfter`, and `duration`) are not scalar values but objects, describing time using a combination of ticks (quantized time) and seconds (continuous time). For instance, a note event generated by the Agent, with `playAfter` = {`tick`: 2, `seconds`: 0.3} will be sent to the main thread and scheduled to play after 2 clock ticks and 0.3 seconds with ticks being addressed initially, followed by the seconds delay). This combination of tick and physical timings enables sophisticated note scheduling, potentially allowing support for a grid-based agent that is able to generate notes with micro-timing variations for a more expressive performance.

3.4 Scheduler

As previously mentioned, the Scheduler functions as the "brain" of Euterpe. For note and audio inputs, the Scheduler implements two sets of logic as detailed below.

3.4.1 Note Scheduling

Grid-based. We implemented a "Clock" module that sends out "ticks" every clock period, which we use to support music interaction algorithms that need to sync their input and output operations to a time grid, such as BachDuet and RL-Duet. The developer determines the Clock's period in two ways: either explicitly, by setting the period of the Clock in seconds, or implicitly, by specifying the tempo in beats per minute (BPM) and the ticks per beat.

In grid-based systems that require the time grid to be adjustable, we provide a user-controllable tempo slider, which allows the user to change the Clock period on the fly. We also integrate an optional safe-keeping feature to alert the user when the Agent's processing time exceeds the Clock's period, as this would make it difficult to sync to the correct tick.

Another two critical tasks the Scheduler performs are: 1) the temporal quantization of the user's input on the time grid defined by the Clock and 2) the application of polyphony constraints on the user's quantized input. These polyphony constraints are essential in cases where interactive music systems operate under specific polyphony requirements, such as BachDuet which expects a monophonic input by the user. For instance, if the system is designed for 2-voice polyphony and the user activates 3 notes, the Scheduler removes the note with the oldest onset to adhere to the predefined constraints. In Fig. 2, we provide a diagram illustrating these two tasks related to the user's input stream. The horizontal time-tick axis shows tick positions (denoted as $\times$ label); the green bars represent the user's MIDI input stream. At every tick, the MICP packet sent to the Agent has `hookType` equal to `processClockEvent` and contains two messages. The first is of type `note_list` and includes all the new user note events that happened since the last tick. The second is of type `quantized_notes` and contains a list of the quantized and polyphony-constrained user note events for the current tick.

Additionally, in Fig. 2, the reader will notice a slight time delay between the tick (× label) and the dispatch of the MICP packet to the Agent (○ label); this is to make Euterpe robust to slight involuntary misalignments between user input and the metronome. For strict grid-based interactions, where the user has to align their input to the tick of a metronome, even the slightest misalignment with the metronome could cause the event to be registered on the next tick than the one the user intended. For example, in our figure, the user's intention was to play the notes *A*, *B*, and *C* simultaneously as a chord on tick -1; however we can see that the *B* note's onset happened after the tick. This small time delay between the tick onset and the MICP dispatch ensures that the *B* note will be included in the same group along *A* and *C*.

The developer can set the amount of delay; however, they should be careful of providing enough time for the Agent to process the packet before the next tick. Notice that the `note_list` message does not contain the note *C* since its "on" and "off" events occurred before and after the current tick. However, it appears in the `quantized_notes` message as a "hold" event (when the polyphony is larger than 2). On the other hand, the note *D* is included in the `note_list` but not in the `quantized_notes`, since its duration was less than the Clock's period.

Event-based. These are systems that do not operate on a time grid. In this case the Scheduler's job is much easier. In event-based mode, whenever the Scheduler receives an event from the user's input stream, it instantly dispatches it to the Agent using a MICP packet with `hookType` equal to `processNoteEvent` and a message of type `note_list` which contains that single note event.

If needed, the Scheduler can operate on both grid-based and event-based operations simultaneously.

3.4.2 Audio Scheduling

Audio scheduling differs from note scheduling in some important aspects. As we described in Sec. 3.4.1, notes played by the user would first go to the Scheduler, undergo processing if necessary, and then be relayed to the Agent through a MICP packet (via the `postMessage` method). However, due to the finer time granularity of audio compared to note events, as well as the overhead associated with `postMessage`, this method is not suitable for ensuring uninterrupted flow of audio samples to the Agent. To address this, we employ a more immediate means of communication between the Scheduler and the Agent by utilizing SharedAudioBuffers. These buffers allow for instantaneous sharing of data among different processes, ensuring that changes made by one process are immediately visible to others with access to the shared buffer.

More specifically, we use the `ringbuf.js` library that allows us to create a ring buffer based on a SharedAudioBuffer object. This ring buffer operates on a single-consumer single-producer model, allowing for efficient sharing of audio data between two processes. The maximum delay introduced by this buffer is equal to the size of the underlying SharedAudioBuffer, which can be configured by the user.

The initialization of the ring buffer takes place in the main thread, and it is shared with both the Audio Recorder and the Agent during the initialization stage. Once initialized, the main thread (Scheduler) no longer interacts with the ring buffer. The Audio Recorder assumes the role of the "producer", responsible for pushing audio samples into the ring buffer, while the Agent serves as the "consumer", responsible for reading the samples at regular intervals. To ensure smooth operation, it is essential that the Agent reads the samples from the ring buffer at a rate equal to or faster than the AudioRecorder writes to it. This prevents buffer overflow and loss of audio samples. By maintaining this requirement, we can achieve seamless audio data transfer between the AudioRecorder and the Agent.

The agent also performs the task of creating audio frames based on a window size, *w*, and a hop size, *h*, specified by the developer. When audio samples are retrieved from the ring buffer, they are added to an internal FIFO queue with a size equal to the window-size. Every *h* audio samples, a new audio frame with length *w* is created from the current state of the queue. This audio frame is then passed to the `processAudioBuffer` hook, which allows for further processing of the audio frame.

A similar process is used for sending audio samples generated by the Agent to the Audio Player. In this case, a new ring buffer is shared between the Agent and the Audio Player, but with the Agent serving as the producer and Audio Player serving as the consumer.

3.5 Audio Components

3.5.1 Audio Recorder and Player

To handle audio input and output, we utilize AudioWorklets, which serve as the Audio Recorder and Audio Player components. The Audio Recorder captures audio input from the user's device and sends it to be further processed by the Agent. On the other hand, the Audio Player receives audio data from the Agent and plays it back through the output device, allowing the user to hear the generated music in real-time. Details on how the Audio Recorder and Audio Player worklets communicate with the main thread and the Agent can be found in Sec. 3.4.2.

3.5.2 Sampler Instruments

As the primary audio component for note events, we provide an interface for creating sampler instruments that use concatenative synthesis to generate audio for note sequences in real time. Each sampler instrument comprises audio samples corresponding to individual notes; when a note is triggered, its corresponding sample is played. For more dynamic performance, the velocity of a given note determines the playback volume for its corresponding sample. Euterpe's sampler instruments are implemented via the web audio framework `Tone.js` [28].

3.6 Visual Components

Euterpe offers a range of visual components or widgets, that are synchronized with the Scheduler to display real-time information about the interaction. These widgets are adaptive in nature, as their content and behavior directly depend on the entries specified in a configuration file by the developer (Sec. 3.7).

3.6.1 Score

Western modern staff notation, referred to as "score" here, is a widely recognized and utilized musical notation system, and has been integrated into Euterpe. This inclusion aims to provide a familiar and intuitive interface for musicians who are accustomed to reading and interpreting music in score notation. `VexFlow.js` [29], a library that provides an interface for a Scalable Vector Graphics (SVG) score, is used to implement the score notation. At each clock tick, the notes are drawn on the graph, and a scrolling animation is triggered to ensure that the currently displayed section of score notation corresponds to the current interaction. Due to the complexity of implementing a polyphonic score notation engraving algorithm, we currently only support monophonic voices for the grid-based operation mode.

3.6.2 Piano Roll

The rising popularity of music sequencing software, Digital Audio Workstations (DAWs), and musical games has led to widespread user familiarity with piano roll notation. In this notation system, each note is represented as a rectangle, with one dimension indicating the pitch and the other indicating the timing. Euterpe has also integrated this notation system by leveraging `Three.js` [30], a graphical library that facilitates the creation and display of 3D computer graphics. Similar to A.I. Duet [17], an orthographic camera perspective is utilized to produce a 2D visual effect. The piano roll is synchronized with note events, where user and agent outputs correspond to note rectangles with different colors. Contrary to the score notation, Piano roll supports polyphonic voices in both event-based and grid-based operation modes.

3.6.3 Settings Window

The Settings window in Euterpe offers end users the ability to adjust various interaction settings, such as the BPM (clock speed) and selection of an external MIDI device. It additionally includes interactive elements such as buttons, sliders, and switches. The amount and behavior of these elements is adaptive and can be easily customized by developers through the configuration file.

3.6.4 Mixer Widget

The Mixer widget in Euterpe is an audio mixer designed to control the volume and mute status of each instrument belonging to the players involved in the interaction. The content of the Mixer widget is automatically populated based on the players and instruments defined in the configuration file. This dynamic behavior ensures that the mixer interface accurately reflects the structure of the interaction and the audio elements involved.

3.6.5 Monitor Widget

The Monitor widget serves as a non-interactive tool designed to monitor real-time changing variables of the interaction. Its purpose is to provide developers and end users with insights into the system's behavior and assist in tracking and analyzing specific parameters. It can only be used to track floating-point values and it offers two ways to present the monitored data: a text format and a 1-d rolling graph. Developers can customize the widget by adjusting the configuration file to specify the variables they want to monitor, set the frequency of value updates, and choose between the text or graph display format.

3.6.6 Other Widgets

- ***Label Widget:*** This is a straightforward text box that is used to displaying text sent from the Agent to the main thread. It serves as a convenient tool for presenting small textual elements such as chord or key labels.
- ***Vector Widget:*** This is a bar plot designed to showcase the values of a one-dimensional array sent from the Agent. This array can represent various data, such as probability predictions for different classes, audio chroma vectors, and more. To configure the widget, the developer needs to specify the desired number of bins as well as optional labels for each bin in the configuration file.
- ***Spectrum Widget:*** A widget that provides a visual representation of the frequency content of the user's audio input.

3.7 Global Configuration

To allow developers to easily customize the system, we provide a unified interface for configuring interaction logic and color palette. For interaction logic, we supply a YAML-format configuration file that specifies settings such as tempo, interaction mode, window and hop size for audio and other interaction related parameters. The configuration file also allows developers to specify the application title, introduction text, and agent related hyperparameters. Additionally, the developers can specify the instrument types available to the end user and the Agent, as well as the behavior and types of information to be displayed by the GUI widgets (Sec. 3.6). Finally, we provide a Cascading Style Sheets (CSS) file that defines the color palette for the system. By modifying this CSS file, developers have the ability to easily customize and define the color scheme for all components in the application.

3.8 Deployment

Euterpe can be deployed as a SPA using Webpack, a module bundler. At build time, a Euterpe application's dependency graph is analyzed by Webpack, with all dependencies converted into a unified format. The resulting output bundle contains all the code and assets necessary to run the entire web application in the user's browser envi-

ronment, with a single webpage as the application's entry point. This process happens automatically with each build without the need for manual user intervention. Therefore, Euterpe can be deployed with readily available web page hosting sites, such as Heroku, Netlify, or GitHub Pages, or be deployed as a standalone offline application with frameworks like `Electron` or `NW.js`.

4 CASE STUDIES

4.1 BachDuet

To demonstrate the deployment process of a symbolic music based system with Euterpe, we choose BachDuet [13] as an example. BachDuet enables a human performer to improvise a real-time duet counterpoint with a computer agent in Baroque style. The concept of this interaction is illustrated in Fig. 3. The input to the system is a human musician's monophonic MIDI performance, while the output is the Agent's monophonic performance in real time generated by a Recurrent Neural Network (RNN).

As a neural-network based real-time interaction system, the deployment of BachDuet poses unique challenges and allows us to showcase many Euterpe features. We present both user data analysis from passively collected metadata from the published website, as well as a more focused subjective evaluation involving ten end-user participants. The final system is deployed at `https://bachduet.com`.

4.1.1 GUI

Since BachDuet is an interactive music system in the Baroque style, an easy-to-use interface for classically-trained musicians is needed. Therefore, besides the Piano Roll notation, we also enable the Score notation component.

We further customize the color by specifying the CSS variables, which automatically changes the entire application's color palette. Also, we injected custom CSS to customize the on-screen keyboard's appearance further. The final GUI is illustrated in Figure 4

4.1.2 Configuration

As we briefly introduced in Sec. 1.1, this is a MIDI interaction system that uses a sixteenth-note time grid, which means that we have a grid-based interaction mode that uses the Clock to create the time grid. In the `clockSettings` block, we set the Clock to tick four times per beat. The user's monophonic MIDI input stream is temporally quantized based on the sixteenth-note time grid, and the only time signature it supports is 4/4. Since the interaction is monophonic and grid-based, we can also enable the Score visualization. Based on the descriptions above, we provide part of the configuration file in Listing 1.

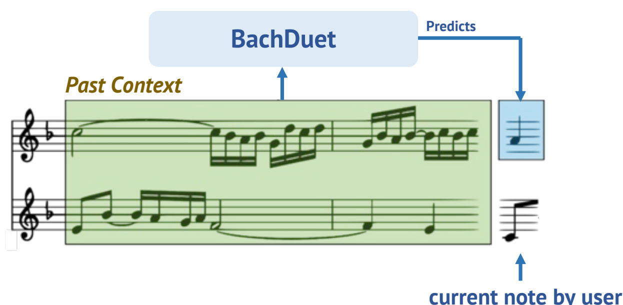


Fig. 3: The interaction concept of BachDuet

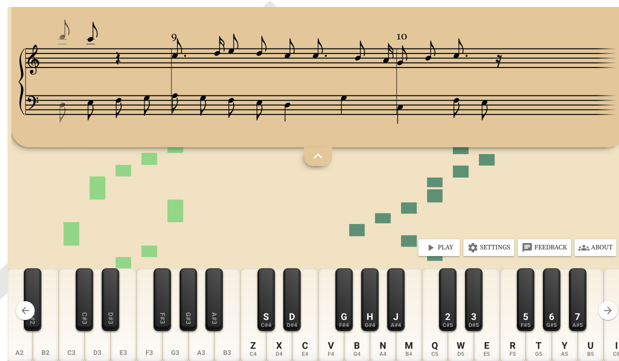


Fig. 4: Screenshot of BachDuet GUI

```
title: "BachDuet"
subtitle: "Baroque-style AI"
interactionMode:
    noteMode: true
    audioMode: false
noteModeSettings:
    eventBased: false
    gridBased: true
    polyphony:
        input: 1
        output: 1
clockSettings:
    ticksPerBeat: 4
    timeSignature:
        numerator: 4
        denominator: 4
    tempo: 90
    clockPeriod: null
agentSettings:
  warmupRounds: 2
  randomness: 0
gui:
    score: true
    pianoRoll: true
```

Listing 1: Configuration file for BachDuet

4.1.3 Agent

Before implementing the Agent logic, an important preliminary step is to convert the neural network model into a format compatible with JavaScript libraries that support neural network inference. For BachDuet, we choose `TensorFlow.js`. This step is necessary not only for the

deployment of BachDuet but also for any other neural-network based agent.

In the `loadConfig` hook, the Agent receives the configuration from the main thread and stores it in a local variable. BachDuet also relies on an external JSON dictionary loaded in this hook, which contains the mapping between MIDI numbers and neural network tokens.

In the `loadAlgorithm` hook, we initialize `TensorFlow.js`, then we load the neural network's weights and we run a few warm-up rounds. Finally, a MICP packet is sent to the main thread stating that it is ready for interaction.

In the `processClockEvent` hook, we take the user's latest MIDI input quantized to the current clock tick and the Agent's previous output, then run an inference step to generate the next note to be played by the Agent.

4.1.4 Deployment

To deploy the website, we use the web page hosting services provided by Netlify. We created a GitHub repository[1] that hosts the source code, then connects the push action to Netlify. Netlify runs the build command on each push, then sets the `Webpack` bundle as the website root directory, as mentioned in Sec. 3.8. This allows us to utilize the free webpage hosting service of Netlify, including a Secure Sockets Layer (SSL) certificate from "Let's Encrypt," allowing for HTTPS connections.

4.1.5 Feedback

BachDuet was initially launched in May 2022. As of its one-year anniversary, the site recorded a total of 794 musical interactions with its users. The breakdown of user interactions across different operating systems was as follows: Windows users accounted for 360 interactions, Macintosh users similarly accounted for 360, Linux users had 50 interactions, and tablet users (such as those using Chrome OS and iPadOS) had 24 interactions. An integral aspect of the web user experience is the loading time; based on the recorded data, 36.4% of users were able to load the entire webpage within 1 second, and 90.0% of users loaded the webpage within 3.5 seconds.

For a more in-depth understanding of the user experience, we solicited feedback from ten participants who had no previous interaction with BachDuet. The participants were asked to interact with the platform and subsequently complete a questionnaire, which contains the following questions:

1. *Which types of input methods did you use? (Participants can choose computer keyboard, MIDI keyboard or on-screen keyboard.)*
2. *Can you estimate the approximate duration of your interaction with the system? (Participants can choose <1 minute, 1-5 minutes, 5-10 minutes or 10-30 minutes.)*
3. *What device are you currently using to play with this system?*
4. *Do you find the GUI easy/intuitive to use? (scale 1-10)*
5. *Do you find the GUI to be simple and uncluttered? (scale 1-10)*
6. *Do you find the visualizations to be accurate compared to your actual input? (scale 1-10)*
7. *Does the GUI clearly reflect the notes and timeline of the real-time interaction between you and the computer? (scale 1-10)*
8. *Do you find the GUI allows you to quickly and easily perform tasks, like changing BPM, changing volume? (scale 1-10)*
9. *What's your rating of your interaction with BachDuet? (scale 1-10)*

For Questions 4 to 9, a higher value indicates higher agreement with the question. An additional open-ended question was also included at the end to encourage participants to provide further comments.

Regarding the first three questions, six participants reported using a computer keyboard for interactions, three used a combination of the on-screen and computer keyboards, and one used a MIDI keyboard. Three participants engaged for a span of 1-5 minutes, the majority of five participants interacted for a duration of 5-10 minutes, and the remaining two reported a longer interaction time of 10-30 minutes. All participants used a laptop, and specifically, four participants were Macintosh users with the remaining six using Windows.

Boxplots of scores for Questions 4 to 9 are shown in Figure 5. The data suggests a favorable perception of the user interface by participants, particularly in terms of intuitiveness (Q4), simplicity and clean design (Q5), accuracy on visualizations of user input (Q6), and clarity of GUI in reflecting the real-time interaction (Q7). Overall, participants were positive about their interaction with BachDuet (Q9). In open-ended responses, participants commented that the system was "very fluid and responsive", and they had "a very nice and smooth experience" and "really liked the general UI".

[1] https://github.com/yongyizang/BachDuet-WebGUI

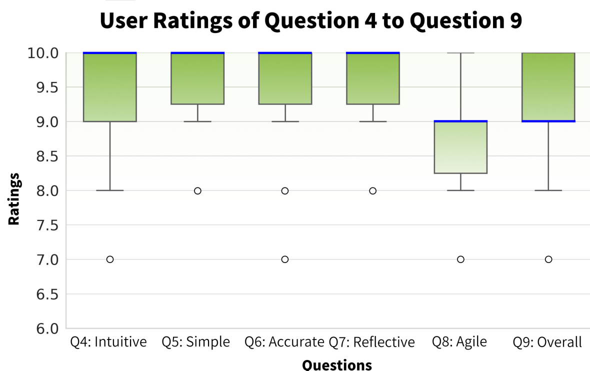


Fig. 5: Box plot of ratings of Questions 4 to 9 received from all ten participants. Higher values indicate better agreement with the questions. Median of each box is shown as the blue line. The top and bottom of each box represent the 25 and 75 percentiles, respectively. Outlines are shown as circles.

The survey responses also indicated a potential area for improvement in system controllability (Q8). From the open-ended responses, it is evident that users had specific feature requests, some of which were related to Euterpe as a whole, while others specifically pertained to the BachDuet system. Regarding Euterpe, users expressed a desire to have additional functionality, such as the ability to pause, rewind, and replay previously generated music. They also requested the flexibility to customize the mapping of the computer keyboard to MIDI notes. Furthermore, four users requested a direct export feature that would allow them to save the interaction data as a MIDI file.

4.2 JazzImprov

To better illustrate the real-world experience of deployment with Euterpe, we invited Yiyang Wang, an interactive music system researcher and musician, to develop and deploy her JazzImprov. Unlike BachDuet, JazzImprov accepts audio input and outputs MIDI events, making it an ideal test case for evaluating Euterpe's audio capabilities. The prototype of the system can be found under the deployed systems section in `https://euterpeframework.org`.

Details of the system's design are shown in Fig. 6. JazzImprov allows a human performer to improvise with a computer-generated backing track that emulates a Jazz trio band consisting of Piano, Double Bass, and Drums. The Piano voice plays chords generated by an RNN based on current human input and prior interaction history. The RNN first generates probabilities for each of the 12 notes, and uses rule-based logic to extract the most probable chord. The Drums, and Double Bass parts are generated based on the chord predictions using rule-based logic.

4.2.1 GUI

Given the audio-based nature of JazzImprov, Yiyang needed to provide visual representations of audio interactions. For this purpose, she selected the Spectrum Widget and added RMS and perceived loudness values to the Monitor Widget. She used the the Vector Widget to visualize her model's raw probability outputs (12-d vector). Additionally, the model's predicted chord is displayed using a Label Widget in real-time. Finally, to monitor the model's performance, Yiyang added her agent's inference time to the Monitor Widget as well. The final GUI is illustrated in Fig. 7.

4.2.2 Configuration

As we described earlier, JazzImprov supports both audio and MIDI modes so we enable the audio functionality using the `audioMode` flag and `audioModeSettings` block. Similarly to BachDuet, the system's music output is alignment to a 16th note grid; The chord predictions are generated once every beat, while the Drums and Double Bass tracks are generated on every 16th note. As for the GUI, all of the visual components are enabled 3.6, except for the score which is not functional when taking audio input. Part of the configuration file for JazzImprov is provided in Listing 2.

4.2.3 Agent

As another neural-network based system, JazzImprov shares similar `loadConfig` and `loadAlgorithm` hooks with BachDuet.

Inside the `processAudioBuffer` hook, Yiyang added the audio frame-level feature extraction. She used `Meyda.js` to convert the raw audio frames into a chroma vector representation. Each chroma vector is pushed to a queue that exists within the scope of the Agent. This queue serves as a storage mechanism for the chroma features extracted from the audio frames.

The `processClockEvent` hook is triggered by the main thread on every clock tick (16th note). Within this hook, JazzImprov performs the inference of the RNN model every four ticks, as the model operates on the beat and quarter note level. At each interval of four ticks, JazzImprov accesses the chroma queue, which is populated by the `processAudioBuffer` hook, runs the RNN inference using the chroma vectors, and then empties the queue to prepare for the next batch of chroma vectors. At the same time, within the `processClockEvent` hook, JazzImprove also runs an rule-based algorithm that generates drums, bass, and the piano rhythm, at every tick.

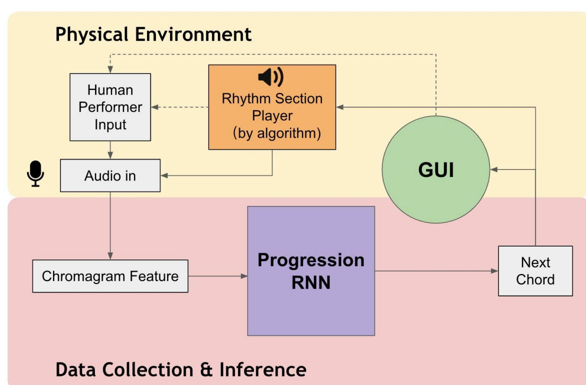


Fig. 6: Overview of JazzImprov interaction system

Fig. 7: Screenshot of JazzImprov GUI

```yaml
title: "JazzImprov"
subtitle: "An AI JazzTrio"
interactionMode:
    noteMode: true
    audioMode: true
noteModeSettings:
    eventBased: false
    gridBased: true
audioModeSettings:
    input: true
    output: false
    windowSize: 2048
    hopSize: 1024
clockSettings:
    ticksPerBeat: 4
    timeSignature:
        numerator: 4
        denominator: 4
    tempo: 90
    clockPeriod: null
agentSettings:
  warmupRounds: 2
gui:
    score: false
    pianoRoll: true
    vectorWidget: true
    labelWidget: true
    spectrumWidget: true
```


Listing 2: Configuration file for JazzImprov

4.2.4 Deployment

To deploy the website, Yiyang followed similar steps as BachDuet to deploy using Netlify. In the interest of simplicity, we skip the details. At the time of this paper's writing, the source code repository for Yiyang's web application is not open-sourced; we will update Euterpe's website with the repository link once it is publicly released.

4.2.5 Feedback

The opportunity to collect feedback from an external researcher who has thoroughly explored Euterpe and successfully deployed an interaction system with it is very valuable. To gather comprehensive comments, we designed a questionnaire containing open-ended questions for Yiyang to detail her experience. Her responses are as follows:

- Have you implemented your algorithm into a prototype in other languages/platforms? If yes, how long, roughly, did you spend in building the interface and application side of the prototype?
  `Yes. I spent roughly 60 hours in total (used Max and Python, proficient in both).`
- How "portable, shareable, and accessible" do you think your previous application was, on a scale of 1 to 10? Higher values are better.
  `4. Max/MSP supports multiple coding platforms; however, the application had OS-specific dependencies, making it not quite portable.`
- Do you have any background in terms of web programming? If yes, can you specify your background?
  `Yes. I had some experience with JavaScript (1000 lines of code in single projects), and some working knowledge of general website structure (html, css, etc.) and limited experience on Python web frameworks (Django, Flask). I haven't built a standalone website from frameworks previously.`
- How long did it take, roughly, for you to implement your application as a webpage using Euterpe?
  `I used a time-tracking tool to track the entire application development process; it took 15 hours in total, starting from the template provided by Euterpe. This did not include the time in exporting the neural network from PyTorch to TensorFlow.js which took around 5 hours.`
- How long do you think it may take, roughly, for you to implement your application as a webpage without using Euterpe?
  `More than 400 hours. It's going to be a headache to figure out the GUI and the system's overall structure.`
- How "portable, shareable, and accessible" do you think the newly built application is, on a scale of 1 to 10? Higher values are better.
  `9. The web-based nature of Euterpe extends a natural layer of portability and accessibility.`
- What do you think are the strongest strengths of Euterpe?
  `Euterpe abstracts away the ''dull'' process of web user interface design and the tricky data transport specifications.`
- What do you think are the most prominent weaknesses of Euterpe?
  `The UI components have only limited options to customize directly from the config files, so if the developer or researcher is aiming for something quite different, more intricate changes to the overall framework need to be made. Better documentation needs to be written for that purpose as well.`

This feedback highlights Euterpe's ability to facilitate the prototype deployment process. Euterpe achieves this through its abstraction of the system, which only exposes a limited number of agent hooks, thereby simplifying the development process. That being said, this simplification also limits the options for the researchers, as Yiyang pointed out

as an area for improvement. Taking into consideration her constructive critique, our ongoing objective is to expand our range of widgets to cater to the diverse needs of developers, while also extending the documentation to support those seeking further customization of Euterpe.

5 FUTURE WORK

Euterpe is an active project, and we are committed to enhancing its features to provide better support for a broader range of interactive music systems. We have outlined a roadmap of enhancements and features for Euterpe, ranked from short-term to long-term priorities:

- **Exporting audio and MIDI.** Even though the interaction data are stored internally in Euterpe, we do not provide the option to directly export the symbolic or audio interaction history in popular formats such as MIDI, PDF or WAV. This is a feature that many end users, as well as our independent developer, asked.
- **Performing assessment under diverse scenarios.** Web applications run on diverse browser platforms from a wide range of devices. Testing Euterpe across diverse platforms and devices will provide us with insights about its robustness under various real-world conditions.
- **Extending polyphony constraints.** Currently we are only applying polyphony constraints on the quantized notes during grid-based operation. Our next step is to apply the polyphony constraints directly at the user's un-quantized input events. Additionally, since currently we only support "last" priority (the last note played by the user has higher priority), We also want to implement other types of priorities, such as "highest" and "lowest".
- **Supporting server based agents.** Another area of future work for Euterpe is to enable interactions with agents that run on remote servers instead of the browser. While this may not be suitable for real-time simultaneous interactions with strict deadlines, it can be valuable for real-time interactions with more flexible timing requirements. Additionally, working with larger models, such as audio language models, may necessitate running agents on remote servers due to resource constraints.
- **Conditioning on external material.** Score-driven interactive music systems (as defined in [1]) may require external material, such as chord sequences or MIDI files. We plan to add support for importing external material to better adhere to those systems' needs.
- **Looping based interactions.** We also aim to support a looping-based music interaction paradigm that involves both the user and the Agent. In this interaction mode, users create and manipulate musical loops, while also enabling the Agent to contribute its own layers to the loop. Users will have the ability to record their own musical phrases or sounds, overlay them with the Agent's generated material, and explore new musical possibilities through iterative layering.
- **Polyphonic and free time score notation.** We plan to expand the capabilities of the score notation visualization. As mentioned in 3.6.1, The score notation is limited to displaying monophonic note sequences with a predefined time grid and a known time signature.
- **Visual input methods.** Besides musical input, there are interactive music systems that use visual input methods such as dance moves or hand gestures to interact with music [31]. Thus, we plan to introduce support for web cameras as input and transmit video information to the Agent.
- **Better mobile device support.** As of now, Euterpe is designed for desktop devices, not taking advantage of interaction methods available on touchscreen devices. To address this limitation, we plan to introduce new visual components specifically designed for mobile devices, such as a multi-touch on-screen keyboard.
- **Multi-player mode.** The current design of Euterpe assumes a two-player interaction system with one human player and one agent. We plan to introduce a multi-player interaction mode to provide broader support.

6 CONCLUSION

In this work, we presented Euterpe, a prototyping web framework, designed to simplify the deployment process of interactive music systems on the web. By leveraging the web's cross-platform compatibility, Euterpe enhances the accessibility and potential impact of these systems. Our focus was on reducing the development burden by handling JavaScript aspects such as real-time input streams and data synchronization, allowing developers to focus on their core algorithms. Additionally, Euterpe offers a series of widgets for data visualization, further helping developers in creating an engaging prototype.

To demonstrate the capabilities of Euterpe, we conducted case studies of the deployment of two neural network music improvisation systems.The first system was BachDuet, a MIDI-based improvisation system that has been accessed by hundreds of end users on the web. Subjective evaluation revealed a high level of approval and positive user experience regarding the interface of BachDuet. The second system was JazzImprov, an audio-based improvisation system developed by an independent researcher. The researcher reported that she was able to create a working prototype in one-third of the time it took to implement using her previous approach. This feedback suggests that Euterpe can be a valuable tool in accelerating the development process of interactive music systems.

Euterpe is under active development, and we plan on adding more features to support a wider range of interactive music systems as well as addressing the needs of researchers and end users.

7 ACKNOWLEDGMENT

This work has been partially funded by the National Science Foundation grants Nos. 1846184 and 2222129. We thank Yiyang Wang for her feedback, bug reports, and feature suggestions in the case study of the deployment of her JazzImprov system with Euterpe. We also thank Tianyu

Huang for his assistance in the early stages of the development.

8 REFERENCES

[1] R. Rowe, *Interactive music systems: machine listening and composing* (MIT press, 1992).

[2] K. Tatar and P. Pasquier, "Musical agents: A typology and state of the art towards Musical Metacreation," *Journal of New Music Research*, vol. 48, no. 1, pp. 56–105 (2019).

[3] T. Winkler, *Composing Interactive Music: Techniques and Ideas Using Max* (MIT press, 2001).

[4] R. Rowe, "The Aesthetics of Interactive Music Systems," *Contemporary Music Review*, vol. 18, no. 3, pp. 83–87 (1999).

[5] H. Flores Garcia, A. Aguilar, E. Manilow, D. Vedenko, and B. Pardo, "Deep Learning Tools for Audacity: Helping Researchers Expand the Artist's Toolkit," presented at the *5th Workshop on Machine Learning for Creativity and Design at NeurIPS 2021* (2021).

[6] L. Wyse and S. Subramanian, "The Viability of the Web Browser as a Computer Music Platform," *Computer Music Journal*, vol. 37, no. 4, pp. 10–23 (2013).

[7] S. Pfeiffer, "HTML5 Audio API," in *The Definitive Guide to HTML5 Video*, pp. 223–245 (Springer, 2011).

[8] J. Jacobs, "From Prototype to Product: Deployment strategies in computer science research," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 23, no. 1, pp. 5–6 (2016).

[9] R. B. Dannenberg, "Languages for Computer Music," *Frontiers in Digital Humanities*, vol. 5 (2018), doi:10.3389/fdigh.2018.00026, URL https://www.frontiersin.org/articles/10.3389/fdigh.2018.00026.

[10] S. Ji, J. Luo, and X. Yang, "A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions," *arXiv preprint arXiv:2011.06801* (2020).

[11] G. E. Lewis, "Too many notes: Computers, complexity and culture in" voyager"," *Leonardo Music Journal*, pp. 33–39 (2000).

[12] J. Biles *et al.*, "GenJam: A Genetic Algorithm for Generating Jazz Solos," presented at the *International Computer Music Conference*, vol. 94, pp. 131–137 (1994).

[13] C. Benetatos, J. VanderStel, and Z. Duan, "BachDuet: A Deep Learning System for Human-Machine Counterpoint Improvisation," presented at the *Proceedings of the International Conference on New Interfaces for Musical Expression* (2020).

[14] N. Jiang, S. Jin, Z. Duan, and C. Zhang, "Rl-duet: Online Music Accompaniment Generation Using Deep Reinforcement Learning," presented at the *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 710–718 (2020).

[15] C. Donahue, I. Simon, and S. Dieleman, "Piano genie," presented at the *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 160–164 (2019).

[16] C. Donahue, "Piano Genie," https://imaginary.github.io/piano-genie/ (accessed Mar. 10, 2023).

[17] Y. Mann, "A.I Duet," https://experiments.withgoogle.com/ai/ai-duet/view/ (accessed Mar. 10, 2023).

[18] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341 (2003).

[19] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of Musical Things: Vision and Challenges," *IEEE Access*, vol. 6, pp. 61994–62017 (2018 09), doi:10.1109/ACCESS.2018.2872625.

[20] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Zou, "Gradio: Hassle-free sharing and testing of ml models in the wild," *arXiv preprint arXiv:1906.02569* (2019).

[21] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, *et al.*, "Transformers: State-of-the-Art Natural Language Processing," presented at the *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45 (2020 Oct.), doi: 10.18653/v1/2020.emnlp-demos.6, URL https://aclanthology.org/2020.emnlp-demos.6.

[22] S. Raschka, J. Patterson, and C. Nolet, "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence," *Information*, vol. 11, no. 4, p. 193 (2020).

[23] M. Puckette, D. Zicarelli, *et al.*, "Max/MSP," *Cycling*, vol. 74, pp. 1990–2006 (1990).

[24] V. J. Manzo, *Max/MSP/Jitter for music: A Practical Guide to Developing Interactive Music Systems for Education and More* (Oxford University Press, 2016).

[25] D. Fober, S. Letz, *et al.*, "FAUST: an efficient functional approach to DSP programming," (2009).

[26] M. D. Network, "Web Workers API," https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API, accessed on June 6, 2023.

[27] M. D. Network, "Web Workers API," https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer, accessed on June 6, 2023.

[28] tonejs, "Tone.js," https://tonejs.github.io/ (accessed Mar. 15, 2023).

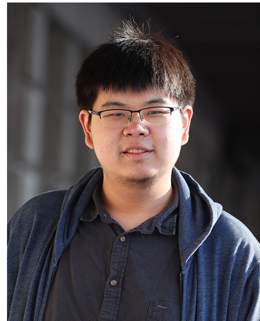
[29] L. K. Mohit Muthanna Cheppudira, Michael Scott Cuthbert *et al.*, "VexFlow," https://github.com/0xfe/vexflow (accessed Mar. 15, 2023).

[30] mrdoob, "Three.js," https://threejs.org/ (accessed Mar. 15, 2023).

[31] T. Winkler, "Making Motion Musical: Gesture Mapping Strategies for Interactive Computer Music," presented at the *International Computer Music Conference*, p. 26 (1995).

DRAFT

THE AUTHORS



Yongyi Zang

Christodoulos Benetatos

Zhiyao Duan

Yongyi Zang received his B.S. with honors in 2023, majoring in Audio and Music Engineering under the Electrical and Computer Engineering department at the University of Rochester, minoring in Computer Science. His research interest lies mainly in Computer Audition, as well as combining techniques with relevant fields, such as Natural Language Processing and Computer Vision.

●

Christodoulos Benetatos is a 5th year Ph.D candidate in Electrical and Computer Engineering department at the University of Rochester. He received his B.S and M.Eng in Electrical Engineering from National Technical University of Athens in 2018. His research interests lie primarily in designing and developing computer-assisted music-making systems.

●

Zhiyao Duan is an associate professor in Electrical and Computer Engineering, Computer Science and Data Science at the University of Rochester. He received his B.S. in Automation and M.S. in Control Science and Engineering from Tsinghua University, China, in 2004 and 2008, respectively, and received his Ph.D. in Computer Science from Northwestern University in 2013. His research interest is in computer audition and its connections with computer vision, natural language processing, and augmented and virtual reality. He received a best paper award at the Sound and Music Computing (SMC) conference in 2017, a best paper nomination at the International Society for Music Information Retrieval (ISMIR) conference in 2017, and a CAREER award from the National Science Foundation (NSF). He served as a Scientific Program Co-Chair of ISMIR 2021, and is serving as an associate editor for IEEE Open Journal of Signal Processing, a guest editor for Transactions of the International Society for Music Information Retrieval, and a guest editor for Frontiers in Signal Processing. He is the President-Elect of ISMIR.