Enhancing Expressive Musical Conversation in the JAM_BOT

Lancelot Blanchard*
lancelot@media.mit.edu
MIT Media Lab
Cambridge, USA

Perry Naseck*
pnaseck@media.mit.edu
MIT Media Lab
Cambridge, USA

Katherine Liang
krliang@mit.edu
Massachusetts Institute of Technology
Cambridge, USA

Joel Tan
joeltjy1@mit.edu
Massachusetts Institute of Technology
Cambridge, USA

Heidi Lei
hlei@mit.edu
MIT Music Tech
Cambridge, USA

Cheng-Zhi Anna Huang
huangcza@mit.edu
MIT Music Tech
Cambridge, USA

Joseph A. Paradiso
joep@media.mit.edu
MIT Media Lab
Cambridge, USA

Figure 1: Musician playing with the JAM_BOT on a player piano. The MIDI foot switch is used to trigger the system generation.

Abstract

Previous work introduced the JAM_BOT, a real-time system that embeds live music language models capable of generating symbolic music sequences coherent with a performer's input. The system supports multiple interaction strategies that have been demonstrated in several public performances. However, these strategies limit expressive musical conversation by constraining tempo, form, or musical roles. We extend the JAM_BOT to support more expressive, open-ended interaction through four key improvements: (1) modeling velocity, a key dimension of expression in symbolic music; (2) increasing model throughput via a ggml implementation–required to accommodate the longer sequences induced by velocity modeling; (3) developing a new training modality that enables free-form call-and-response interaction across varying tempi; and (4) compensating for external MIDI output latency to ensure rhythmic coherence with the performer. We quantitatively evaluate the model throughput improvement and our latency compensation strategy, and offer generated MIDI samples online.[1] Together, these enhancements enable the JAM_BOT to engage in natural, expressive musical conversation, eliminating key musical limitations to enable the development of future performances and installations.

Keywords

AI, artificial intelligence, machine learning, music, expression, velocity, performance, optimization, piano

*Both authors contributed equally to this research.

NIME '26, June 23–26, 2026, London, UK

[1] https://jambot.media.mit.edu/nime2026

1 Introduction

Artificial intelligence methods have a long history in musical performance interfaces, ranging from early rule-based and statistical approaches to modern deep learning models. Early pioneering systems such as GenJam [4] used genetic algorithms to evolve jazz solos in real time alongside human performers, while Pachet's Continuator [20] employed Markov models to learn a musician's style on the fly and generate stylistically coherent responses. More recent work has explored neural and latent-space approaches for interactive improvisation [1, 11, 15]. However, there has been little work exploiting the modeling capabilities of music Language Models (LMs) for real-time uses [3, 9]. In previous work, we introduced the JAM_BOT, a novel system able to generate sequences of symbolic music in real-time, in response to the live MIDI input of a human performer [5]. The JAM_BOT is one of the first systems to enable the use of large music LMs in real time through an optimized inference architecture, and has been featured in multiple performances. However, because of its design and training data, it imposes constraints on the form (e.g., trading sequences of four bars), tempo, or musical role that a performer has to decide on during a performance. The original JAM_BOT does not model velocity, and its note scheduling is subject to inherent MIDI output latencies, limiting its effectiveness for expressive musical conversation. Collaborative improvisation between humans is sometimes described as a "musical conversation" [2, 22] due to the similarity it shares with verbal interaction, such as a shared vocabulary and a capability for both participants to steer an improvised dialogue. The aforementioned limitations of the JAM_BOT make it difficult for a natural musical conversation to take place by limiting the shared vocabulary and taking control away from the human user. To enable this type of interaction, we develop four key improvements that extend the abilities of the JAM_BOT:

First, we model velocity, a key characteristic of expressive symbolic music sequences. While expressive symbolic music sequences containing velocity are modeled in early architectures such as Performance RNN [23] and Music Transformers [14], more recent architectures such as the GPT-2-based Anticipatory Music Transformer (AMT) [25] sacrifice expression modeling to focus on the pre-training of base models on a larger amount of data. In this work, we combine the pretraining benefits of AMT with the expressive capabilities of the original Music Transformer work.

Second, since modeling velocity requires generating more tokens, we increase the throughput of the JAM_BOT's underlying music LMs. In the first iteration of this work, we use *ONNX Runtime* to efficiently run *ONNX* model inferences in C++, allowing the models to run faster than real-time (also defined as having a "real-time factor" (RTF) greater than 1 [24]). In this iteration, we use the *ggml* framework [13], another Machine Learning framework, which exhibits great speedup benefits.

Third, we develop a new interaction strategy that enables the human musician to perform with the JAM_BOT by creating musical prompts of arbitrary length and tempo in a *call-and-response* modality, facilitating the musical conversation by setting clearer and more intuitive expectations. We do so by collecting an expert dataset of MIDI sequences recorded on a piano with annotated *calls* and *responses*, and use it to finetune an AMT model.

Finally, we implement a way for the JAM_BOT to offset any MIDI output latency by scheduling generated notes to be played earlier, effectively enabling it to account for hardware- or network-induced delays in a systematic way, removing the need for alternative methods of combating latencies as proposed in other interfaces [16]. This allows the JAM_BOT to better preserve rhythmic coherency and create smoother transitions when interacting with a human performer.

Together, we believe that these improvements can allow human performers and the JAM_BOT to engage in expressive musical conversation, making it an optimal interface to interact with music LMs in real time.

2 Methodology

2.1 Modelling Velocity

In the MIDI protocol, velocity refers to the volume at which a note is played. A value between 0 and 127 captures the intensity of the note and enables the modeling of expressive dynamics, phrasing, and articulation. In order to effectively model velocity, the vocabulary of the model needs to be adapted to accommodate this new dimension. As with the previous iteration of this work, we use the AMT models (in particular, the `medium` model with around 400M parameters) as our pre-trained model for MIDI sequence generation. AMT represents every note on/off pair as a tuple of three tokens: an *onset time* $\mathbf{t} \in \{0, ..., 9\,999\}$, a *duration* $\mathbf{d} \in \{0, ..., 999\}$, and a *note* $\mathbf{n}$–which combines the *pitch* $\mathbf{p} \in \{0, ..., 127\}$ with the *instrument* $\mathbf{k} \in \{0, ..., 128\}$ as $\mathbf{n} = 128\mathbf{k} + \mathbf{p}$. To add velocity information, we need to add a velocity token $\mathbf{v} \in \{0, ..., 127\}$ to each note. We first considered adding this token by merging the velocity information with another token, similar to the *note* token–for example, defining a *duration-velocity* hybrid token $\mathbf{dv} = 128\mathbf{d} + \mathbf{v}$. The drawbacks of this method are: the substantially increased vocabulary size of $128 \times 1,000$ tokens (introducing many randomly-initialized embeddings that need to be learned), increased training time, and decreased stability. Instead, we encode each note as a tuple of four tokens: $(\mathbf{t}, \mathbf{d}, \mathbf{n}, \mathbf{v})$. While this method increases the vocabulary by only 128, it reduces the number of notes from 341 to 256 that can fit within a $1,024$-token fixed context length. In order to enable the *anticipation* mechanism of AMT, we further add 128 tokens for anticipated velocity. Finally, we resize the model's input embedding and output projection layers accordingly, and randomly initialize the newly-added parameters before training. To increase data diversity, we also perform data augmentation by scaling individual note velocities by a random factor between 0.5 and 2.0 at a specified rate. Outside of these modifications, we train the model in the same way as our original paper. In particular, and as justified in the original paper, we ensure that the model is allowed to overfit the training data to provide guarantees on the style and structure of its outputs at performance time.

It is important to note that, by expanding the vocabulary of the pre-trained model and changing its note representation, we risk losing some of the benefits of the large pre-training on the Lakh MIDI dataset [21]. However, previous work on *transfer learning* in the domain of symbolic music [8] shows that further finetuning on slightly different representations can effectively help large pre-trained models learn new behaviors while keeping the original pre-training benefits. We use these results and finetune a `medium` AMT on a new dataset (described in 2.3) using this new velocity representation.

2.2 Increasing Model Throughput

In the original JAM_BOT, we used Microsoft's *ONNX Runtime* to run model inferences [18]. *ONNX Runtime* provides a portable cross-platform runtime system: it supports various backends and operating systems including NVIDIA CUDA and Linux. *ONNX Runtime* automatically traces the compute graph of each model, which reduces the implementation time and complexity. However, it has some disadvantages for our application: First, *ONNX Runtime* has become a very large framework with many features. This results in large supporting binaries and increased execution overhead. While *ONNX Runtime* provides binaries/shared libraries for some platforms, building the framework for other platforms (such as NVIDIA's ARM-based *DGX Spark GB10* platform) takes a considerable amount of time and computation power; building on a continuous integration runner can take upwards of an hour. Additionally, *ONNX Runtime* does not well support Apple macOS Metal acceleration. In practice, only the *ONNX Runtime* CUDA backend can be used for the JAM_BOT.

To improve the model throughput and portability, we now use the *ggml* framework [13]. *ggml* requires explicit implementation of the compute graph, but it has additional optimization and significantly reduced overhead. We compile *ggml* into our binary packages as a set of shared libraries. *ggml* compiles an individual shared library for each accelerated backend and CPU instruction feature set. These shared libraries are dynamically loaded at runtime. When the JAM_BOT application starts, *ggml* scores each CPU backend option by supported features on the CPU and loads the best one. In practice, our smallest models (170M parameters) can run short generation segments with the ggml *ggml* CPU backend and AMD AOCL BLAS optimization on an AMD Ryzen 9 7950X CPU. CPU backends do not perform as well on the same processor with *ONNX Runtime*. Additionally, *ggml* supports NVIDIA CUDA, Apple Metal, Vulkan, and AMD ROCm backends (among others). The *ggml* implementation for Apple Metal allows for the JAM_BOT to now perform on common Apple laptop and desktop computers.

2.3 Call-and-response Training

A key consideration in designing live music models is their interaction modality. In particular, when working with autoregressive music LMs, we need to consider when the model will be prompted, when it will be refreshed, and when it will respond to the human agent. In previous work [5], we designed three *interaction strategies* to control when the model should be prompted: (1) *at regular time intervals*, (2) *at every musical gesture*, and (3) *on request*. These strategies all enable different kinds of interplay with the JAM_BOT, allowing the human performer and the AI agent to either take turns or to play at the same time.

Follow-up work [6] experimented with giving the ability for the performer to control the start and end of musical gestures through the use of a MIDI foot switch. This enabled a more fluid musical conversation to take place between the performer and the JAM_BOT, allowing the performer to have more agency on the start and end of the system's generations. An observation that stemmed from this work was that the outputs of the system tended to be highly stochastic and unpredictable due to multiple factors described later. While this was sometimes seen by participants as an opportunity for creative interplay, it was also a limitation hindering the development of a natural musical conversation due to the lack of control in the human-AI handoff and in the musical roles, as well as a lack of choice in whether gestures are followed or contrasted. In this iteration of the work, we aim to make this interaction more predictable and controllable.

We hypothesize the lack of control in the modality to be due to two main factors. First, the PiJAMA dataset [10], which the model was fine-tuned on, is much more diverse than the custom datasets we collected for the previous iteration of the project. This makes it harder for the model to capture a specific style, making independent generations more likely to significantly differ. Second, the training of the model focuses on the task of sequence continuation, which does not necessarily align well with a performance context. When using the system, participants naively expected the JAM_BOT to respond to their prompt in a similar way that a human performer would.

To improve on these shortcomings, we devise a new model finetuning strategy that explicitly allows the model to learn the *call-and-response* task in a way that feels natural for the performer. We formally model the distribution:

$$p(y_t | y_{1:t-1}, x_{1:n})$$

where $x_{1:n}$ is a sequence of notes of length n that represents the performer's input ("*call*"), and $y_{1:t}$ is a sequence of notes of length t that represents the model's output ("*response*"). Using the tokenization scheme from previous iterations of the JAM_BOT, and introduced by AMT, we represent the *call* and the *response* using two different MIDI instrument IDs (0 and 1, respectively). This effectively dedicates slices of the model's vocabulary (each with 128 tokens) to represent the notes of either instrument, enabling both musical roles to be modeled. We note that the vocabulary for the time, duration, and velocity tokens overlaps across both instruments.

The supervised finetuning of our model on this new task requires the collection of a new dataset. Through our collaboration with Jordan Rudess, we collect a dataset of around 3.5 hours of solo piano, manually segmented between *calls* and *responses*. In the dataset, Rudess presses on a MIDI foot switch (mapped to continuous control 1) when recording *calls* and releases it when recording subsequent *responses*. Figure 2 visualizes the structure of the training data. The training data is also recorded without any specified tempo, in order to avoid any future restriction when performing with the system.

In contrast to our previous approach, we effectively enforce the musical roles of *calls* and *responses* at *training time* instead of expecting the musical roles to emerge at performance time. Our approach is similar to the first step of the *instruction tuning* realized in natural language models [19], in which an expert labeler demonstrates a desired output behavior by crafting answers that are then used to fine-tune the model. In our case, Rudess acts as the expert, choreographing the musical roles of call and response at training time for the system to model effectively. At inference time, or during a performance, Rudess can record a musical prompt by pressing a foot switch, which stops the model playback. When the foot switch is released, the prompt is passed to the model and the playback restarts.

2.4 Latency Compensation

The JAM_BOT aims to generate future note sequences at a faster rate than they need to be played in real-time (RTF > 1). This creates an opportunity to mitigate the latency introduced by other parts of the system by scheduling MIDI messages to be played slightly earlier than their target time. In particular, this allows us to offset latencies introduced by MIDI output devices

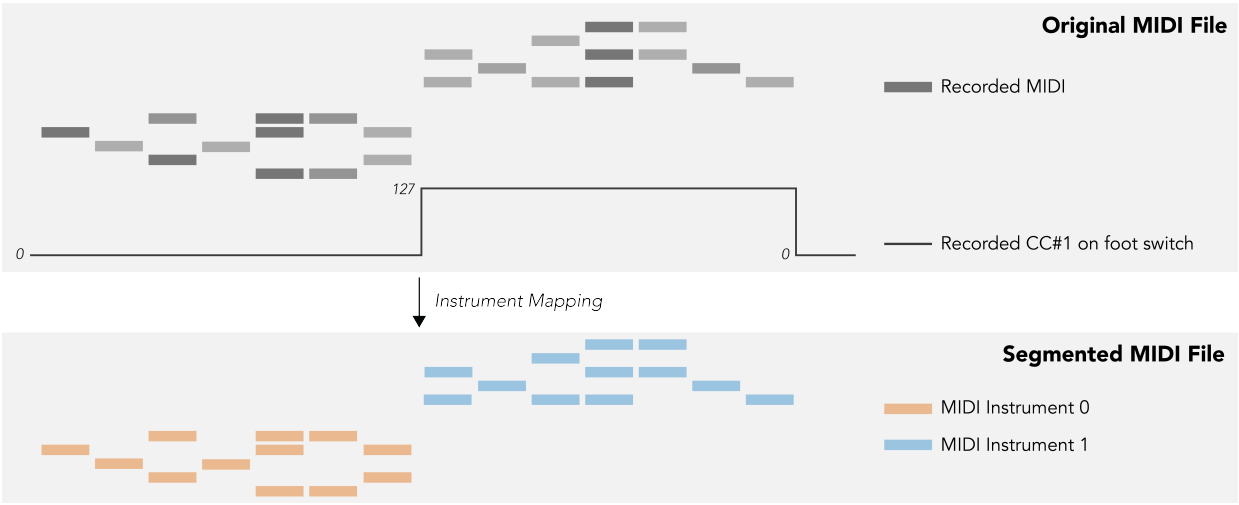

Figure 2: The annotation process of the *call and response* model. The recorded MIDI file is segmented into two instruments using the signal of Continuous Control 1, as recorded by a MIDI foot switch.

and network connections. Physically-actuated MIDI output devices, such as a player piano, require a lead-in time to play each note. When streaming MIDI data over the internet, the network latency can be measured and compensated. Compensating for MIDI output latency is especially important when the JAM_BOT outputs varying velocities. If a MIDI output device adds distinct latencies for each velocity, then the JAM_BOT's intended rhythm and note ordering may be misrepresented by the output instrument. This misrepresentation is exemplified in the Evaluation subsection 3.2.

We design two ways of offsetting the MIDI output latency of the system: (1) *globally* and (2) *individually* based on pitch and velocity. Global latency offsets are configured from a textbox in the interface. We found the global MIDI latency value reported by the DAW to be sufficient, though a more precise value could be measured using an external recorder. Per-note and -pitch offsets are loaded as a lookup table (LUT) from a comma-separated value (CSV) file. This LUT is selected from a dropdown menu to allow for simple switching of latency profiles for different output devices.

To create the CSV file for a MIDI output device, we record and measure the offsets using MIDI and audio. First, we prepare a MIDI file containing each note of the instrument at increasing velocity steps. Each note is played sequentially at an interval to allow each note to near entirely decay. We play back this MIDI file in a digital audio workstation (DAW) while simultaneously recording the MIDI output device's synthesis or actuation. Using a Python script, the note onset times in the recorded audio are analyzed by *librosa* and compared against the target note times in the original MIDI file [17]. We additionally account for the audio interface sample latency time reported by the DAW. For physical instruments with distinct sound source locations for each note, we record with multiple microphones, offset for the physical distance between the microphones and notes, and use the earliest onset reported by *librosa* across the microphone array. We write the note, velocity, and millisecond offset times to a CSV file. This method does not account for latency introduced by the DAW or MIDI interfaces, but in practice, these specific latencies did not contribute to a perceivable overall latency.

3 Evaluation

In order to evaluate our system, we focus on two areas for which we could retrieve quantitative metrics: the throughput of our model and the latency compensation. The velocity and call and response modality can be evaluated by the reader by listening to samples on the accompanying website.[2]

3.1 Evaluating Throughput

We evaluate the throughput of our model by measuring the number of tokens per second the model can generate during a performance setting. We measure this by keeping a running average of the number of tokens generated every second and waiting for 15 seconds before reporting the value. The number of tokens per second decreases with time as the context window fills up and plateaus once it is filled. We use a maximum context window of 120 tokens for the model, corresponding to 40 notes without velocity, and 30 notes with velocity. We also report the latency of the model, defined as the time it takes for the model to generate a single token, in milliseconds, which we calculate by multiplying the inverse of the number of tokens per second by 1000.

We compare the performance difference between our previous *ONNX Runtime* implementation and our new *ggml* implementation, and run the models using CPU (AMD Ryzen 9 7950X), CUDA GPU (NVIDIA RTX 4090), and Apple Metal (M3 Max MacBook Pro) backends. We test the performance of a model finetuned on a `small` pre-trained AMT model (with a parameter count of 170M parameters) and another model finetuned on a `medium` pre-trained AMT model (with a parameter count of 416M parameters). We report our results in Tables 1 and 2. We observe that the use of *ggml* improves the model throughput by a factor of 2 for both model sizes when using CUDA. *ggml* also improves model throughput when using CPU, to a lesser extent. We note that *ggml* also allows us to run the models on Apple accelerated AI hardware through an implementation of operators using the Metal framework, opening the door for wider adoption of the system.

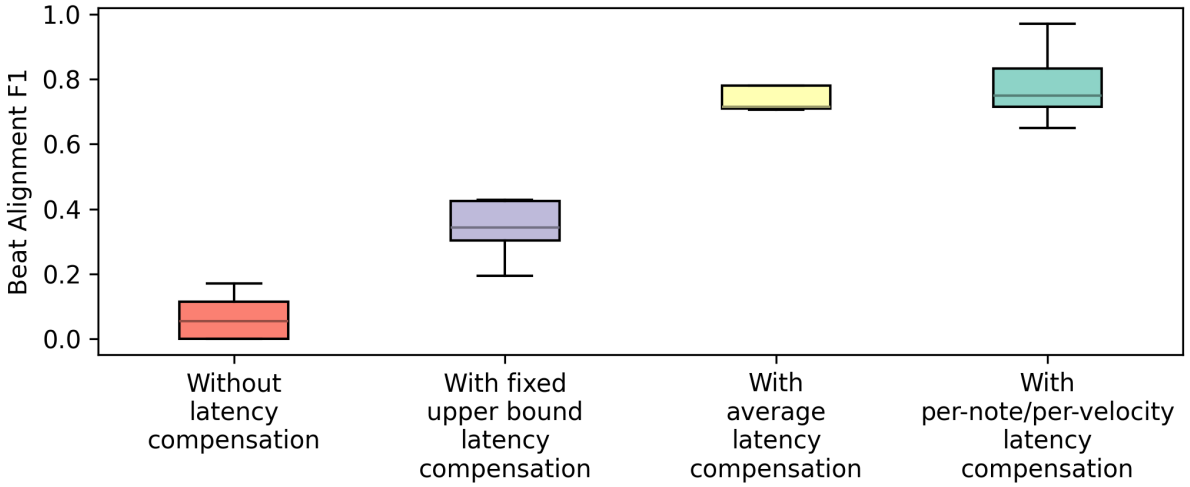

Figure 3: Beat alignment with and without latency compensation, as measured by the *madmom* F_1 score

Table 1: 170M parameter model throughput

Framework	Backend	Latency (ms) ↓	Tokens/second ↑
ONNX	CUDA	4.00	249.75
ggml	**CUDA**	**2.00**	**499.50**
ggml	Metal	10.13	98.76
ONNX	CPU	54.83	18.24
ggml	**CPU**	**33.45**	**29.90**

CUDA: NVIDIA RTX 4090; CPU: AMD Ryzen 9 7950X; Metal: M3 Max MacBook Pro

Table 2: 416M parameter model throughput

Framework	Backend	Latency (ms) ↓	Tokens/second ↑
ONNX	CUDA	8.01	124.84
ggml	**CUDA**	**4.04**	**247.65**
ggml	Metal	26.17	38.21
ONNX	CPU	118.55	8.44
ggml	**CPU**	**100.17**	**9.98**

CUDA: NVIDIA RTX 4090; CPU: AMD Ryzen 9 7950X; Metal: M3 Max MacBook Pro

3.2 Evaluating Latency Compensation

To evaluate the benefits of our latency compensation method, we use the first minute of a MIDI transcription of Beethoven's *Für Elise* and play it back on a player piano. To test different velocities, we artificially modify the velocity of each note following a sine wave with a period of 4 seconds, oscillating between values 20 and 70. We then play the MIDI file back on the player piano using 4 strategies: (1) without any latency compensation; (2) by using a fixed upper bound latency measurement; (3) by using the average latency; and (4) by using an individual latency for each note and velocity (as described in 2.4). We record the resulting audio and compare it with an audio file of the synthesized MIDI using a virtual piano. We estimate the beats for each file by using the beat estimation model *Beat This!* [12] and by reporting the F_1 score using *madmom* [7], measuring the alignment of the audio recording to the reference synthesized audio. We measure the beat alignment for 5 non-overlapping windows and report the results in Figure 3. We observe an almost complete lack of alignment when not using latency compensation, with an F_1 score of 0.0678 across all 5 samples, while the per-note/per-velocity latency compensation helps reach a median F_1 score of 0.7834. Using the fixed upper bound latency results in a median F_1 score of 0.3386, while using the average latency results in a median F_1 score of 0.7638. The measured latency varies greatly across notes and velocities, which explains why using the fixed upper bound latency strongly impacts the alignment, while using an average gives a close estimate of the actual latency.

[2]https://jambot.media.mit.edu/nime2026

4 Future Work

We aim to implement even more expressivity in our models, such as sustain pedal, pitch bend, and other expressive controls. Modeling these controls properly, however, requires precise tokenization. In particular, expressive controls can have very different densities than notes–pitch bend messages are much more dense, while sustain is much less dense. This proves challenging since a high density of controls can hinder the real-time performance of the model, while a sparse control can be difficult to model due to long dependencies extending outside of the context window.

Additionally, while we have already experimented with diverse machine learning frameworks and optimization techniques, we believe that there are more methods, such as model distillation or more advanced quantization methods, that could continue to improve the model throughput.

Finally, while using training data crafted by a single artist improves the output coherence and provides strong performance guarantees, it limits reuse by other performers and applications to different styles. Making the JAM_BOT more generalizable will require collecting more diverse data and developing fine-tuning strategies that broaden stylistic range without sacrificing the precise stylistic adherence that overfit models can provide.

5 Conclusion

In this work, we presented four improvements to the JAM_BOT that remove some previous constraints that limited the musical conversation capabilities of the system. In particular, we modeled velocity as a way to improve the expressivity of the model, we improved the model throughput by using the *ggml* framework, we designed a call-and-response modality for a more controlled and intuitive interaction, and we developed a method to offset MIDI output latencies. Through these improvements, we enhanced the capabilities of the JAM_BOT, making it an expressive, responsive, powerful, and easily reproducible interface for real-time interaction with music language models. We hope that this work can inspire future applications, such as live performances and installations.

Acknowledgments

This work was made possible through an Innovation Project Grant by the MIT Generative AI Impact Consortium (MGAIC), as well as the support of Jordan and Danielle Rudess, and the MIT Center for Art, Science, and Technology (CAST).

6 Ethical Standards

There is no observed conflict of interest. The research was funded using university funding and used university-owned compute power for the training of the models. Consent by Jordan Rudess was obtained before training the models and distributing the samples.

References

[1] Misagh Azimi and Mo H. Zareei. 2025. Live Improvisation with Fine-Tuned Generative AI: A Musical Metacreation Approach. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Zenodo, 389–393. https://doi.org/10.5281/zenodo.15698902

[2] Oded Ben-Tal and David Dolan. 2023. Musical and Meta-Musical Conversations. *AIMC 2023* (Aug. 2023). https://aimc2023.pubpub.org/pub/c6dv22xq/release/2

[3] Christodoulos Benetatos, Joseph VanderStel, and Zhiyao Duan. 2020. BachDuet: A Deep Learning System for Human-Machine Counterpoint Improvisation. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Romain Michon and Franziska Schroeder (Eds.). Birmingham City University, 635–640. https://doi.org/10.5281/zenodo.4813234

[4] John A Biles. 1994. GenJam: A Genetic Algorithm for Generating Jazz Solos. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, 131–137.

[5] Lancelot Blanchard, Perry Naseck, Stephen Brade, Kimaya Lecamwasam, Jordan Rudess, Cheng-Zhi Anna Huang, and Joseph Paradiso. 2025. The jam_bot, a Real-Time System for Collaborative Free Improvisation with Music Language Models. In *Proceedings of the 26th International Society for Music Information Retrieval Conference*. ISMIR, 769–776. https://doi.org/10.5281/zenodo.17811478

[6] Stephen Brade, Teng Ma, Lancelot Blanchard, Kimaya Lecamwasam, Carlos Mariano Salcedo, Suwan Kim, Perry Naseck, Andrew Li, Matthew R. Michalek, Sebastian Franjou, and Anna Huang. 2026. Agents in Concert: A Case-Study of Bringing AI to the Stage in Practice. In *Proceedings of the 31st International Conference on Intelligent User Interfaces (IUI '26)*. ACM, Paphos, Cyprus, 1–22.

[7] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. 2016. madmom: a new Python Audio and Music Signal Processing Library. In *Proceedings of the 24th ACM International Conference on Multimedia*. Amsterdam, The Netherlands, 1174–1178. https://doi.org/10.1145/2964284.2973795

[8] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W. Cottrell, and Julian J. McAuley. 2019. LakhNES: Improving Multi-instrumental Music Generation with Cross-domain Pre-training. In *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands, November 4-8, 2019*, Arthur Flexer, Geoffroy Peeters, Julián Urbano, and Anja Volk (Eds.). 685–692. http://archives.ismir.net/ismir2019/paper/000083.pdf

[9] Chris Donahue, Ian Simon, and Sander Dieleman. 2019. Piano Genie. In *Proceedings of the 24th International Conference on Intelligent User Interfaces (IUI '19)*. Association for Computing Machinery, New York, NY, USA, 160–164. https://doi.org/10.1145/3301275.3302288

[10] Drew Edwards, Simon Dixon, and Emmanouil Benetos. 2023. PiJAMA: Piano Jazz with Automatic MIDI Annotations. *Transactions of the International Society for Music Information Retrieval* 6, 1 (Sept. 2023), 89–102. https://doi.org/10.5334/tismir.162

[11] Nicholas Evans, Behzad Haki, and Sergi Jordà. 2025. Repurposing a Rhythm Accompaniment System for Pipe Organ Performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Zenodo, 116–120. https://doi.org/10.5281/zenodo.15698807

[12] Francesco Foscarin, Jan Schlüter, and Gerhard Widmer. 2024. Beat this! Accurate beat tracking without DBN postprocessing. In *Proceedings of the 25th International Society for Music Information Retrieval Conference (ISMIR)*. San Francisco, CA, United States.

[13] ggml.ai. 2026. *ggml*. https://ggml.ai/

[14] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2018. Music Transformer. https://doi.org/10.48550/arXiv.1809.04281 arXiv:1809.04281 [cs, eess, stat].

[15] Kıvanç Tatar and Philippe Pasquier. 2019. Musical agents: A typology and state of the art towards musical metacreation. *Journal of New Music Research* 48, 1 (2019), 56–105.

[16] Ari Liloia and Roger Dannenberg. 2025. Exploiting Latency In The Design Of A Networked Music Performance System For Percussive Collective Improvisation. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Zenodo, 473–480. https://doi.org/10.5281/zenodo.15698934

[17] Brian McFee, Matt McVicar, Daniel Faronbi, Iran Roman, Matan Gover, Stefan Balke, Scott Seyfarth, Ayoub Malek, Colin Raffel, Vincent Lostanlen, Benjamin van Niekirk, Dana Lee, Frank Cwitkowitz, Frank Zalkow, Oriol Nieto, Dan Ellis, Jack Mason, Kyungyun Lee, Bea Steers, Emily Halvachs, Carl Thomé, Fabian Robert-Stöter, Rachel Bittner, Ziyao Wei, Adam Weiss, Eric Battenberg, Keunwoo Choi, Ryuichi Yamamoto, CJ Carr, Alex Metsai, Stefan Sullivan, Pius Friesch, Asmitha Krishnakumar, Shunsuke Hidaka, Steve Kowalik, Fabian Keller, Dan Mazur, Alexandre Chabot-Leclerc, Curtis Hawthorne, Chandrashekhar Ramaprasad, Myungchul Keum, Juanita Gomez, Will Monroe, Viktor Andreevitch Morozov, Kian Eliasi, nullmightybofo, Paul Biberstein, N. Dorukhan Sergin, Romain Hennequin, Rimvydas Naktinis, beantowel, Taewoon Kim, Jon Petter Åsen, Joon Lim, Alex Malins, Darío Hereñú, Stef van der Struijk, Lorenz Nickel, Jackie Wu, Zhen Wang, Tim Gates, Matt Vollrath, Andy Sarroff, Xiao-Ming, Alastair Porter, Seth Kranzler, Voodoohop, Mattia Di Gangi, Helmi Jinoz, Connor Guerrero, Abduttayyeb Mazhar, toddrme2178, Zvi Baratz, Anton Kostin, Xinlu Zhuang, Cash TingHin Lo, Pavel Campr, Eric Semeniuc, Monsij Biswal, Shayenne Moura, Paul Brossier, Hojin Lee, Waldir Pimenta, Jon Petter Åsen, Shin Hyun, Iliya S, Eugene Rabinovich, Geo Lei, Jize Guo, Phillip S.M. Skelton, Matt Pitkin, Anmol Mishra, Slava Chaunin, BenedictSt, Scott VanRavenswaay, and David Südholt. 2025. *librosa/librosa: 0.11.0*. https://doi.org/10.5281/zenodo.15006942

[18] Microsoft. 2025. *ONNX Runtime*. https://onnxruntime.ai/

[19] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. https://arxiv.org/abs/2203.02155 _eprint: 2203.02155.

[20] François Pachet. 2003. The Continuator: Musical Interaction With Style. *Journal of New Music Research* 32, 3 (Sept. 2003), 333–341. https://doi.org/10.1076/jnmr.32.3.333.16861

[21] Colin Raffel. 2016. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD Thesis.

[22] R. Keith Sawyer. 2005. Music and conversation. In *Musical Communication*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780198529361.003.0003 _eprint: https://academic.oup.com/book/0/chapter/149286979/chapter-ag-pdf/44987242/book_5956_section_149286979.ag.pdf.

[23] Ian Simon and Sageev Oore. 2017. Performance RNN: Generating Music with Expressive Timing and Dynamics. https://magenta.withgoogle.com/performance-rnn Publication Title: Magenta Blog Type: Blog.

[24] Lyria Team, Antoine Caillon, Brian McWilliams, Cassie Tarakajian, Ian Simon, Ilaria Manco, Jesse Engel, Noah Constant, Yunpeng Li, Timo I. Denk, Alberto Lalama, Andrea Agostinelli, Cheng-Zhi Anna Huang, Ethan Manilow, George Brower, Hakan Erdogan, Heidi Lei, Itai Rolnick, Ivan Grishchenko, Manu Orsini, Matej Kastelic, Mauricio Zuluaga, Mauro Verzetti, Michael Dooley, Ondrej Skopek, Rafael Ferrer, Savvas Petridis, Zalán Borsos, Äaron van den Oord, Douglas Eck, Eli Collins, Jason Baldridge, Tom Hume, Chris Donahue, Kehang Han, and Adam Roberts. 2025. Live Music Models. https://arxiv.org/abs/2508.04651 _eprint: 2508.04651.

[25] John Thickstun, David Hall, Chris Donahue, and Percy Liang. 2024. Anticipatory Music Transformer. arXiv:2306.08620 [cs.SD] https://arxiv.org/abs/2306.08620