JIM 2026 : EXPRESSIVE MUSICAL PERFORMANCE CONTROLLED BY VOCAL GESTURES

RÉSUMÉ

L'objectif de ce projet est de permettre à des non-musiciens d'interpréter une partition sur un instrument à vent virtuel, sans apprentissage instrumental. Inspiré du concept de MetaPiano, qui garantit la justesse des notes tout en laissant le contrôle des paramètres expressifs, le système transpose cette approche aux instruments à vent.

Les utilisateurs produisent dans un microphone des gestes vocaux en combinant voyelles et consonnes. Ces gestes sont ensuite associés aux articulations musicales : une consonne suivie d'une voyelle déclenche une note détachée, tandis que deux voyelles consécutives produisent une note liée. L'intensité du souffle contrôle la dynamique, la vitesse de transition influence le rythme et la voyelle choisie module le timbre.

Le signal audio est analysé en temps réel pour identifier les phonèmes et déclencher les notes d'une partition préalablement sélectionnée. Chaque nouveau geste vocal fait progresser la partition, ce qui garantit la justesse des notes tout en laissant à l'utilisateur la liberté de l'interprétation expressive. Cette approche permet ainsi de contrôler le rythme, l'articulation, la dynamique et le timbre, en offrant une interface à la fois accessible et expressive pour l'interprétation, sans nécessiter de maîtrise technique préalable.

1. INTRODUCTION

Recent research in computer music has explored new forms of expressive performance that reduce technical barriers while preserving musical nuance. Within this context, the SCRIME laboratory (Studio for Creation and Research in Computer Science and Experimental Music) has developed interactive meta-instruments designed to facilitate musical interpretation through intuitive approaches.

One such system, *MetaPiano*, allows performers to generate the correct notes of a pre-selected score by pressing any key on a keyboard, while still allowing control over rhythm, tempo, and dynamics [1]. By separating pitch accuracy from the performer's gestures, the instrument lets users focus on musical expression rather than technical skill.

Beyond reducing technical barriers, this type of system can also improve musical accessibility for users with disabilities. By limiting the number of required keys and simplifying the gesture-to-sound mapping, such instruments may enable performers with reduced motor abilities to experience musical performance in an expressive way.

Building on this concept, the web-based application *MidifilePerformer* [3], [4] enables the interpretation of musical scores encoded as MIDI files, preserving expressive flexibility while ensuring the correct pitch of the notes.

Other research has also investigated "simplified instruments" or meta-instruments. For instance, *Piano Genie*, developed by the Magenta team at Google [11], enables users to freely improvise on the piano using only eight keys, relying on a learned neural mapping to generate plausible piano performances in real time. The system has also been made available as a web-based interactive platform.

The present work builds upon these meta-instruments approach and proposes a system adapted to wind instrument performance. Instead of controlling a keyboard, users interact through vocal gestures captured by a microphone, allowing expressive interpretation of a score without needing to master the instrument.

This project is set within that context, with the objective of designing an extension of the *MetaPiano* and *MidifilePerformer* adapted for wind instruments, thus continuing research on expressive musical performance freed from the technical constraints of traditional playing.

The proposed tool is designed to allow users to control musical sound through unvoiced onomatopoeia, captured via a microphone. When the user produces an onomatopoeia, the system detects it and generates a corresponding Musical Instrument Digital Interface (MIDI) message. This message is then used to produce the desired sound, effectively translating the performer's vocal gestures into musical articulation.

To facilitate the recognition of onomatopoeia, users can train the tool by pronouncing a set of onomatopoeia, allowing the system to learn and associate them with the correct MIDI events. Alternatively, pre-recorded training sets can be used. An "onomatopoeia tuner" interface allows users to verify and adjust their pronunciation so that it matches the reference examples, ensuring reliable control of the musical output.

This approach enables an intuitive, expressive interaction with virtual wind instruments, linking vocal gestures directly to musical performance.

2. MOTIVATION AND OBJECTIVE

The objective of this project is to enable non-musicians, or individuals physically unable to play a wind instrument, to experience the act of interpreting a musical score.

The user activates notes by whispering onomatopoeia (for example, “ta”) into a microphone. Pitch accuracy is automatically ensured by the pre-selected score, meaning that the user does not need to read music or have a trained ear.

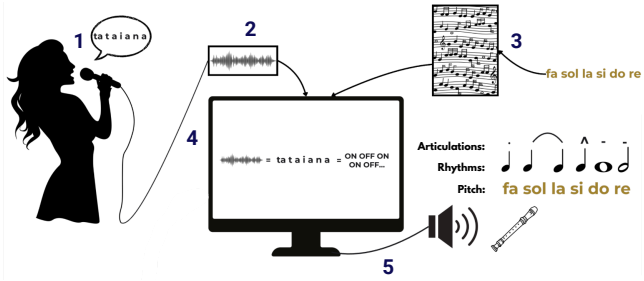

Figure 1. Prototype diagram of the onomatopoeia-to-sound system.

As shown in Figure 1, the system operates as follows :

1. Performer input : A person produces unvoiced onomatopoeic sounds, which can indicate fast or slow rhythms depending on their duration. Stronger airflow indicates a higher intended dynamic.

2. Audio signal capture : The sound of the onomatopoeia is recorded and processed in real time.

3. Music score input : The user provides a musical score in *MusicXML* format. Notes to be played are extracted and encoded as MIDI messages.

4. Onomatopoeia interpretation : An algorithm detects the pronounced onomatopoeia from the audio signal and translates them into "ON" and "OFF" events. These events activate and stop the corresponding notes at the pitches specified in the selected score.

5. Sound output : The final output reproduces the chosen rhythm, articulation, and instrument sound (e.g., flute), while respecting the pitch of the score. The system also allows real-time score following via the *MidifilePerformer* and provides feedback on pronounced onomatopoeia using a phoneme tuner.

Key considerations include recognizing unvoiced onomatopoeia so that the voice does not overpower the instrument sound, and minimizing latency. Ideally, the delay between the pronunciation of an onomatopoeia and the produced instrument sound is around 30 ms.

Videos demonstrating the system in use are available online [1]

3. SYSTEM DESIGN AND IMPLEMENTATION

In this section, we present the different parameters that were used to create a tool that approximates as closely as possible the performance techniques of wind instruments.

1. https://anonymous.4open.science/r/Videos-0767/

3.1. Capturing Vocal Gestures

The choice of gesture acquisition method is closely related to an understanding of the playing techniques specific to wind instruments. Analyzing these techniques makes it possible to identify the relevant expressive parameters, such as air pressure, articulation, and timbre, that can be exploited in the design of a capture and expressive control system.

Chatziioannou and Hofmann [2] provide a physical analysis of articulatory actions in single-reed instruments. Their study highlights the strong influence of performer gestures, particularly variations in air pressure and tongue movement, on sound production. In particular, they show that mouthpiece pressure evolves differently depending on whether notes are separated by an active tongue gesture or by a simple interruption of the airflow.

This analysis is extended by Chatziioannou, Schmutzhard, Pàmies-Vilà, and Hofmann [5], who developed a physical model and an artificial blowing machine capable of generating tonguing-type articulations. Their results confirm that, unlike the piano, where interpretation mainly relies on pressing and releasing keys, wind instrument performance primarily depends on airflow control and tongue interaction.

Two principal articulation types are modeled using unvoiced vocal gestures :

— **Staccato (detached)** articulation, characterized by a complete occlusion of the airflow by the tongue between notes.
— **Legato** articulation, in which the tongue does not fully interrupt the airflow, allowing continuous airflow between successive notes.

Beyond articulation, timbre modulation represents another essential expressive parameter in wind instruments. Pàmies-Vilà, Hofmann, and Chatziioannou [6] demonstrate that the configuration of the vocal tract significantly influences attack characteristics in clarinet playing, enabling performers to shape timbre according to musical intention. This suggests that vocal tract adjustments constitute a meaningful control dimension for expressive performance modeling.

An initial prototype mouthpiece was designed to measure intraoral and external pressure in order to detect tongue gestures and timbral variations. However, this solution appeared too complex. It was therefore replaced by a simplified approach offering equivalent expressive information through audio signal analysis.

Indeed, previous studies (notably Pàmies-Vilà et al. [6] and Auvray et al. [7]) have shown that articulation and timbre variations can be deduced directly from the harmonic content of the acoustic signal. Auvray et al. observed that changing the vocal tract during a note changes its timbre. These findings support the use of microphone-based signal analysis as a efficient sensing strategy for expressive wind performance control.

3.2. Generating Articulations from Vocal Input

As described in Section 3.1, two types of articulation are considered : detached and legato. These articulations are the ones we want to preserve and reproduce.

To achieve this, the objective is to imitate the movements of wind players through the microphone input in order to obtain a similar sound result.

3.2.1. Extraction of onomatopoeia from sound

The system can be trained in two ways : performers may record their own unvoiced onomatopoeia to train the tool, or a pre-recorded training dataset can be used. The goal is to link the sound of each onomatopoeia to a phoneme label, which motivates the use of a classification algorithm.

The first step is to capture the sound of the onomatopoeia produced by the performer. From this audio, 13 Mel-Frequency Cepstral Coefficients (MFCCs) are extracted using the `Python` library `librosa` ([8]), along with the zero-crossing rate and signal slope, providing a spectral and temporal representation of the sound.

For classification, a K-nearest neighbors (KNN) algorithm is employed using the `scikit-learn` `Python` library [9]. This choice is motivated by its simplicity and effectiveness for small, well-labeled datasets. When a new onomatopoeia is detected, its feature vector is compared to the labeled training data, and the majority label among its k nearest neighbors determines the recognized phoneme.

To improve accuracy and reduce dimensionality, the feature vectors are projected using Principal Component Analysis (PCA) using the `scikit-learn` `Python` library. This concentrates the most discriminative information in a smaller number of dimensions, facilitating faster computation and reducing noise in the classification. The optimal number of dimensions is selected by evaluating clustering quality using the silhouette score provided by the *scikit-learn* library, ensuring that phoneme classes are well-separated in the reduced.

Figure 2 illustrates the clusters formed by the labeled training phonemes. New inputs are projected into this space in real time, and their distances to these clusters are used to assign phoneme labels automatically.

3.2.2. Mapping onomatopoeia to articulation

The objective is to generate the appropriate articulation from the received onomatopoeia. Each onomatopoeia corresponds to a single phoneme used by the system for activating the notes.

We developed an algorithm that links each onomatopoeia to the corresponding system action (3). A note is activated whenever an onomatopoeia is detected, which can be a consonant or a vowel. The sound lasts for the duration of the vowel and is stopped when the next onomatopoeia occurs.

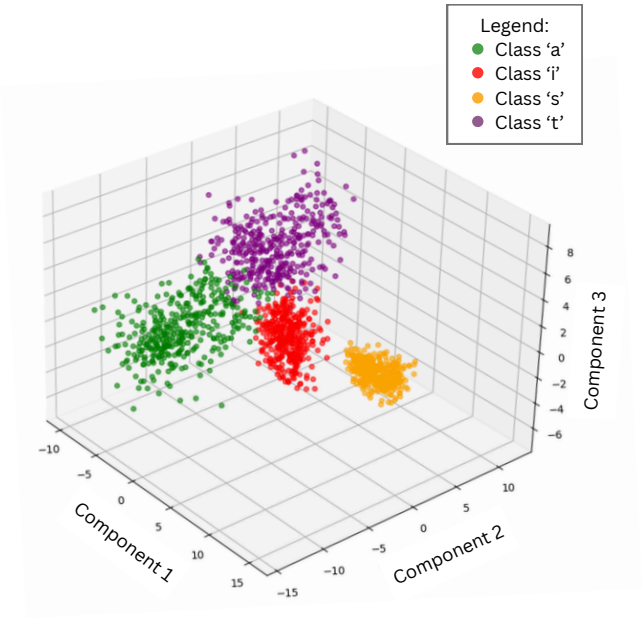

Figure 2. Projection of the training phonemes in the PCA-reduced space. Each color represents a label : green for [a], red for [i], purple for [t], and yellow for silence. Clusters show the separation used for real-time classification of incoming onomatopoeia.

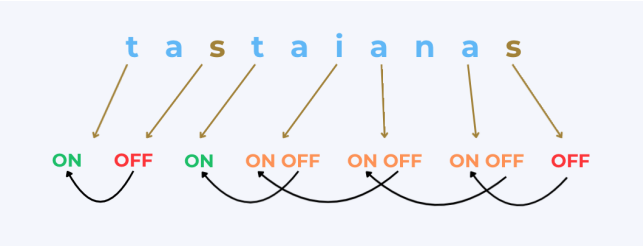

Figure 3. Description of the mapping between the onomatopoeia and the resulting articulation produced by the tool.

For the consonant [t], the airflow is stopped, and the sound is also stopped to create a detached articulation. When two vowels occur in a row, or when the letters [l] or [n] appear, the second note starts before the first one ends. This overlap simulates a continuous airflow, producing a legato articulation.

To allow easier changes in articulation, an automata, shown in Figure 4, was implemented in `Python`. For clarity, only the phonemes [l], [a], [t], and silence are displayed in this automata.

In the automata, transitions are shown with colored arrows for each phoneme, and states are represented by colored circles. The initial state allows transitions from all three phonemes and silence.

The "ON" (green) and "OFF" (red) states correspond to note activated and stopped : when the automata enters these states, it sends the associated MIDI messages.

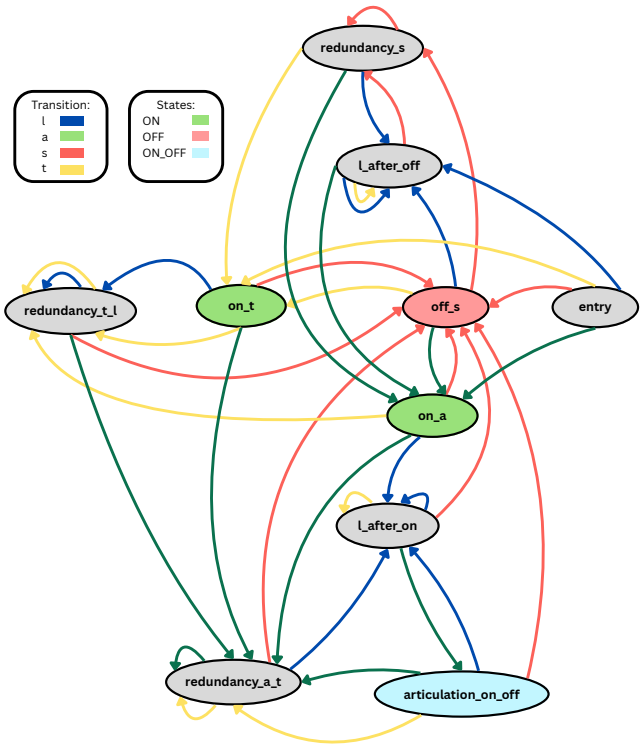

Figure 4. Minimal deterministic finite automata representing the types of articulation according to the phoneme types [a], [t], [l], and silence.

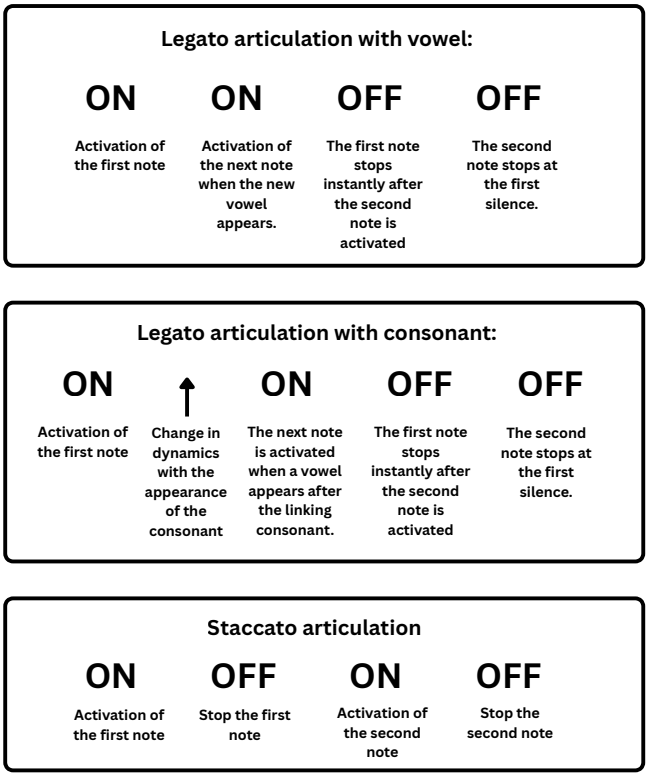

Figure 5. Differences between staccato and legato articulations with consonants or vowels.

It distinguishes between “ON” states activated by [t] or [a], since both can initiate a note.

The “ON OFF” state (blue) handles overlapping transitions by sending a MIDI Note "ON" for the next note immediately followed by a Note "OFF" for the current note. Redundant states indicate sustained vowels, keeping the automata in the same state.

States such as “l_after_on” and “l_after_off” differentiate whether a phoneme [a] should activate an articulation depending on the previous note state. The automata also enforces that [t] can only occur after silence, reflecting that airflow is fully blocked before [t] is pronounced. This design helps prevent articulation errors, such as a [t] appearing outside a silent context.

Figure 5 summarizes the different types of articulation between phonemes and how they correspond to staccato or legato articulations with consonants or vowels.

A legato articulation produces a perceptible sense of continuity, as two consecutive "ON" events leave no gap of silence between notes. Similarly, an articulation involving a consonant also allows a continuous transition, but the airflow is partially blocked by the tongue, resulting in a lower sound intensity between the two notes.

4. DYNAMIC AND TIMBRE MODULATION

Once an “ON” or “OFF” is generated, the corresponding MIDI message is transmitted.

We use four main types of MIDI messages :

— **Note On / Note Off :** activate and stop a note
— **Control Change (CC) :** modify parameters such as intensity or modulation
— **Program Change :** switch the sound or instrument on a MIDI channel
— **Pitch Bend :** change the pitch of a note

The algorithm is implemented to send a **Note On** message when the state corresponds to an "ON", and a **Note Off** message when it corresponds to an "OFF" state. The pitch of each note follows the MIDI score.

When the first "ON" occurs, the first note of the score is activated with a Note On. This note is stopped upon the next "OFF" state with a Note Off. Each "ON" start the next note in the score, and this process continues until all notes have been played.

4.1. Adaptation of wind blow

To enable expressive dynamic control during score performance, the Root Mean Square (RMS) of the audio signal is computed within the algorithm in `Python`.

The RMS allows estimating the average intensity of an audio signal over a given time window. For a signal $x[n]$ of length N, it is defined as :

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} x[n]^2} \quad (1)$$

This value reflects the strength or dynamic level of the

pronounced phonemes.

To produce smoother dynamic variations, the RMS value is transformed using a hyperbolic tangent, allowing the dynamics to gradually increase or decrease towards a plateau, rather than being abruptly limited :

$$\text{RMS}_{\text{tanh}} = \tanh(k \cdot \text{RMS}) \quad (2)$$

where k is a scaling factor. This transformation ensures a smoother progression of dynamic levels.

4.2. Timbre Control and Synthesis

The objective of using vowels as a performance parameter is also to reproduce the timbral variations produced by wind instrument players through modifications of their vocal tract.

In our system, each vowel is associated with the transmission of a *Control Change* (CC) MIDI message on a dedicated channel. For instance, when the vowel [i] is detected, a value of 127 is sent to a specific CC parameter, resulting in the selection of a particular timbre. In contrast, the vowel [a] is mapped to a different CC parameter, leading to a different timbral configuration.

To synthesize these MIDI messages as sound, software capable of translating MIDI data into audio signals can be used, such as Respiro [10].

Respiro enables the generation of wind instrument sounds that closely resemble those produced by physical instruments.

In this work, the software is used to generate the sound of the instrument selected by the user. It processes the incoming MIDI messages and produces the corresponding audio output. The instrument sound is activated upon receiving a *Note On* event and is stopped when a *Note Off* event is received.

Figure 6. Visualisation of the main page of software *Respiro*.

Figure 6 presents the main interface of Respiro. Instrument presets and timbre control macros (e.g., Brass Timbre) can be mapped to MIDI *Control Change* (CC) messages, allowing phonemes or breath intensity to modify the sound in real time. Breath sensitivity is controlled through a CC proportional to the RMS value of the detected signal.

Since MIDI CC values range from 0 to 127, the transformed RMS values are normalized by a maximum reference value obtained during a calibration phase, in which the user is asked to blow into the microphone at maximum intensity.

Although Respiro is used in this study, any MIDI compatible software could provide similar functionality.

5. SOFTWARE ARCHITECTURE

A Python-based algorithm was developed to simplify the system's use through a graphical user interface (GUI).

5.1. Phoneme Tuner

For users who wish to verify the base training or ensure that their custom training aligns with their pronunciation, a **phoneme tuner** is available.

The tuner functions similarly to a frequency tuner for musicians, displaying which phoneme was selected by the classifier and how close it is to the training phonemes. An algorithm calculates the distance of each phoneme from the centroid of its training label and provides a graphical representation. The tuner also shows the second-closest phoneme based on the number of neighboring points in the classification.

Figure 7 illustrates the phoneme tuner interface. In this example, the pronounced phoneme is [a], indicated by the red bar. The display also shows the symbol `s` for silence, representing the second-closest phoneme cluster. The slight leftward tilt of the bar suggests that [a] could be confused with silence if the whisper is too soft, but it is not confused with any other phoneme.

Below the tuner, the real-time phoneme sequence ("tatatatata") is displayed, allowing the user to monitor phonemes as they are pronounced.

Silence is included among the phonemes in the tuner display but is omitted from the real-time sequence to avoid confusion.

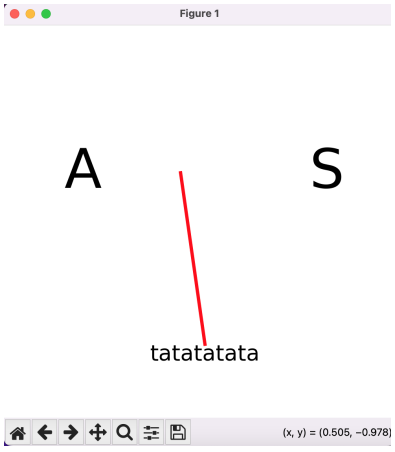

Figure 7. Phoneme tuner visualization showing the `[a]` phoneme and silence.

5.2. YAML Configuration File

Key algorithm parameters, including those for MFCC calculation can be adjusted by the user. A `.yaml` file

(YAML : Yet Another Markup Language) is used to define key–value pairs in a readable format. This allows the user to adapt the system to personal needs and optimize interpretation.

Almost all parameters, including MFCC extraction settings, KNN neighbors, RMS scaling, feature selection, and MIDI CC mapping, can be adjusted through the `.yaml` file, allowing users to customize the system without editing configuration files.

Users can also enable or disable auxiliary features such as *delta MFCC*, zero-crossing rate, spectral centroid, or signal slope. They also have full control over which phonemes to use for interpretation, and can configure the corresponding *Control Change* (CC) numbers for each selected vowel.

5.3. Graphical User Interface (GUI)

To simplify usage, a GUI was implemented in `Python`, allowing users to launch scripts and adjust parameters without editing the YAML manually.

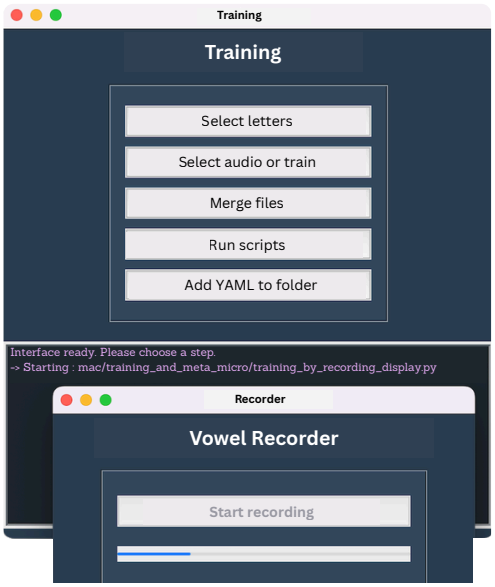

Figure 8. Training homepage showing five steps and the start of recording whispered phonemes.

The user first selects the operating system (Mac or Linux), then chooses between a new training session or using an existing system.

During training (Figure 8), users select a phoneme to record. For example, if they want to add a new phoneme such as [a], they can indicate this choice and record themselves pronouncing [a]. The system then learns this pronunciation, allowing it to recognize the phoneme in future inputs. Users can either capture new audio in this way or use pre-recorded audio with manually annotated markers. Recorded files can also be concatenated with previously stored phonemes to expand the training dataset.

The full script calculates MFCCs and performs classification, visualized with a progress bar (Figure 9).

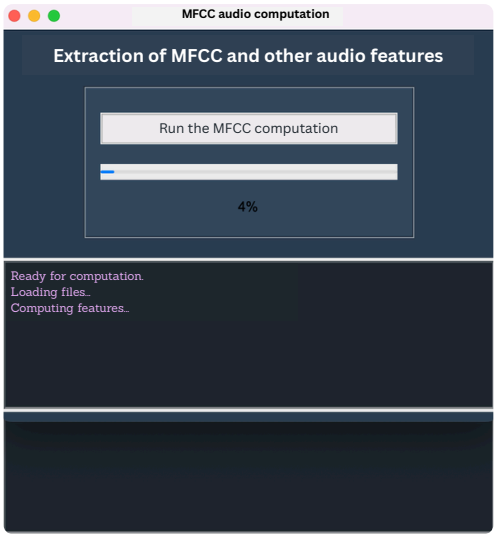

Figure 9. MFCC calculation with progress bar during processing.

All parameter adjustments and generated files are automatically saved in the YAML configuration and stored in a folder, facilitating later reuse.

For regular use, the user chooses the training folder containing the YAML configuration, modifies audio input/output settings or CC values, and optionally measures maximum breath. The system can then interpret music with or without the vowel tuner (Figures 10 and 11).

Additionally, the GUI can be integrated with the *MidifilePerformer* for real-time score following, allowing the system to activate notes in sync with the pre-selected musical score.

This GUI therefore simplifies script execution and parameter management, providing a user-friendly interface for both training and real-time musical interpretation.

6. EVALUATIONS

The tool was evaluated to measure its effectiveness and functionality. After several trials, the system demonstrated a strong responsiveness to intended musical interpretations. It reacts appropriately to tempo variations, including accelerations and decelerations, as well as to different articulations, producing the impression of continuous transitions between legato notes.

Dynamic control was particularly effective when adjusting the maximum RMS, allowing nuanced expressive variations. The phoneme tuner was also well-received, being recognized for its utility in facilitating user interaction and improving instrument control.

Regarding latency, the ideal 30ms delay is not always achieved, as it depends on algorithmic parameters such as the number of neighbors considered in the KNN classification. Nevertheless, the latency remains acceptable, generally around 50ms.

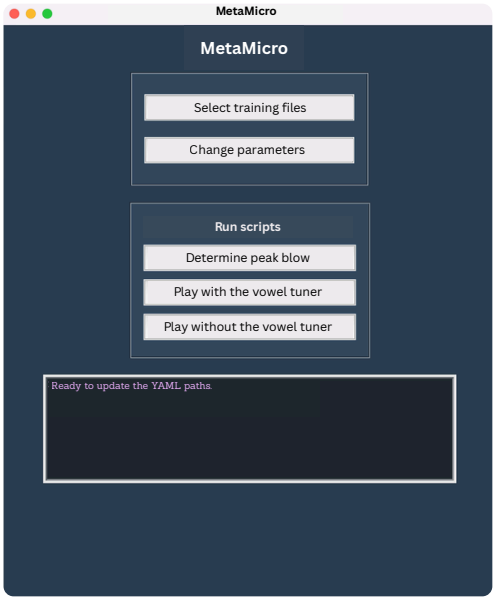

Figure 10. MetaMicro system homepage with interpretation options.

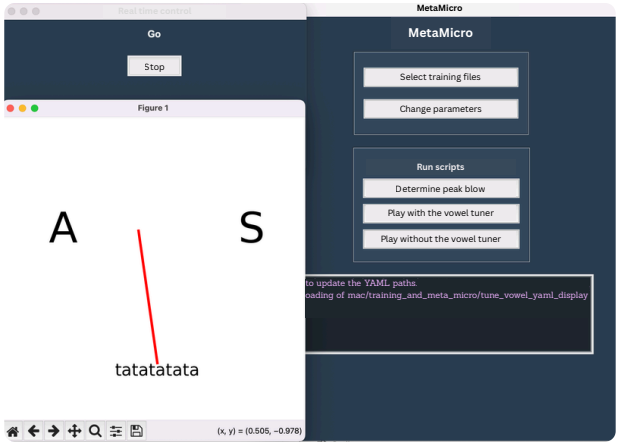

Figure 11. Phoneme tuner interface, referred to here as the *Vowel Tuner*.

7. CONCLUSION

This work presents a system enabling expressive musical performance through vocal gestures, specifically whispered onomatopoeia. By combining real-time audio capture, feature extraction, and KNN-based classification with PCA dimensionality reduction, the system successfully converts vocal inputs to musical articulations and dynamic variations. The integration of a graphical user interface and a phoneme tuner facilitates user training and ensures accurate control over the resulting sound, while MIDI-based synthesis allows flexible sound rendering.

Evaluations have shown that the system is responsive to tempo changes, articulation types, and dynamic nuances, providing an intuitive and expressive experience for both trained musicians and non-specialists. Some latency remains, which could be further reduced by exploring alternative classification methods or more optimized signal processing techniques.

As a future direction, it would be interesting to combine this vocal interface with the *MetaPiano* system, enabling performers to separately control melody and accompaniment, and allowing more complex musical interactions.

8. REFERENCES

[1] *SCRIME, Performance et méta-instruments*. https://scrime.u-bordeaux.fr/projets-de-recherche/performance-et-meta-instruments

[2] Vasileios Chatziioannou et Alex Hofmann. *Physics-based analysis of articulatory player actions in single-reed woodwind instruments*. Acta Acustica united with Acustica, vol. 101, no. 2, pp. 292–299, 2015. DOI : https://doi.org/10.3813/AAA.918827

[3] Juliette Chabassier, Myriam Desainte-Catherine, Jean Haury, Marin Pobel, and Bernard P. Serpette. *MidifilePerformer : a case study for chronologies*. Proceedings of the 26th ACM SIGPLAN International Conference on Functional Programming (ICFP '21), Aug. 2021, Online (South Korea), pp. 13–22. DOI : https://doi.org/10.1145/3471872.3472968. HAL : https://hal.science/hal-03578817

[4] Joseph Larralde, Bernard Paul Serpette, Jean Haury, Myriam Desainte-Catherine, and Raphaël Blard. *Élargir l'accès à l'interprétation musicale avec Web Midifile Performer*. Journées d'Informatique Musicale (JIM), PRISM, May 2024, Marseille, France, p. 38. HAL : https://hal.science/hal-04870929

[5] Vasileios Chatziioannou, Sebastian Schmutzhard, Montserrat Pàmies-Vilà et Alex Hofmann. *Investigating clarinet articulation using a physical model and an artificial blowing machine*. Acta Acustica united with Acustica, vol. 105, no. 4, pp. 682–694, 2019. DOI : https://doi.org/10.3813/AAA.919348

[6] Montserrat Pàmies-Vilà, Alex Hofmann et Vasileios Chatziioannou. *The influence of the vocal tract on the attack transients in clarinet playing*. Journal of New Music Research, vol. 49, no. 2, pp. 126–135, 2020. DOI : https://doi.org/10.1080/09298215.2019.1708412

[7] R. Auvray, A. Ernoult, S. Terrien, B. Fabre & C. Vergez. *Effect of Changing the Vocal Tract Shape on the Sound Production of the Recorder : An Experimental and Theoretical Study*. Acta Acustica united with Acustica, vol. 101, no. 2, pp. 317–330, 2015. DOI : 10.3813/AAA.918829

[8] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto.

librosa : Audio and music signal analysis in Python. In Proceedings of the 14th Python in Science Conference, pp. 18–25, 2015.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. *Scikit-learn : Machine Learning in Python*. Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[10] THOMANN. *Respiro – Guide utilisateur*. 2024. PDF. Available on : https://images.thomann.de/pics/atg/atgdata/document/installation/569466_respirodownloadinstructions.pdf

[11] Chris Donahue, Ian Simon, and Sander Dieleman, *Piano Genie*, arXiv :1810.05246v2 [cs.LG], 2019. DOI : https://doi.org/10.48550/arXiv.1810.05246.