Real-Time Application of Time Scale Modification (TSM) Algorithms

Sayema Lubis
Harvey Mudd College
Claremont, CA USA
slubis@hmc.edu

Jared Carreño
Harvey Mudd College
Claremont, CA USA
jcarreno@hmc.edu

TJ Tsai
Harvey Mudd College
Claremont, CA USA
ttsai@hmc.edu

***Abstract*—This paper presents a real-time implementation of classical time-scale modification (TSM) algorithms that preserve audio quality while allowing users to dynamically adjust the time-stretch factor without affecting pitch. By combining the Overlap-Add (OLA), Phase Vocoder (PV), and harmonic-percussive separation (HPS) algorithms, we exploit the strengths of each method—OLA for preserving percussive transients and PV for maintaining harmonic coherence. We address the challenges of buffer management, synchronization, and low-latency processing in a real-time system. Runtime analysis confirms that our hybrid approach meets real-time constraints across various input lengths and stretch factors, making it suitable for practical applications such as interactive audio playback and live performance tools.**

***Index Terms*—overlap-add, phase vocoder, time-scale modification, harmonic percussive separation.**

I. Introduction

This paper explores an application in which a user can manipulate the speed of an audio recording in real time without modifying its original pitch. Time-scale modification (TSM) is the process of manipulating an audio signal to either speed up or slow down the signal from its original tempo without changing its original pitch. TSM is particularly useful in offline music-based applications, such as music production and audio editing.

There exist many different TSM algorithms, each with its own focus and specific application; however, none are as well known as overlap-add (OLA) and phase vocoder (PV) methods. While OLA preserves transients, PV maintains phase coherence across frames, which makes it better suited for harmonic signals [1]. These disparities can be overcome by combining multiple TSM algorithms that can handle the respective percussive and harmonic elements of the signal.

Using *harmonic percussive separation* (HPS), the input signal is split into its harmonic and percussive elements that can be individually processed by the appropriate algorithm. The combination of algorithms produces a new *hybrid* method that is capable of handling harmonic and percussive information. This hybrid implementation creates an output by superimposing the stretched signals produced individually by the algorithms being used in parallel, as shown in Figure X.

However, developing a true real-time implementation of these TSM algorithms poses several critical challenges beyond their offline counterparts. First is the requirement for low-latency processing while maintaining audio quality, particularly when users dynamically adjust the time stretch factor α during operation. This real-time parameter control demands efficient buffer management and on-the-fly algorithm adaptation without introducing audible artifacts or latency. Second, the system must handle continuous audio streaming while performing time-scale modification, creating unique constraints on memory usage and computational efficiency. These challenges motivate the investigation into optimized real-time variants of both OLA and phase vocoder approaches, as well a hybrid solution that dynamically balances these challenges.

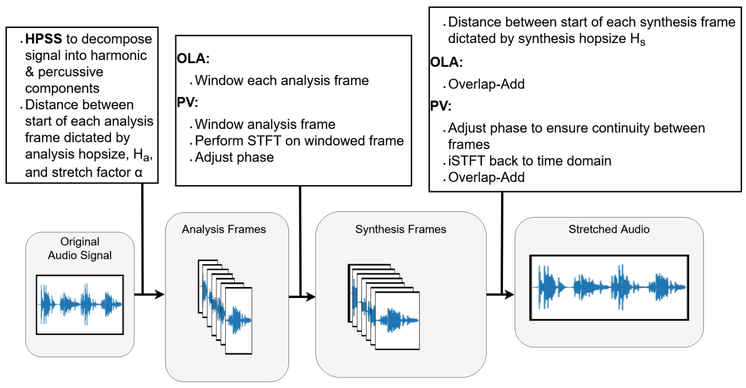
. **HPSS** to decompose signal into harmonic & percussive components
. Distance between start of each analysis frame dictated by analysis hopsize, H$_a$, and stretch factor α
OLA:
. Window each analysis frame
PV:
. Window analysis frame
. Perform STFT on windowed frame
. Adjust phase
. Distance between start of each synthesis frame dictated by synthesis hopsize H$_s$
OLA:
. Overlap-Add
PV:
. Adjust phase to ensure continuity between frames
. iSTFT back to time domain
. Overlap-Add
Original Audio Signal
Analysis Frames
Synthesis Frames
Stretched Audio

Fig. 1. Generalized TSM processing chain. Relationship between H_a, α, and H_s described by Equation 2.

II. Time-Scale Modification Algorithms

A. *Overlap-Add (OLA)*

The overlap-add or OLA algorithm is arguably the most intuitive among the various time-stretching algorithms. Its fundamental approach involves segmenting the input audio signal into overlapping windows, referred to as **analysis windows** $x_m[n]$. The window is characterized by length L and **analysis hopsize** H_a as illustrated in Figure 1.

In OLA, each analysis frame $x_m[n]$ is multiplied by a window function – commonly the Hann window $w[n]$ – to reduce boundary discontinuities. The resulting windowed frames are referred to **synthesis windows**.

$$y_m[n] = \frac{w[n] \cdot x_m[n]}{\sum_k w[n - kH_s]} \tag{1}$$

where

$$w[n] = \begin{cases} 0.5(1-\cos(\frac{2\pi}{N}n)) & n = 0, ..., N-1 \\ 0 & \text{otherwise} \end{cases}$$

These frames are then spaced with its **synthesis hopsize**, denoted as H_s. In this implementation, the synthesis hopsize is fixed at $H_s = L/2$, ensuring appropriate overlap between frames. H_s and H_a are related through the time-stretching factor α by the equation:

$$H_a = \frac{H_s}{\alpha} \tag{2}$$

The final output signal $y[n]$ is obtained using the following equations:

$$y[n] = \sum_k y_m[n - kH_s] \tag{3}$$

To ensure smooth reconstruction without discontinuities, the constant-overlap-add condition (COLA) must be satisfied. For this reason, the application of a window function to the analysis frames is essential. This condition ensures that the overlapping windows sum up to a constant value, avoiding amplitude modulation artifacts in the output. In Equation 1, the denominator represents the cumulative contribution of all overlapping windows at each sample n. Under the COLA condition, this summation evaluates to a constant value, typically equal to 1, thereby normalizing the output signal and preventing amplitude modulation artifacts.

There are important practical constraints on the choice of the window length L. If L is too large, the algorithm becomes less responsive to transients. Therefore, to preserve transient information effectively, we select a relatively small window size of $L = 256$ samples.

Overall, the OLA algorithm offers a balance between computational simplicity and effectiveness, especially for signals with prominent transients/percussive-heavy audio signals where percussive waveforms are short and high.

B. Phase Vocoder (PV)

The **phase vocoder** is a more sophisticated time-stretching algorithm compared to methods like OLA, as it explicitly accounts for phase information in the frequency domain. While OLA only handles the magnitude of the short-time frames and ignores the phase evolution, the phase vocoder incorporates phase modification to preserve harmonic structure during time-stretching or compression.

When stretching a signal by the time-scale factor α, simply modifying the spacing between analysis frames without adjusting the phase leads to discontinuities and loss of harmonic coherence. To address this, the phase vocoder operates on the Short-Time Fourier Transform (STFT) of the signal. The algorithm keeps the magnitude of each frame unchanged but carefully modifies the phase to ensure phase continuity across frames.

This modification is crucial to preserve constructive interference of the frequency components. Without correcting the phase, overlapping frames can interfere destructively during synthesis, resulting in degraded audio quality as shown visually in Figure 2.

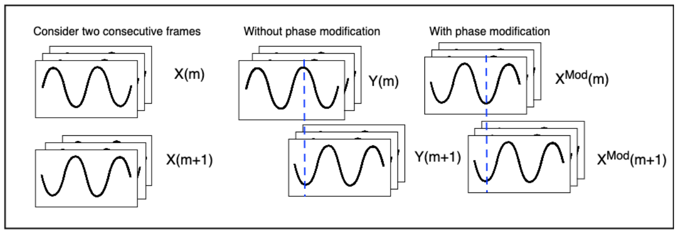

Fig. 2. Visualization of the destructive to constructive interference when applying phase modification.

The phase vocoder algorithm proceeds as follows:

1) Compute the STFT of the input signal using a suitable window function $w[n]$.
2) Modify the STFT by preserving the magnitude by altering the phase to maintain continuity.
3) Since the modified STFT may not correspond to a valid time-domain signal, reconstruction is performed by applying the inverse FFT to each modified frame to obtain synthesis frames, which are then combined using the overlap-add method.

The modified STFT is given by:

$$X^{\text{Mod}}(k,m) = |X(k,m)| \cdot e^{j\phi^{\text{Mod}}(k,m)} \tag{4}$$

Here, $\phi^{\text{Mod}}(m,k)$ denotes the modified phase for frame mm and frequency bin k, and is initialized as:

$$\phi^{\text{Mod}}(k,0) = \phi(k,0) \tag{5}$$

For subsequent frames, the phase is recursively updated using the instantaneous frequency $\omega^{\text{IF}}(k,m)$, estimated from the STFT:

$$\phi^{\text{Mod}}(k,m+1) = \phi^{\text{Mod}}(k,m) + \omega^{\text{IF}}(k,m) \cdot \frac{H_s}{f_s} \tag{6}$$

where f_s is the sampling frequency. The instantaneous frequency $\omega^{\text{IF}}(k,m)$ captures how the phase of each frequency bin evolves over time and is key to ensuring phase coherence. The instantaneous frequency is calculated with this equation:

$$\begin{aligned} &\omega^{\text{IF}}(k,m) = \omega_{\text{nom}}(k) + \\ &\frac{\Psi(\phi(k,m+1) - \phi(k,m)) - \omega_{\text{nom}}(k) \cdot H_a}{H_a} \\ &w_{\text{nom}} = 2\pi f_k \end{aligned} \tag{7}$$

where Φ is a wrapping function that ensures the angle stays in between $[-\pi, \pi]$ and f_k is the frequency that corresponds to frequency bin k, $f_k = \frac{k \cdot f_s}{L}$ (f_s is the sampling rate, while L is the window size). By incorporating phase correction, the phase vocoder allows for time-stretching of audio while preserving harmonic relationships, making it well-suited for applications involving speech, music, or any signal with structured frequency content.

As mentioned previously, a modified STFT is generally not directly invertible due to alterations in the phase. Therefore, to reconstruct the time-domain signal, the inverse FFT must be applied to each individual frame. This is expressed mathematically as:

$$x_m^{\text{Mod}}[n] = \frac{1}{N} \sum_{k=0}^{N-1} X^{\text{Mod}}(m,k), e^{j\frac{2\pi}{N}kn} \tag{8}$$

where $X^{\text{Mod}}(k,m)$ is the modified STFT, N is the FFT size and $x_m^{\text{Mod}}[n]$ represents the analysis frames in the time domain.

A similar method can be used to derive the synthesis frames for the OLA-based reconstruction. However, a key difference lies in the normalization term used during synthesis. Specifically, the OLA synthesis frame is computed as:

$$y_m[n] = \frac{w[n] \cdot x_m^{\text{Mod}}[n]}{\sum_k w[n-kH_s]^2} \tag{9}$$

This expression has a similar structure to Equation 1, with an important distinction in the denominator. While Equation 1 uses the sum of window functions to normalize overlapping contributions (to satisfy the COLA condition), Equation (9) includes the squared window in the denominator. This accounts for the energy of the windowed frames, providing a more accurate reconstruction in the presence of overlapping windows with squared energy contributions.

Once the synthesis frames $y_m[n]$ are obtained, they are recombined using the same overlap-add method described in Equation 3, ensuring continuity in the time-domain signal.

It is important to note that the window length used in the phase vocoder method is typically restricted to powers of two. This constraint ensures efficient computation of the FFT and inverse FFT operations. Additionally, a relatively long window is desirable in this context, as it allows for more accurate resolution of frequency components and better preservation of phase information over time. In our implementation, a window length of $L = 2048$ samples was selected to balance these considerations and $H_s = L/4$, the relation between H_s and H_a remain the same as Equation 2.

However, a known limitation of the phase vocoder is its tendency to disrupt vertical phase coherence. The alignment of phase across frequency bins at a given time frame. This disruption leads to smeared or washed-out transients, making the method less suitable for percussive sounds, which rely heavily on sharp, time-localized energy. As a result, while the phase vocoder performs well for harmonic content, it tends to degrade the quality of percussive elements due to its inability to maintain time-domain precision.

C. Hybrid Method

To recap the two previous algorithms, the Overlap-Add (OLA) method is particularly effective for percussive audio signals, while the phase vocoder performs better with harmonic audio signals. To leverage the strengths of both methods, it is beneficial to first separate the input signal into its harmonic and percussive components, denoted as x_h and x_p respectively. This process is known as **Harmonic-Percussive Separation (HPS)**. This separation is carried out by applying masks to the time-frequency representation of the signal. The general steps are as follows:

1) Compute the STFT of the input signal, resulting in a complex spectrogram $X(k,m)$.
2) Extract the magnitude spectrogram $Y(k,m) = |X(k,m)|$.
3) Apply **median filtering** to the magnitude spectrogram $Y(m,k)$ to enhance either harmonic or percussive structures, depending on the filtering direction. When applied along the time axis (i.e., across each frame) using a filter of shape $1 \times l$, it emphasizes horizontal (harmonic) structures, resulting in the modified spectrogram $\tilde{Y}_h(k,m)$. Conversely, applying the filter along the frequency axis with shape $l \times 1$ highlights vertical (percussive) structures, yielding $\tilde{Y}_p(k,m)$. The filter replaces each value with the median of its neighboring values within a tunable window of length l.

Median filtering is chosen for its robustness to outliers [2]. When a set of values contains one that differs greatly from the rest, the median remains close to the consistent values and is largely unaffected by the outlier.

Filtering across time emphasizes sustained, horizontal structures in the spectrogram – those correspond to harmonic content – while suppressing vertical, transient features. In contrast, filtering across frequency enhances vertical structures, making percussive components more prominent. Filtered spectrograms help generate the **binary masks** that, when applied to the original signal's STFT, will isolate the harmonic and percussive elements, denoted as $X_h(k,m)$ and $X_p(k,m)$ respectively. The binary masks are created through the following conditions:

$$M_h(k,m) \begin{cases} 1 & \tilde{Y}_h(k,m) > \tilde{Y}_p(k,m) \\ 0 & \text{otherwise} \end{cases}$$

$$M_p(k,m) \begin{cases} 1 & \tilde{Y}_p(k,m) \geq \tilde{Y}_h(k,m) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $M_h(k,m)$ and $M_p(k,m)$ are the binary masks for the harmonic and percussive parts respectively. These masks are applied as follows:

$$\begin{aligned} X_h(k,m) &= X(k,m) \odot M_h(k,m) \\ X_p(k,m) &= X(k,m) \odot M_p(k,m) \end{aligned} \tag{11}$$

Finally, reconstruction is performed separately on the two modified STFT matrices corresponding to the harmonic and percussive components. For each, the inverse FFT is applied to obtain the synthesis frames (Equation 9), which are then recombined using the overlap-add method to reconstruct the time-domain signals $x_h[n]$ and $x_p[n]$.

After obtaining the time-domain signals $x_h[n]$ and $x_p[n]$, corresponding to the harmonic and percussive components respectively, each can be processed using the appropriate algorithm (the phase vocoder for $x_h[n]$ and OLA for $x_p[n]$). The two resulting time-stretched signals are then summed to

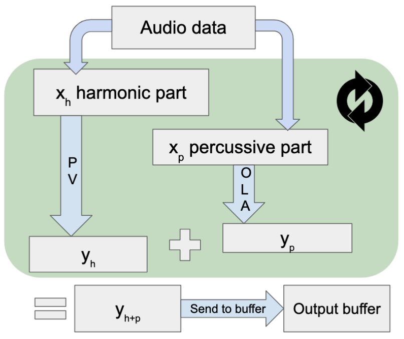

Fig. 3. High-level approach to the real-time hybrid algorithm.

produce the final time-stretched output that preserves both harmonic and percussive characteristics.

III. System Description

A. System Overview

The main approach to implementing the real-time versions of the OLA, phase vocoder, and hybrid methods are fundamentally the same as described in Section II; however, they differ in the approach to collecting the analysis frames and reconstructing the signal. Since audio is being outputted in real-time and α is a dynamic parameter, implementing these algorithms would require updating an audio buffer after a synthesis frame is calculated, shifting the audio data by the synthesis hopsize, then adding the following synthesis frame to the buffer in order to satisfy the COLA condition. This would allow for a constant audio stream that outputs the stretched audio signal for the current user-defined time-stretch factor α. The remainder of this section will describe the approach that we took to implement the real-time hybrid method – a combination of the real-time OLA and phase vocoder methods.

B. Data Pre-Processing

In order to reduce runtime and improve efficiency, HPS was performed offline. Since the input signal is not bound to change while being modified, the signal can be decomposed a single time to extract its harmonic and percussive components then manipulated (refer to Section II-C).

C. Real-Time Hybrid

The real-time implementation of the hybrid method resembles the offline implementation mentioned in Section II-C, but is designed to handle a dynamic time-stretch factor α and handles signal reconstruction differently as a result.

As shown in Figure 3, the approach to the real-time hybrid function is iterative. Before the end of the signal has been reached, the algorithm will constantly calculate the analysis hopsize, H_s, using the current time-stretch factor, as described by Equation 2.

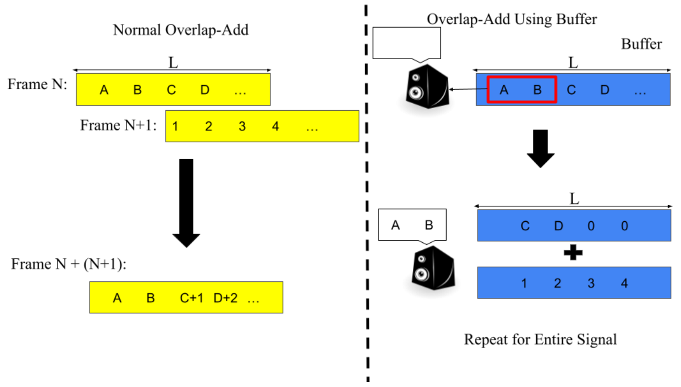

Fig. 4. Side-by-side comparison of normal overlap-add and overlap-add using an audio buffer.

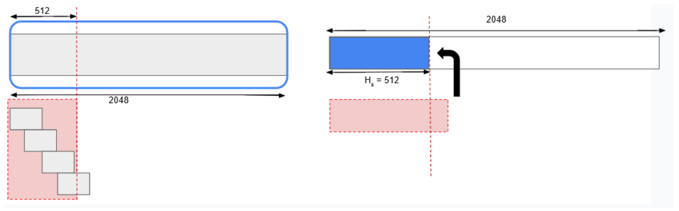

Fig. 5. Buffer management and frame alignment in hybrid time-scale modification (TSM). A PV window of length $L = 2048$ (white box) occupies the entire buffer. Given $H_{s,PV} = 512$ (blue box), multiple OLA windows of length 256 (enclosed in red) are summed to match the PV synthesis step. Since the amount of samples outputted at a time is defined by $H_{s,PV}$, this ensures synchronized output despite differing window sizes and hop sizes.

Implementing a buffer: Since we are working with real-time audio data, it is necessary to use an output buffer to temporarily store the processed audio before it is sent to the output device (in this case, the speakers). This buffer is the same length as the Phase Vocoder (PV) window and serves as a holding area for audio data after both the PV and OLA processing stages are complete. Once a new chunk of processed audio is added to the buffer, we shift the buffer contents to the left by the PV synthesis hop size, $H_{s,\text{PV}}$. The shifted-out portion is sent to the speakers, and the remaining space is filled with new incoming audio data. This ensures a continuous and real-time audio output. Refer to Figure 4 for a visual representation of this buffering and shifting process.

The details of the algorithm are as follows. After the preprocessing step, the harmonic signal is processed using the phase vocoder method (Section II-B), and the percussive signal is processed using Overlap-Add (OLA) (Section II-A).

First, we extract a windowed frame from x_h using the PV analysis hop size $H_{a,PV}$ and apply the PV synthesis procedure to this frame. The resulting synthesis frame occupies the entire length of the output buffer.

For each PV synthesis frame, we process multiple OLA windows from the percussive signal. Since the synthesis hop size for PV ($H_{s,\text{PV}}$) is generally larger than that for OLA ($H_{s,\text{OLA}}$), we loop through $i = 0$ to $H_{s,\text{PV}}/H_{s,\text{OLA}}$. In each iteration, we compute an offset based on the OLA analysis hop size and the current iteration index. Using this offset, we extract a frame from x_p, compute its OLA synthesis window, and add it to the same output buffer.

After processing both the PV and OLA components for the current segment, the system sends the resulting audio chunk to the output buffer and advances the main pointer by $H_{s,\mathrm{PV}}$. This process repeats until the end of the signal.

The core idea is that we are managing two independent frame-processing schemes in parallel: one for harmonic content and one for percussive content, each with their own analysis and synthesis hop sizes. The algorithm ensures proper alignment between these two by maintaining and updating separate pointers based on their respective hop size ratios.

D. *Runtime Analysis*

To assess the performance and practical feasibility of the proposed time-stretching system, we characterize the runtime in two key stages: precomputation and real-time processing. This distinction allows us to evaluate whether the system meets real-time constraints and to identify any stages that may require optimization. In particular, the analysis helps determine if the computational load during real-time operation is low enough for smooth playback, while also quantifying the cost of preprocessing steps such as harmonic-percussive separation.

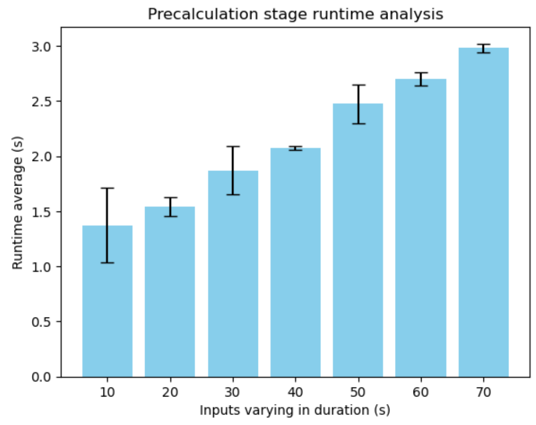

Fig. 6. Runtime analysis of the precalculation stage for varying input durations. Error bars represent the standard deviation across five trials. The graph illustrates a consistent upward trend in runtime as input length increases, with minimal variance across trials.

From Figure 6, the run time increases with the duration of the input signal, particularly during the precomputation stage, which includes the entire harmonic-percussive separation (HPS) process. We measured the average run time over five trials for varying durations of the input signal, ranging from 10 to 70 seconds in 10-second intervals. The shortest processing time was approximately 1.4 seconds, while the longest was close to 3 seconds.

This increase is expected and not problematic for real-time use. For example, a full-length song of approximately 3 minutes (180 seconds) would be expected to complete the precomputation phase in under 10 seconds.

Since this is part of the preprocessing pipeline, and does not affect real-time performance, the observed run times are considered entirely acceptable for practical use.

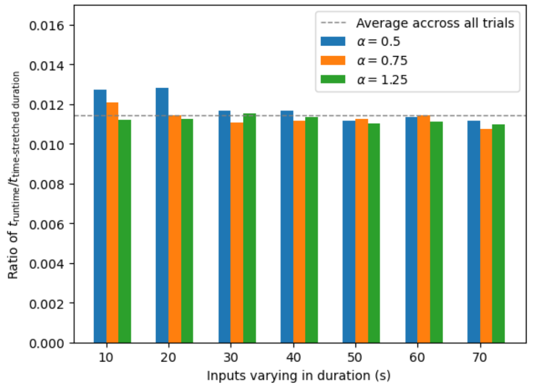

Fig. 7. Ratio of total algorithm runtime to the time-stretched signal duration for varying input lengths and stretch factors α. The dashed line represents the average ratio across all trials, indicating consistent performance well below real-time thresholds.

To determine whether the algorithm operates in real-time, we simulate a real-time run and measure the total runtime of the algorithm. Specifically, we evaluate performance by selecting different time-stretching factors α (in our case, 0.5, 0.75, and 1.25) and compute the ratio of the average runtime (across five trials) to the duration of the time-stretched signal. Additionally, we test input signals with durations ranging from 10 to 70 seconds in 10-second increments.

A ratio below 1 indicates that the algorithm processes faster than real-time, which is the desired outcome. Furthermore, a consistent ratio across different α values and input durations would suggest reliable performance regardless of the parameters used.

The figure demonstrates that across all input durations and α values, the ratio remains relatively constant, with an average value of approximately 0.0114. This is significantly below the threshold of 1, indicating that the algorithm performs well within real-time constraints.

IV. Future Work

In conclusion, the runtime analysis confirms that the proposed algorithm operates well within real-time limits across a range of time-stretching factors and input durations. The consistently low runtime-to-duration ratio that averages around 0.01, demonstrates both the efficiency and scalability of the implementation. This level of performance makes the algorithm suitable for real-time audio processing applications without requiring further optimization.

While the algorithm performs efficiently in real-time scenarios, there remains room for improvement in the precomputation stage. Reducing runtime during this phase could enhance overall responsiveness, particularly in resource-constrained environments. One potential direction is to optimize the STFT operations used in the phase vocoder stage. Specifically, implementing a lookup-based system to approximate phase and magnitude information could significantly reduce computational cost.

It is important to emphasize that the current implementation already meets real-time requirements, and such explorations may confirm that additional optimization is unnecessary. Furthermore, extending this real-time time-stretching algorithm to other practical applications, such as live performance tools, interactive audio systems, or adaptive playback technologies—presents an exciting avenue for future development.

Acknowledgment

We would like to thank Professor TJ Tsai for his guidance and support throughout the course. His availability and clear explanations helped us better understand key concepts in digital signal processing and contributed to the completion of this project.

References

[1] J. Driedger and Meinard Müller, "TSM TOOLBOX: MATLAB IMPLEMENTATIONS OF TIME-SCALE MODIFICATION ALGORITHMS," Jan. 2014.

[2] M. Müller, "Harmonic–Percussive Separation," AudioLabs Erlangen, [Online]. Available: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C8/C8S1_HPS.html [Accessed: May 8, 2025].