Designing for a Pluralist and User-Friendly Live Code Language Ecosystem with Sema

Francisco Bernardo
Emute Lab, School of Music, University of Sussex
f.bernardo@sussex.ac.uk

Chris Kiefer
Emute Lab, School of Music, University of Sussex
c.kiefer@sussex.ac.uk

Thor Magnusson
Emute Lab, School of Music, University of Sussex
t.magnusson@sussex.ac.uk

 ICLC 2020, February 5-7, 2020, University of Limerick ,Limerick, Ireland

[1]Project MIMIC, `https://mimicproject.com`, accessed: 2019-09-15

Abstract

The growing popularity of the live coding and algorave scenes has inspired incentive and support for accessible, diverse and innovative approaches in expressing art through code. With live coding, the real-time composition of music and other art becomes a performance art by centering on the language of the composition itself, the code. Sema is a new open source system which aims to support user-friendly approaches to language design and machine learning in live coding practice. This paper reports on the latest technical advances and user research with Sema. We provide an overview and design rationale for the early technical implementation of Sema, including technology stack, architecture, user interface, integration of machine learning, and documentation and community resources. We also describe the activities of the MIMIC Artist Summer workshop, a full-week workshop with a group of 12 participants, which we designed and delivered to gather user feedback about the first design iteration of Sema. Findings from our workshop corroborate that language design and machine learning are advanced topics in computer science which may be challenging to users without such a background. Nevertheless, we found that such topics can inform the design of systems which may be both useful and usable to the live coding community.

Keywords: Programming Language Design, Web Live Coding, Machine Learning, User-Centred Design, Coding Ecosystems

Introduction

This paper presents Sema, a new Web-based, open-source, live coding language design and performance playground. Sema is aimed for real-time signal generation and processing, machine listening and machine learning. We are developing it as part of the AHRC-funded project MIMIC[1] ("Musically Intelligent Machines Interacting Creatively"), a three-year AHRC-funded project, run by teams at Goldsmiths Col-

lege, Durham University and the University of Sussex. MIMIC explores how to design and communicate machine learning and machine listening tools in simple and accessible ways for composers, instrument makers and performers. It does this through the design and adoption of new web-based computational tools that leverage on the internet as substrate for a live software ecosystem.

We are interested in the symbiosis of creative machine learning (Grierson et al. 2018) and live coding (Magnusson 2014) approaches to music. Live coding can facilitate pedagogical approaches to computational thinking in the context of creative and artistic practices (Roberts et al. 2016) and STEAM (Yee-King et al. 2017). We seek to understand how well new users from creative areas-i.e. as opposed to more technical backgrounds such as computer science and engineering are able to grasp and apply computational processes of considerable complexity, such as real-time interactive signal processing, machine learning model-building (Bernardo et al. 2017), and language design and grammar specification. We are employing user-centered techniques to leverage the design of new software development tools and evaluate progress through open-ended and creative processes (Bernardo et al., 2018). The paper is structured as follows: this section introduces the paper and presents background research around live coding systems and practices and machine learning. Section 2 presents an overview of the early technical implementation of our new system Sema and elements of our design strategy and rationale. Section 3 describes the MIMIC Artist Summer workshop and the activities for gathering user feedback about the first design iteration of Sema. In Section 4, we discuss the main findings and emerging themes of the workshop. Section 5 concludes with the main takeaways and future work.

Background

Live coding in the arts has existed as an exciting field of activity since the early 2000s, with seeding work and experimentation from previous decades. Live coding practitioners typically engage simultaneously in programming with a domain specific language (DSL) and other modalities, including audio and visual synthesis, instrument design, algorithmic creation, composition and performance (Magnusson, 2014). Early practitioners would typically invent their own systems for musical and other types of performance (e.g. McLean, 2004), often developing systems that were inspirational, humorous and highly effective in real-time performance under stress. With the growing popularity of live coding and algoraves (Armitage 2018), the live coding community appears to be consolidating their practices around a few systems-e.g. SuperCollider (McCartney, 2002), ixi lang (Magnusson, 2011), Gibber (Roberts and Kuchera-Morin, 2012), Sonic Pi and Overtone (Aaron and Blackwell, 2013), TidalCycles (McLean, 2014), ChucK (Wang et al., 2015) and Extempore (Sorensen, 2018). While such systems are excellent examples of established tools for live coding which help to build the live coding community and attract new beginners in, we have lost some of the variation and diversity which existed before.

Wakefield and Roberts (2017) have conducted previous research in language design for live coding on the Web. Their browser-based environment, which leverages a virtual machine, the Parsing Expression Grammar formalism, and an interactive online tutorial and documentation, aimed at enabling users to define custom DSLs syntax and semantic actions. Wakefield and Roberts also described the results deploying the system on an ICLC 2016 workshop, where a few participants were able employ it to develop their own mini-languages.

This paper follows up on this research and on a previous survey on the design of languages and environments for live coding presented at ICLC 2019 (Kiefer and Magnusson 2019) and the "Live Coding Machine Learning" workshop conducted at the ICLC 20191. In this engagement with communities of practitioners, we asked which features they envisioned for future live coding environments and languages that could integrate machine listening and machine learning. The findings indicated a wide space of possibilities, including support for hybrid approaches and multi-paradigm languages (i.e. OOP and functional), flexible, expressive and extensible languages and proto-

typing environments, good quality documentation and examples, as well as a clear and informative error report system. The emphasis that survey respondents placed on potential qualities for a new live coding language, e.g. brevity, simplicity, expressivity, flexibility, adaptability and plurality, pushed us to reconsider the idea of trying to satisfy everyone with one general live coding language design. Instead, we considered designing a new system which could enable and empower users to create and refine their own idiosyncratic languages for musical expression. Considering the history and tradition of live coders building their own systems (Magnusson 2014), this decision would contribute further to a plurality of systems in a field teeming with inventive solutions. We believe that the recent innovation in Web technologies can afford the evolution of an ecosystem of real-time, user-defined live coding languages which combines interactive machine learning, machine listening and audio threads. In this paper we account for the early stage of our design exploration aimed at fulfilling this vision.

SEMA, A Live Coding Language Design Playground

Our previous findings (Bernardo et al., 2019; Kiefer and Magnusson, 2019) inspired us to build a modern Web-based system to support rapid prototyping of live coding languages, which we titled Sema. We are engaged in a design exploration process pursuing the following principles:

- Integrated signal engine – no conceptual split between the language and signal engine. Everything is a signal
- Single sample signal processing – per-sample sound processing to support techniques that use feedback loops, such as physical modelling, reverberation and IIR filtering
- Sample rate transduction – signal processing with one principal sample rate-i.e. the audio rate-is simpler. Different sample rate requirements of dependent objects can be resolved by up-sampling and down-sampling. We use the 'transducer' analogy to enable and accommodate a variety of processes with varying sample rates (video, spectral rate, sensors, ML model inference) within a single engine.
- Minimal abstractions – no high-level abstractions such as buses, synths, nodes, servers, or any language scaffolding in our signal engine. Such abstractions sit within the end-user language design space.
- Striving for an adequate compromise between simplicity and flexibility – support the different user needs in the continuum which comprises beginner and expert live coders.
- Prioritizing usability and learnability – support a smooth and gradual learning curve, ease of use, and straightforward applicability.
- Balancing performance trade-offs with an efficient implementation – considered the constraints above and the performance overhead they may entail, we build upon an efficient implementation to optimize the utility of our system for live coding performance.

In this section, we provide a technical overview of the first design iteration of Sema. Figure 1 below illustrates the general architecture and main elements of our solution.

Machine Learning

Machine learning (ML) has been integrated into Sema as a first-class citizen and core component. ML processes have computationally intensive stages which can undermine the user experience of an interactive application. Previously, we described the critical usability issues of Web-based applications with interactive machine learning

(IML) workflows and audio, where end-users build custom ML models from small, lightweight, user-created data sets (Bernardo et al. 2018). In simple IML implementations, the ML model-training stage can have a thread-hogging behaviour which results in a freeze of the main JS thread. Furthermore, the ML-inference stage competes with the DOM and audio rendering, which may cause audio clicks and dropouts. These processes are therefore better suited for execution on a dedicated thread. This motivated the design of a multi-thread and loosely-coupled architecture for Sema, based on JS workers for ML and AudioWorklet for audio signal processing and rendering (further detailed in Section 2.2). Sema imports the latest version of Tensorflow.js (TFJS) into a JS Web worker and where it is used in the dynamic evaluation of the JS code for bespoke ML pipelines. This enables the user to enact parts of a ML workflow through partial evaluations of TFJS code related to different parts of the ML workflow, such as the set up the training datasets, inputs and outputs, the creation of a model architecture, the definition and configuration the models' hyperparameters, and communication with the user-defined live coding language context. The mechanism is similar to Jupiter Notebooks, where the user can evaluate different code blocks or regions in a non-linear fashion. ML processes in Sema adhere to our transducer concept, in that the sample rates from the event streams they receive from and generate to the live code language context, are converted to and from the sample rate of the audio context.

Signal Engine

The critical usability issues described in the previous section motivated the first step in our design strategy: to implement a signal engine which could run client-side in the browser, in a dedicated thread. Bernardo et al. (2019) provide a more detailed treatment of how we accomplished our innovative design pattern for an WAAPI AudioWorklet-based signal engine and of the performance tests conducted. In a nutshell, we refactored the C++ DSP library Maximilian (Grierson and Kiefer, 2011) and transpiled it into a WebAssembly[2] (WASM) module using Emscripten (Zakai, 2011). Our signal engine loads the Maximilian WASM module into a custom Web Audio API (WAAPI) AudioWorklet processor (AWP) (Choi, 2018). In the audio rendering loop, the AWP (Figure 1) evaluates dynamic DSP code which is injected through an the AudioWorklet asynchronous messaging system. One trade-off of our scalable and high-performance signal engine is that Sema inherits the current WAAPI AudioWorklet limitations and only runs in Chromium-based browsers (e.g. Chrome, Brave, Microsoft Edge, Opera).

Live Code Language Parser

In Sema's first-iteration implementation, which was used on the MIMIC Artist Summer workshop, users were required to employ and manually execute a Nearley.js[3] shell script to generate a new parser for a their user-defined live code language. The Nearley.js toolkit and library implements the Earley algorithm (Earley, 1970). Users needed to define and write a grammar specification in the Backus-Naur Form (BNF) and compile it against Nearley to generate a JS parser. The resulting parser would then be included in Sema's source code and the solution rebuilt. In comparison with other parsing formalisms (e.g. parsing expression grammars), the Earley algorithm supports a broader set of grammars, including ambiguous grammars with left-side recursion. The trade-off for the versatility and flexibility of Nearley[4] is performance. This is shown by comparisons with

[2] WebAssembly, `https://webassembly.org/`, accessed: 2019-09-15

[3] Nearley.js, `http://nearley.js.org/`, accessed: 2019-09-15

[4] Parsing Libraries Benchmark, `https://sap.github.io/chevrotain/performance/`, accessed: 2019-09-18

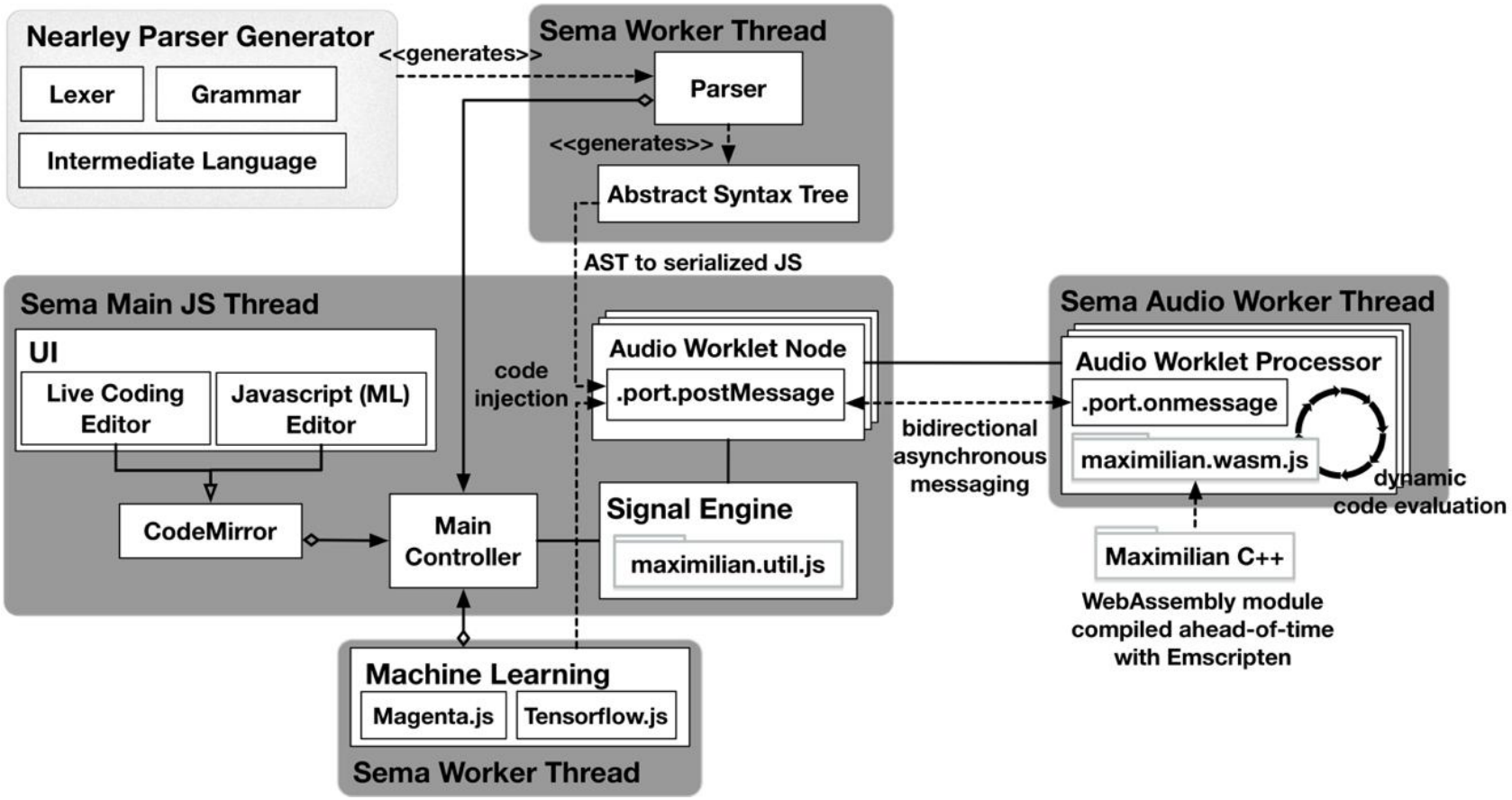

Figure 1: Sema's first-iteration architecture

parsing libraries, DSL and custom-written parser implementations, and other parsing approaches. However, the results from our previous performance tests (Bernardo et al., 2019) show that Nearley, even if slower than other parsers, performs in sub-perceptual time, which is, therefore, adequate for live coding performance.

Graphical User Interface and Code Editors

We experimented with different code editors while considering criteria such as component architecture, community adoption, maintenance and support, and ease of integration with webpack[5]. We opted for CodeMirror[6] to power the two user-facing editor instances in our web-based live coding environment. One CodeMirror instance runs a responsive live coding editor (Figure 2, top, dark background) which provides users with general code editing capacities and manual code evaluation using keyboard shortcuts (CMD-Enter). The other instance runs a second editor (Figure 2, white background) where the user can inspect, customize, or program TFJS-based ML-model pipelines from scratch, as well as define the communication bindings between the user-defined language and the ML worker threads. Sema's first GUI iteration is minimalist and provides a few command buttons, (Figure 2, bottom) for pausing and resuming audio rendering and downloading code from the editors to the local file system. Sema also provides a combo box button with a selection of pre-defined TFJS code for populating the second editor with JS code. This selection consists of a selection of pipelines for building specific ML models built into our system—e.g. simple linear regression (hello-world), two-layer non-linear regression, binary classification, Long-Short-Term-Memory (LSTM) for text generation, echo state networks, and transfer learning with a pre-trained Music Recursive Neural Network (RNN) from Google Magenta[7].

[5]Webpack, `https://webpack.js.org/`, accessed: 2019-09-18

[6]CodeMirror, `https://codemirror.net/`, accessed: 2019-09-18

[7] Google Magenta, `https://magenta.tensorflow.org/`, accessed: 2019-09-18

Workflows

```
@{%
const moo = require("moo");
const lexer = moo.compile({
  click: /click/,
  ws:    {match: /\s+/, lineBreaks: true},
});
%}

Statement -> %click
{% d => [{"@sigOut": {
              '@spawn': {
 '@sigp': {
   '@params':[{ '@num': { value: 1 }},
                  { '@string': 'click' }],
                          '@func': { value: 'loop'}
                  }
      }
        }
  }]
%}
```

Listing 1: Code Example 1

Section 2.3 introduced how the parser for the custom user-defined live code languages is created by the user. Listing 1 shows code of the first Sema tutorial to illustrate a minimal live coding language grammar, written in extended BNF and Sema's intermediate language. A

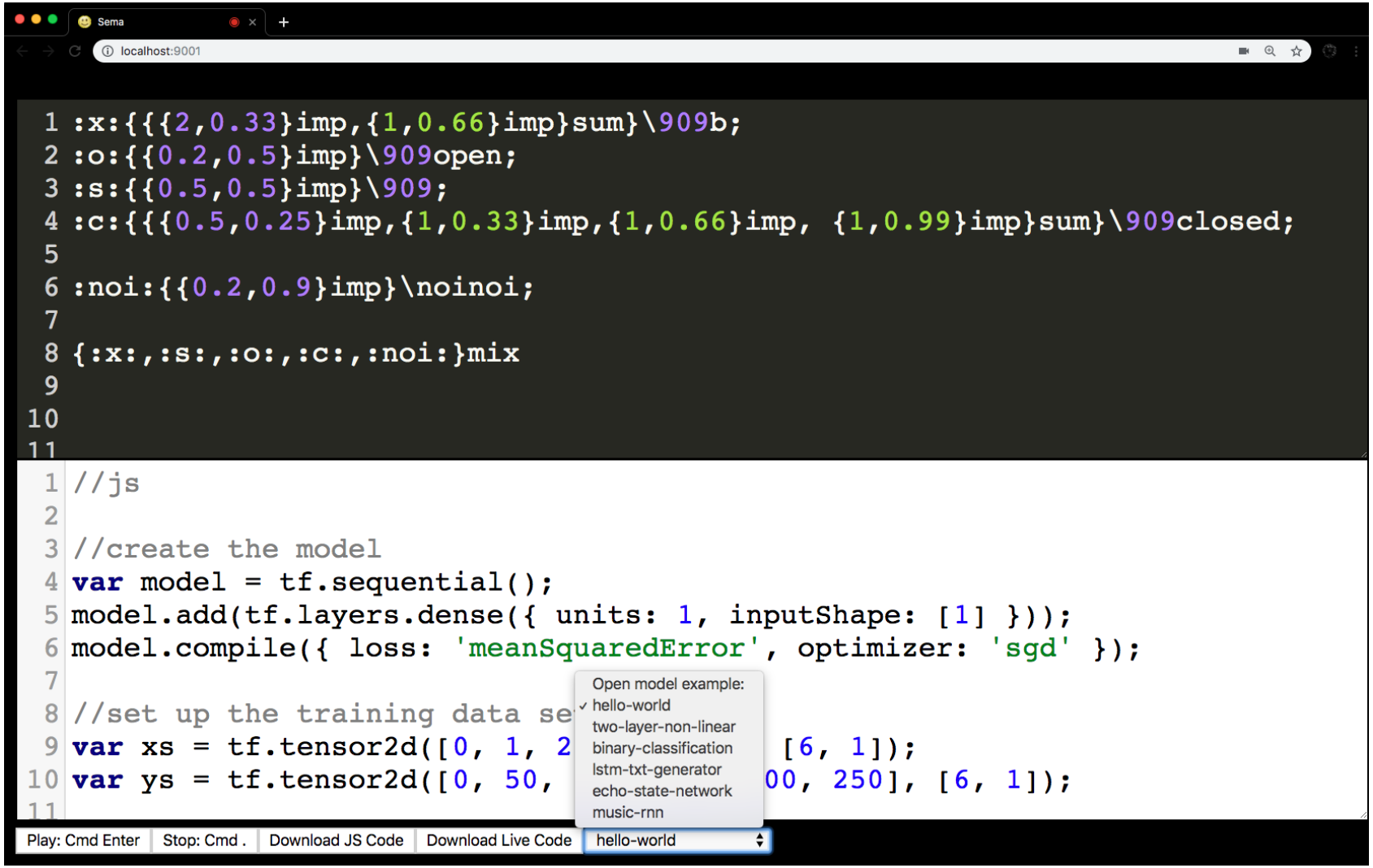

Figure 2: Snapshot of the GUI (pre-workshop version) with default language and machine learning model code

user compiles a file containing this grammar with Nearley to generate a parser for the 1-token language containing the expression "click". The parser is included in Sema source code and used when users evaluate an expression in the live coding editor — i.e. by pressing Cmd-Enter after selecting an expression or placing the cursor on a given line in the editor — and trigger the main workflow in Sema (Figure 1, dashed connectors). The user-evaluated expression is parsed by the Nearley-generated parser. If the expression is valid according to the language formally defined by the BNF grammar specification, the parser outputs an Abstract Syntax Tree (AST). The AST, a tree-like data structure which breaks down the user expression, is serialized to JS expressions that specify which Maximilian DSP objects are used and how they are assembled into DSP functions that will run in the AudioWorklet processor (AWP). These JS expressions are packed into a JS object which is posted through the AudioWorkletNode messaging port (Figure 1) and evaluated dynamically in the AWP audio loop .

Community and Learning Resources

Sema is hosted in a code repository on github.com[8] MIMIC-Sussex organisation, where it is published along other MIT-licensed satellite projects (e.g. osc2sema, sema.github.io). We are developing Sema using a modern web development stack based on node.js[9], webpack, and package managers such as yarn[10] or npm[11]. We are using this stack to leverage on automatic bundling workflows for code, assets, and integrating third-party code from the OSS ecosystem. The documentation for Sema comprises resources that assist the user in the described workflow. Currently, that includes:

- reference and code examples for the default demo language
- intermediate language representation for the signal engine
- reference for the DSP objects and methods of the Maximilan.js API
- data storage and loading functions

Other learning resources are tutorials (Listing 1) embedded in Sema's solution which aim to support a progressive learning curve to grammar editing and language design.

The Mimic Artist Summer Workshop

In this section we describe the elements, activities and results of the MIMIC Artist Summer workshop, which we designed and delivered to gather user feedback about the first design iteration of Sema.

Data Collection

We used an array of data collection methods before, during and after the workshop. We ran a pre-workshop survey to help us understand the background knowledge and skills, motivation, and project proposals of workshop candidates. Data collection during the workshop included participant interactions in the workshop in Slack channel, photos, video and sound recordings of participants' live coding performances with their customized environments, observational notes from the workshop, and notes from the final group discussion. Participants' forks and pull requests during the workshop are also part of the primary data set and publicly available from Sema's github

[8] MIMIC-Sussex/sema, `https://github.com/mimic-sussex/sema`, accessed: 2019-09-18

[9] Node.js, `https://nodejs.org/`, accessed: 2019-09-18

[10] `https://yarnpkg.com/`, accessed: 2019-09-18

[11] `https://www.npmjs.com/` accessed: 2019-09-18

repository. We also ran a post-workshop survey with questions on four main categories: live coding language design, the Sema system, machine learning and community.

Participants and Pre-Workshop Survey

The call for participation for the MIMIC Artist Summer Workshop[12] was released on May 24, 2019. The call addressed artists interested in participating in the workshop and using Sema to build their own live coding languages for live performances and composition using machine learning. The call presented a workshop week-long programme and introduced. We received 16 responses to our pre-workshop survey from which we selected 12 workshop participants (9 males, 3 females). Participants came from the UK (6 participants), Netherlands (2), Norway (1), Germany (1), Sweden (1) and Spain (1). With only one exception, a participant who reported having beginner coding skills, most participants reported being very experienced coders in multiple languages (e.g. JavaScript, Python, C++), including CS graduates, PhD students and teachers of programming. Most participants reported being experienced live coders (e.g. SuperCollider, TidalCycles, ixi lang) also with skills in data-flow languages (e.g. Pure data and Max/MSP). One participant mentioned never having live coded, two participants mentioned not having performed live coding in public. In relation to machine learning skills, the group was more diverse. Half of the group reported having little to no experience in machine learning. Other participants reported having tinkered with a ML toolkit—e.g. Wekinator, GRT, Keras, ml.lib for Max/MSP, and SuperCollider ML tools. Three participants reported having advanced knowledge in ML, two with publications or artworks in the area.

Some of the reasons and motivation that users expressed for attending the workshop included:

- developing their practice in live coding and performing
- understanding the possibilities of machine learning in music
- enhancing their knowledge about machine listening and machine learning
- building new musical instruments and tools
- developing new methods of performance and interacting with audience
- expanding their social network in the machine learning for music and live coding community and meeting like-minded people, learn how to communicate with people in the field
- finding new teaching material
- creative JS coding and exploration

Participants' proposals for projects at pre-workshop stage included:

- clive coding environment with live acoustic audio inputs and algorithmic processing
- building a new instrument with relevant musical parameters for both the performer and the audience
- building tools for song writing
- expanding a personal live coding environment with generative algorithms and recommendations for composing melodic/rhythmic structures
- exploring the possibilities of real-time synthesis using machine learning to create new sounds
- controlling feedback systems
- a live coding system for 3D printing, a rule-based learning system for live coding

[12] MIMIC Artist Summer Workshop, `http://www.emutelab.org/blog/summerworkshop`, accessed: 2019-09-15

Overview of the Workshop Week and Sessions

Workshop sessions took place at the Sussex Humanities Lab (SHL) with the exception for the performance night. The first day of the workshop started with a contextualisation of the workshop within MIMIC research goals and outline of the workshop week activities (Table 1.1). Participants were invited to participate at EMUTE LAB live coding performance night "Musically Intelligent Machines" by the end of the week at the local venue Rose Hill.

Day	Topic
1	Induction session on Language Design with Sema & Induction session on Machine Learning and Sema
2	Counterpoint studio presentation and workshop session
3	Induction session on Machine Listening
4	(Aesth)et(h)ics and creative-AI & Live coding performances with participants' systems at music venue
5	Artist Residence Project showcases + Participant demos & Discussion about Sema (experience, requests, future path)

Table 1.1

The core sessions with Sema were delivered on the first day, one in the morning and the other in the afternoon. The remaining days had blocks of project work interweaved with inspirational and debate sessions, garden lunches, and social activities in Brighton.

The first day workshop sessions with Sema consisted of a practical crash-course and hands-on exploration on language design (Figure 1 a) and an introduction to machine learning and Tensorflow.js. These sessions were preceded by a demonstration of supporting tools, installation and forking of the Sema repository. We introduced Sema's tutorials for language design and grammar specification using extended BNF, Nearley and Sema's intermediate language. We went through simple examples (e.g. Listing 1) to gradually more complex while attempting to get everyone up to speed for them to proceed in autonomous exploration. The machine learning session provided an overview covering ML concepts and terminology, artistic examples and applications, and a walkthrough the Tensorflow.js examples provided with Sema.

Samuel Diggins and Tero Parviainen from Counterpoint[13] studio gave a presentation (Figure 4 a) about their projects with computational design with ML and music participated in the first two days of the workshop. Shelly Knotts from MIMIC-Durham presented some examples of the MMLL library for machine listening (Figure 4 b).

In the EMUTE LAB 4 performance evening, six participants performed along other artists in line-up (Figure 5 a). Three participants performed individually using Sema (Figure 5 a, b, c). Marije Baalman and Henrike Hurtado Mendieta, two artists who were invited to do a 2-week long MIMIC residency, participated in the workshop, performed at the EMUTE lab evening and also presented their work in a session (Figure 6 a and b). In the same morning workshop participant also presented their work in the workshop (e.g. Figure 6 c).

3.4 RESULTS There were two new languages created with Sema during the MIMIC Artist Summer Workshop. One workshop participant created a language titled MAIA[14] and performed with it. The artist-in-residence Marije Baalman created another new language. Two other participants customised Sema and performed with the default language. One of them augmented Sema with 3D graphics and animations, and sonified the machine learning training stage. The other performer developed a probabilistic system that communicated

[13]Counterpoint creative studio, `https://ctpt.co/`, accessed: 2019-09-19

[14]MAIA – Live coding mini-language build upon SEMA, `https://github.com/tmhglnd/maia`, accessed: 2019-09-26

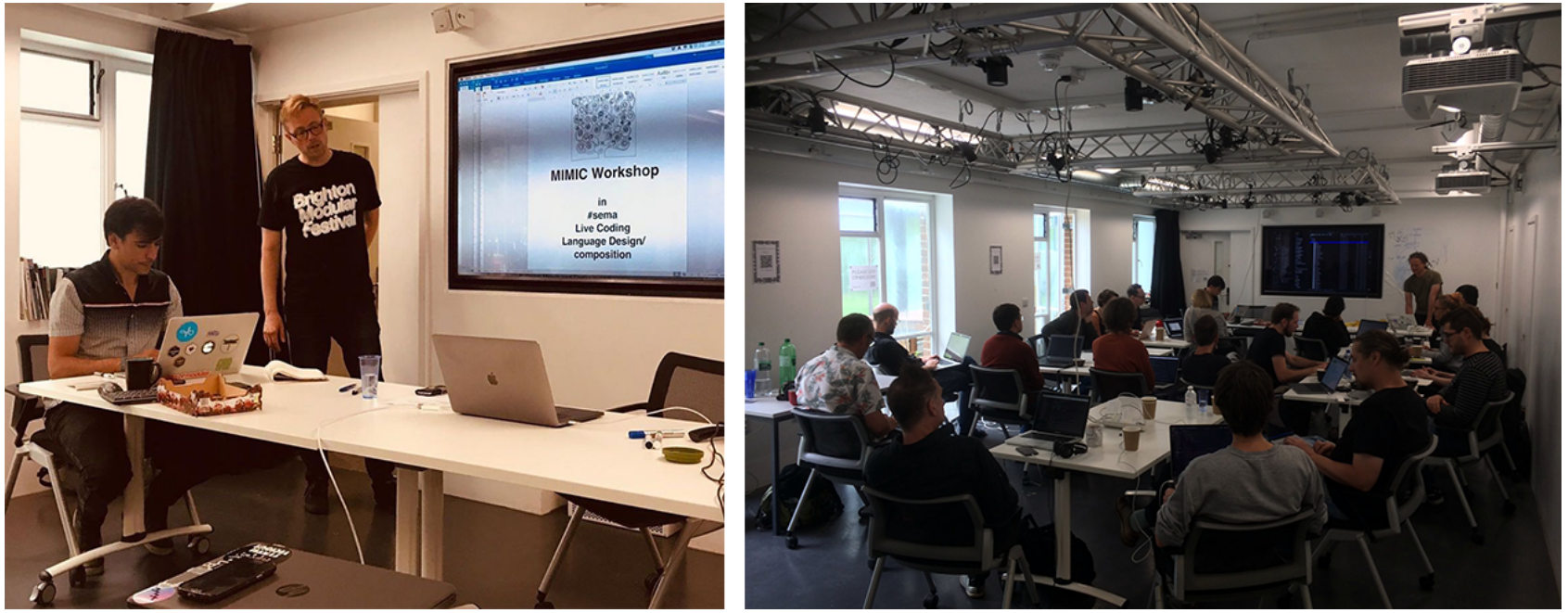

Figure 3: a) The MIMIC Artist Summer workshop opening session and b) the language design induction session with Sema at the Sussex Humanities Lab.

with the live code language to stochastically change tones of the musical sections. Interestingly, one other participant designed a grammar with Nearley playground and used it for generating new text in performance. Other contributions to Sema included extensions to the intermediate language —'amsynth', 'fmsynth', 'oscbank', i.e. three intermediate language constructs corresponding to an AM synthesizer, an FM synthesizer and a bank of oscillators. One contribution consisted of an integration of melody-rnn[15], a pre-trained model from Magenta. There were very meaningful contributions to the documentation. One participant refactored the tutorials with more complete and adequate comments for beginners. Another participant documented the intermediate language. We obtained 12 respondents to our post-workshop survey. We employed NVivo in a qualitative content analysis of participants responses. Participants' names were anonymized and en-

[15]Magenta Models, `https://github.com/tensorflow/magenta/tree/master/magenta/models`, accessed: 2019-09-26

Figure 4: a) Counterpoint presentation on designing with Music and AI, and the b) machine listening session at SHL

coded with labels from the range [MP01-MP12]. The codes employed to classify textual content included: sema, audio engine, machine learning, machine listening, grammars, regexp, documentation, tutorials, workshop, language design, knowledge, experience, programming language paradigms, community, contributing, goals, challenges, suggestions, functionality, understandability, learnability, utility, performance, limitations, negative feedback, positive feedback.

Discussion

In this section, we discuss themes synthesized from the analysis of the main results and primary data.

Signal Engine: Good Audio Quality and Reliability, but more Flexibility and Transparency Required

The audio quality of Sema's signal engine was considered good in general, and in one case, surprisingly solid and reliable for an earlier implementation. Participants who used Sema in their performances also had technically good sounding performances. This confirms the quality and reliability of our signal engine prototype (Bernardo et al., 2019) in particular for use in a live performance setting. These results also show that our strategy for implementing a browser-based signal engine running on its own thread was sound. In one case, our signal engine implementation enabled the sonification of the ma-

Figure 5: a) Poster for the EMUTE LAB 4 night, where b) c) d) three participants performed with Sema L

chine learning model training stage. This shows important improvements over previous technical challenges in integrating machine learning with audio in web-based applications (Grierson et al., 2018), such as the thread-competing behavior of machine learning, as well as audio clicks and drop-outs. There were a few concerns the recurred among participants. One other concern was about the limitations of a browser-based signal engine in terms of processing capacity. This remains an open question for further research with load tests and experimentation in live performance scenarios. One possibility is to explore the trade-offs of an Electron-based build of Sema. There were also concerns was about the investment required to learn yet another audio engine implementation when participants took previous effort with another language—e.g. SuperCollider (McCartney, 2002). In two cases, there were remarks about how the functional 'flavour' and signal-flow-oriented architecture of the audio engine could limit the musical outcomes and artistic expression with Sema. Some suggestions included developing support in Sema for procedural and object-oriented approaches to the intermediate representation. On the other hand, some remarks pinned these limitations to the language design workflow. Such aspects are tied to both the musical affordances of future languages, to the usability of the different components of Sema and to the language-designer experience, all of which require further research and are discussed in the next section.

General Usability, Learning Resources and Documentation Require Improvement

The potential of a language design system for the live coding community was considered useful and appealing. However, several aspects of Sema were considered obscure and very challenging. There were general difficulties with understanding how to use the intermediate language in the grammar specification and how to build the AST; or, understanding how the mechanism of converting the AST with the intermediate language to audio DSP worked. On one hand, these difficulties were related to the specifics of the implementation of Sema,

Figure 6: Final day presentations with a) b) artists-in-residence showcase and c) participants demos

which lacked transparency and abstraction in certain areas and also failed to provide users with adequate documentation. On the other hand, these difficulties are intrinsic to language design, which is considered an advanced topic of computer science. Nevertheless, despite the cumbersomeness of the language design and grammar specification workflow, and of manual and external parser compilation, we observed that people were able to design valid grammars and languages. We got very positive feedback about the Nearley playground for rapid prototyping and exploration of throw-away grammars. The approach of using minimal tutorials to support the gradual exploration of custom-designed languages through adjustments, trial and error, was considered useful and helpful. However, tutorials were considered mostly incomplete and beginner un-friendly, with suggestions for supporting different entry points and skill levels. It is fundamental to improve the usability of Sema and the complex processes that it leverages with better learning resources on conceptual knowledge, system documentation, examples and code comments, as suggested. We found that there is little research on systematic approaches to language design workflows, documentation and learning resources, particularly for live coding languages. There is research sharing a common base of HCI and usability, and focusing on improving API usability (Myers and Stylos, 2016), on design guidelines (Karsai et al., 2009) and usability (Barišić et al., 2013) of DSLs, which we are looking forward to explore with Sema.

Finding the Adequate Approaches, Models, Uses and Data for Machine Learning in Live Coding Practice

There are compelling opportunities for empowering the live coding community with new artistic processes which may arise from the inte-

gration of real-time interactive signal processing and machine learning technologies. Particularly, if such processes are provided in a scalable and accessible environment such as the Web. This was reflected, for instance, by the general appreciation for the knowledge improvement that Sema and our workshop facilitated around different aspects and layers of ML—e.g. from ML concepts and terminology through to specific implementations with Tensorflow.js; also, for how Sema and the workshop attempted to bridge domain-specific concepts of live-coding music performance and interactive audio, in a playful and accessible way.

In our surveys and during discussions, we noticed an overall uncertainty and ambivalence about the utility and use cases of machine learning in live coding. Further research with Sema will prioritize reaching an understanding of which ML-algorithms fulfil a specific live coding use case better. There were interesting workshop outcomes which can help to lead future research with this question. For instance, there were remarks acknowledging that the real-time nature of live coding performance and lack of extensive data sets should be considered. This hints to the future design of Sema to adhere to the live coding constraints pf real-time and small data sets, including curation of machine learning algorithms and probabilistic models (e.g. kNN, Markov Models, RNN), understanding which ML approaches are more suitable for these constraints, for instance, interactive machine learning (Bernardo et al., 2017), and other which may be simultaneously valuable for generation, such as transfer learning (Oore et al., 2018) or reinforcement learning (Jaques et al., 2016).

Design Decisions

The findings and user feedback which we obtained in the workshop helped us to consider, define and prioritize the main development goals for the subsequent iterations of our user-centered design exploration. They are as follows:

- Integrate grammar design and parser compilation in Sema's workflow
- Define and clarify entry points into Sema to improve learning resources and documentation
- Explore the links and adequate abstractions for designing workflows, GUI, AST, intermediate language, machine learning and machine listening
- Explore the adequate ML approaches, models and use cases for live coding
- Design the OSS community strategy and prepare for contributions

Conclusions

In this paper, we presented Sema, a new Web-based OSS system for live coding language design and performance with real-time signal generation and processing, machine listening and machine learning. We contextualised Sema within the research activities of MIMIC, presented the underlying motivation for its development, and presented an overview of the latest technical advances and user research with Sema. We also discussed the main findings and themes which emerged from this work. One group of main findings relate to the quality of the current signal engine implementation and how it enabled to overcome previous challenges in the integration of ML and audio in Web-based applications. Another group findings concerned the challenges and difficulties which users found with the workflows in Sema and general usability issues. A third group of findings concerns the usefulness of Sema as a resource for leveraging pedagogical approaches and learning experiences with ML learning, and which requires further exploration around the provision of the adequate approaches, models and use cases for live coding practice. Sema promises utility and value for the live coding community by filling the gap of systems that support new language design. Whilst the potential to enable users to create their own

languages in a simple web-based playground is strong, Sema needs extensive work to become more usable and welcoming to novices. Future work includes writing better learning resources, making the signal engine more flexible and transparent, and identifying and implementing adequate ML approaches, modes and use cases. We also noticed how people were not be able to grasp the full potential of ML in live coding practice, and this is one of MIMIC's key project objectives. Finally, in this paper, we used workshop findings to motivate and present design and development goals for the next design iteration of Sema. Acknowledgments We would like to thank the participants of the workshop for their participation in workshop and contributions to Sema. We would like to thank Marije Baalman and Enrike Hurtado Mendieta for their inspiring work at the MIMIC residency, participation in the workshop sessions and contributions to Sema. We would like thank Samuel Diggins and Tero Parviainen for their inspiring workshop sessions and contributions to Sema. Finally, we would like to thank Paul McConnell and Alex Peverett for the documentation of the workshop. The research leading to these results has received funding from AHRC through the MIMIC project, ref: AH/R002657/1 `https://gtr.ukri.org/projects?ref=AH/R002657/1`.

References

Aaron, S., Blackwell, A.F., 2013. From Sonic Pi to Overtone: Creative Musical Experiences with Domain-specific and Functional Languages, in: Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling – FARM'13. pp. 35–46. https://doi.org/10.1145/2505341.2505346

Armitage, J., 2018. Spaces To Fail In: Negotiating Gender, Community and Technology in Algorave. Danc. J. Electron. Danc. Music Cult. 10, 31–45.

Barišić, A., Goulão, M., Amaral, V., Barroca, B., 2013. Evaluating the usability of domain-specific languages, in: Software Design and Development: Concepts, Methodologies, Tools, and Applications. pp. 2120–2141. https://doi.org/10.4018/978-1-4666-4301-7.ch098

Bernardo, F., Grierson, M., Fiebrink, R., 2018. User-Centred Design Actions for Lightweight Evaluation of an Interactive Machine Learning Toolkit. J. Sci. Technol. Arts 10, 2. https://doi.org/10.7559/citarj.v10i2.509

Bernardo, F., Kiefer, C., Magnusson, T., 2019. An AudioWorklet-based Signal Engine for a Live Coding Language Ecosystem, in: Web Audio Conference. Trondheim.

Bernardo, F., Zbyszyński, M., Fiebrink, R., Grierson, M., 2017. Interactive Machine Learning for End-User Innovation, in: Proceedings of the Association for Advancement of ArtificiaI Intelligence

Symposium Series: Designing the User Experience of Machine Learning Systems. pp. 369–375.

Choi, H., 2018. AudioWorklet: The future of web audio, in: Web Audio Conference.

Dannenberg, R.B., Mercer, C.W., 1992. Real-Time Software Synthesis on Superscalar Architectures, in: International Computer Music Conference, International. pp. 174–177. https://doi.org/10.2307/3681016

Earley, J., 1970. An efficient context-free parsing algorithm. Commun. ACM 13, 94–102. https://doi.org/10.1145/362007.362035

Grierson, M., Kiefer, C., 2011. Maximilian: An Easy to Use, Cross Platform C++ Toolkit for Interactive Audio and Synthesis Applications, in: Proceedings of The International Computer Music Conference. pp. 276–279.

Grierson, M., Yee-king, M., McCallum, L., Kiefer, C., Zbyszyński, M., 2018. Contemporary Machine Learning for Audio and Music Generation on the Web: Current Challenges and Potential

Solutions, in: Proceedings of The International Computer Music Conference.

Jaques, N., Gu, S., Turner, R.E., Eck, D., 2016. Generating Music by Fine-Tuning Recurrent Neural Networks with Reinforcement Learning. Thesis 410–420.

Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S., 2009. Design Guidelines for Domain Specific Languages. Proc. 9th OOPSLA Work. Domain-Specific Model.

Kiefer, C., Magnusson, T., 2019. Live Coding Machine Learning and Machine Listening: A Survey on the Design of Languages and Environments for Live Coding, in: Proceedings of the International Conference on Live Coding. Madrid.

Magnusson, T., 2014. Herding Cats: Observing Live Coding in the Wild. Comput. Music J. 38, 91–101. https://doi.org/10.1162/COMJ

Magnusson, T., 2011. The IXI Lang: A Supercollider Parasite For Live Coding, in: Proceedings of the International Computer Music Conference. pp. 5–8.

McCartney, J., 2002. Rethinking the computer music language: SuperCollider. Comput. Music J. 26, 61–68. https://doi.org/10.1162/014892602320991383

McLean, A., 2014. Making Programming Languages to Dance to: Live Coding with Tidal, in: Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling –

FARM'14. pp. 63–70. https://doi.org/10.1145/2633638.2633647

McLean, A., 2004. Hacking Perl in Nightclubs [WWW Document]. Perl.com.

Myers, B.A., Stylos, J., 2016. Improving API usability. Commun. ACM 59, 62–69. https://doi.org/10.1145/2896587

Oore, S., Simon, I., Dieleman, S., Eck, D., Simonyan, K., 2018. This Time with Feeling: Learning Expressive Musical Performance. Neural Comput. Appl. 1–24. https://doi.org/10.1007/s00521-018-3758-9

Roberts, C., Allison, J., Holmes, D., Taylor, B., Wright, M., Kuchera-Morin, J., 2016. Educational design of live coding environments for the browser. J. Music. Technol. Educ. 9, 95–116. https://doi.org/10.1386/jmte.9.1.95_1

Roberts, C., Kuchera-Morin, J.A., 2012. Gibber: Live coding audio in the browser. ICMC 2012 Non-Cochlear Sound - Proc. Int. Comput. Music Conf. 2012 64–69.

Sorensen, A.C., 2018. Extempore: The design, implementation and application of a cyber-physical programming language. Australian National University.

Wakefield, G., Roberts, C., 2017. A Virtual Machine for Live Coding Language Design. Proc. New Interfaces Music. Expr. 2017 275–278.

Wang, G., Cook, P.R., Salazar, S., 2015. ChucK: A Strongly Timed Computer Music Language. Comput. Music J. 39, 91–101. https://doi.org/10.1162/COMJ

Yee-King, M., Grierson, M., D'Inverno, M., 2017. STEAM WORKS: Student coders experiment more and experimenters gain higher grades, in: IEEE Global Engineering Education Conference, EDUCON. IEEE, pp. 359–366. https://doi.org/10.1109/EDUCON.2017.7942873

Zakai, A., 2011. Emscripten: An LLVM-to-JavaScript Compiler, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion. ACM.