TISMIR

A Real-Time Beat Tracking System with Zero Latency and Enhanced Controllability

RESEARCH ARTICLES

]u[ubiquity press

PETER MEIER

CHING-YU CHIU

MEINARD MÜLLER

*Author affiliations can be found in the back matter of this article

ABSTRACT

Identifying beat positions in music recordings, a central task in Music Information Retrieval (MIR), is commonly referred to as beat tracking. Typically, this involves computing an activation function to reveal frame-wise beat likelihood and then conducting post-processing to derive final beat positions. Existing methods often operate offline, requiring access to the entire music track for processing. In this article, we introduce a real-time beat tracking system based on the predominant local pulse (PLP) concept, originally designed for offline use. Our main contribution is the successful transformation of the PLP-based algorithm into a real-time procedure. Unlike traditional offline methods providing static beat positions, our real-time approach dynamically captures changes in local pulse characteristics with each frame of an audio stream. This yields additional insights, including beat context, beat stability, and beat lookahead for predicting beats in advance. In this way, our system not only demonstrates high controllability for real-time applications but also can operate at zero latency. Additionally, we present experiments comparing our real-time beat tracking system with other models and evaluating the accuracy of our lookahead feature. Finally, we showcase two real-world applications for interactive music making and educational music gaming that creatively leverage our system's output. In summary, our real-time beat tracking system offers a lightweight algorithm that is particularly well-suited for interactive music software development.

CORRESPONDING AUTHOR:

Peter Meier

International Audio Laboratories Erlangen, Am Wolfsmantel 33, 91058 Erlangen, Germany

peter.meier@audiolabs-erlangen.de

KEYWORDS:

Real-Time, Beat Tracking, Predominant Local Pulse, Zero Latency, Controllability

TO CITE THIS ARTICLE:

Meier, P., Chiu, C.-Y., & Müller, M. (2024). A Real-Time Beat Tracking System with Zero Latency and Enhanced Controllability. *Transactions of the International Society for Music Information Retrieval,* 7(1), 213–227. DOI: https://doi.org/10.5334/tismir.189

1 INTRODUCTION

Music Information Retrieval (MIR) plays a key role in analyzing and understanding musical content, with beat tracking standing out as a central task within this field. The general idea of beat tracking is to have a system that analyzes music and automatically taps along with the beat, similar to what human listeners would do (Dixon and Cambouropoulos, 2000; Ellis, 2007). The human perception of beats in music can occur at different metric levels, for example, the tactus level (quarter notes), the measure level (bars), and the tatum level (temporal atom), which refers to the fastest repeating temporal pattern in the music (Müller, 2021).

Besides beat tracking, there are several closely related concepts: While tapping and marking beat positions in a piece of music, beats can be counted and grouped into bars, a process called *meter tracking* (Holzapfel and Grill, 2016; Klapuri et al., 2006). Identifying the first beat of each bar is helpful for many music applications and is referred to as *downbeat tracking* (Böck et al., 2016b; Durand et al., 2015; Krebs et al., 2016). Analyzing the number of beats per minute (BPM) helps to identify the tempo of the music, which is commonly known as *tempo tracking* (Krebs et al., 2015).

In this work, we mainly focus on the task of beat tracking that involves identifying the time positions of beats within a music recording and is essential for MIR applications ranging from music transcription to rhythm-based interactive systems. Typically, beat tracking is a two-step process: first, calculating an activation function to extract frame-wise beat likelihood from the music signal, and second, performing a postprocessing method to determine final beat positions from the activation function.

Most existing beat tracking methods are designed for offline use and rely on access to the entire music track for processing. In recent years, deep learning has led to substantial improvements in beat tracking with approaches such as Temporal Convolutional Networks (`TCN`) (Böck and Davies, 2020; Davies and Böck, 2019), `Transformer` (Zhao et al., 2022), or `SpecTNT-TCN` (Hung et al., 2022). With the growing demand for real-time applications like interactive music systems and live performance tools, the need for online-capable beat tracking algorithms has become more evident. This trend is reflected in contributions such as `BeatNet` (Heydari et al., 2021), `Novel-1D` (Heydari et al., 2022), or the most recent model `BEAST-1` (Chang and Su, 2024).

When employing online approaches for real-time applications or integrating them into larger interactive systems, two prominent challenges emerge: latency and controllability. First, latency encompasses delays not only from the beat tracker itself but also from network communication, audio output processing, or controller input lags. These delays make it challenging to maintain synchronization between analyzed input audio streams and generated output audio streams. Second, controllability in this context refers to the challenges most online beat trackers face in being easily adjustable for specific real-time application requirements. Most real-time beat tracking systems lack explicit control over parameters such as pulse level (e.g., quarter-note or eighth-note level) or the amount of latency.

In this article, we present a real-time beat tracking system built on the predominant local pulse (PLP) concept (Grosche and Müller, 2011), originally developed for offline use. Our main contribution is the successful transformation of the PLP-based algorithm into a real-time procedure. With our approach, we achieve beat detection performance comparable to that of most other online beat trackers. In contrast to previous beat trackers, our method allows for achieving zero latency while remaining lightweight and easily controllable. In particular, our approach incorporates beat lookahead functionality to mitigate latency effects in interactive real-time systems and provides enhanced controllability through adjustable parameters.

To evaluate the influence of the lookahead feature on beat detection performance, we conduct experiments using a variety of datasets. In addition to quantitative evaluation, we demonstrate the practical usability of our system through two real-world applications for interactive music making and educational music gaming. These applications creatively utilize the output of our system, highlighting its controllability and versatility. In summary, our real-time beat tracking system represents a lightweight algorithm that is especially useful for interactive music software development. By bridging the gap between offline beat tracking methods and real-time application requirements, our system opens up new possibilities for musicians to use real-time beat tracking for their interactive music systems.

The subsequent sections of this article follow this structure: In Section 2, we explore the original PLP-based algorithm. Section 3 discusses the conversion of this algorithm into a real-time procedure. In Section 4, we analyze the system output resulting from this real-time procedure. With Section 5, we conduct multiple experiments, and their results are discussed in Section 6. We demonstrate our real-time system through two real-world scenario applications in Section 7. Finally, Section 8 concludes our method. Additional materials and audio examples are available on a supplemental website.[1]

2 PLP-BASED ALGORITHM

Before discussing how to convert the PLP-based algorithm into a real-time procedure (Section 3), we first look at the original offline procedure (Grosche and Müller, 2011). All the essential steps for calculating the PLP are illustrated in Figure 1 (left column).

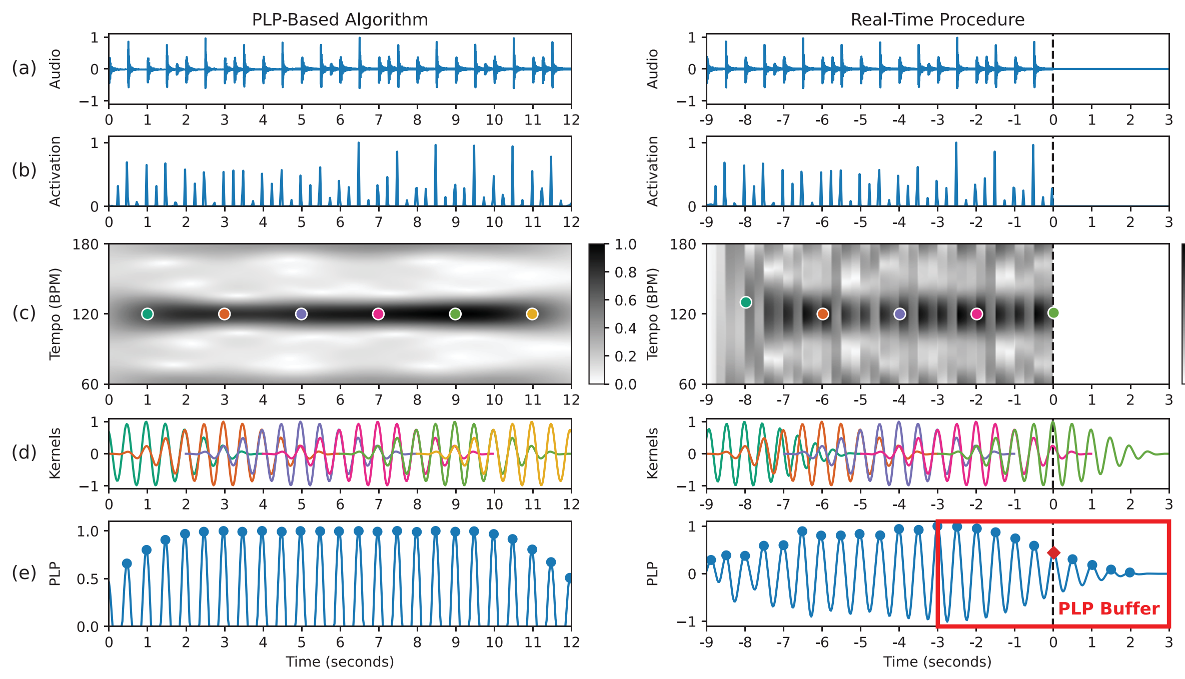


Figure 1 Illustration of the original offline PLP-based algorithm (left side), as described in Section 2, and the real-time procedure (right side), detailed in Section 3. **(a)** Audio signal. **(b)** Activation function. **(c)** Tempogram. **(d)** Pulse kernels. **(e)** PLP function. To provide clearer visualization and illustrate the general idea, we plot kernels only at 2-second intervals.

First, the audio signal (Figure 1a) is transformed into an activation[2] function $\Delta : \mathbb{Z} \to \mathbb{R}$ for time positions $n \in \mathbb{Z}$. The peaks of the activation function Δ (Figure 1b) indicate the likelihood of observing a beat at time positions n. To identify local periodic patterns, we perform a Short-Time Fourier Transform (STFT) on Δ. For this, we define a window function $\mathcal{W} : [-N : N] \to \mathbb{R}$ for $N \in \mathbb{N}$ that is normalized and centered around each time position n, resulting in a total window size of $K = 2N + 1$. For an arbitrary but fixed time position $n \in \mathbb{Z}$, the window $\mathcal{W}$ defines a neighborhood indexed by $m \in [n - N : n + N]$. The complex-valued Fourier coefficient

$$\mathcal{F}(n,\omega) = \sum_{m=n-N}^{n+N} \Delta(m) \cdot \mathcal{W}(m-n) \cdot \exp(-2\pi i \omega m) \tag{1}$$

obtained from the STFT is defined for time positions $n \in \mathbb{Z}$, frequencies $\omega \in \mathbb{R}_{\geq 0}$, and local time indices $m \in [n - N : n + N]$. From the Fourier coefficient $\mathcal{F}(n,\omega)$, we derive a Fourier tempogram $\mathcal{T}(n,\tau)$, given by

$$\mathcal{T}(n,\tau) = |\mathcal{F}(n,\tau/60)|. \tag{2}$$

This time–frequency representation of the activation function Δ, as illustrated in Figure 1c, is commonly measured in BPM rather than Hertz, with a tempo parameter $\tau = 60 \cdot \omega$. Utilizing the Fourier tempogram $\mathcal{T}(n,\tau)$ relies on two assumptions commonly employed in beat tracking. First, beat positions go along with note onsets, and, second, beat positions are periodically spaced. These assumptions are exploited by comparing local sections of the activation function Δ with windowed sinusoidal kernels $\kappa_n : [n - N : n + N] \to \mathbb{R}$, given by the equation

$$\kappa_n(m) := \mathcal{W}(m-n) \cdot \cos(2\pi\left((\tau_n/60) \cdot m - \varphi_n\right)), \tag{3}$$

for $m \in [n - N : n + N]$, as shown in Figure 1d. To obtain a local beat (or pulse) tracker, we choose for each time position $n \in \mathbb{Z}$ a kernel κ_n that optimally matches the local tempo structure of the signal within a given tempo range Θ, as illustrated with colored dots in Figure 1c. For instance, when denoting $\Theta = [30 : 240]$, we refer to a tempo range spanning from 30 to 240 BPM. In Equation 3, τ_n is defined as the tempo parameter that maximizes the tempogram $\mathcal{T}(n,\tau)$ for each time position n.

$$\tau_n := \underset{\tau \in \Theta}{\operatorname{argmax}}(\mathcal{T}(n,\tau)) \tag{4}$$

Additionally, φ_n represents the phase of the windowed sinusoid with tempo τ_n that best correlates with the local section around n of the activation function Δ.

$$\varphi_n = -\frac{1}{2\pi}\text{angle}(\mathcal{F}(n,\tau_n/60)) \tag{5}$$

To obtain a globally defined pulse tracker, the final step of the original offline procedure involves overlap-adding all optimal pulse kernels over time to form a global pulse function $\Gamma : \mathbb{Z} \to \mathbb{R}_{\geq 0}$:

$$\Gamma(n) = \left| \frac{1}{C} \sum_{\ell=n-N}^{n+N} \kappa_\ell(n) \right|_{\geq 0} \tag{6}$$

This curve, which reveals predominant local pulse (PLP) information, is referred to as the PLP curve. To reach its final state, the PLP function Γ undergoes two additional computation steps, included in Equation 6: normalization and half-wave rectification. The constant

$$C = \sum_{n=-N}^{N} \mathcal{W}(n) \tag{7}$$

ensures the normalization of the PLP function, keeping the values $\Gamma(n)$ within the range of $[-1 : 1]$. With halfwave rectification, we only consider the positive values of the PLP function, where $|x|_{\geq 0} := x$ for a nonnegative real number x and $|x|_{\geq 0} := 0$ for a negative number x. For a more detailed description, we refer to Grosche and Müller (2011).

3 REAL-TIME PROCEDURE

MIR is an area of research that often deals with the analysis of entire corpora. The algorithms used for these analyses typically operate offline. In contrast to this, the algorithms used for real-time analysis face multiple challenges. This includes working with causal data only, addressing the trade-off between accuracy and latency, and meeting audio processing deadlines, as discussed by Stefani and Turchet (2022). In adapting our model from offline to real-time (Section 3.1), we encountered three significant differences, which we discuss in more detail in the following subsections. First, data is streamed in small blocks, requiring the balancing of latency and computation times (Section 3.2). Second, only causal data is available, affecting the overall accuracy of the beat tracker (Section 3.3). Third, data needs to be stored in buffers, enabling access to previous time frames for handling larger context windows (Section 3.4).

3.1 REAL-TIME PLP ALGORITHM

The original offline procedure outlined in Section 2 can be transformed into a real-time procedure, as depicted in Figure 1 (right column). Given the centric nature of the kernels κ_n around each time position n, the kernel window $\mathcal{W}$ is essentially divided into two halves, see Figure 1d. The left half of the kernel window supporting $[-N : 0]$ is utilized to compute the pulse structure based on past and present data, while the right half supporting $(0 : N]$ is used to extrapolate this pulse structure to future time positions. This allows the superimposed kernels κ_n to predict future beat positions, as illustrated in Figure 1e.

To describe the real-time procedure, we update the equations in Section 2 as follows. Let n_0 denote the current time position, where we have access to activation values $\Delta(n)$ for all time positions $n \leq n_0$. The complex-valued Fourier coefficient $\mathcal{F}(n, \omega)$ from Equation 1 becomes

$$\mathcal{F}'(n, \omega) = \sum_{m=n-N}^{n} \Delta(m) \cdot \mathcal{W}(m-n) \cdot \exp(-2\pi i \omega m), \quad (8)$$

which is defined for all time positions $n \leq n_0$, frequencies $\omega \in \mathbb{R}_{\geq 0}$, and local time indices $m \in [n-N : n]$. For the current time position n_0, we obtain a real-time PLP function called $\Gamma_{n_0} : [-\infty : n_0 + N] \rightarrow \mathbb{R}$

$$\Gamma_{n_0}(n) := \frac{1}{C} \sum_{\ell=n-N}^{n_0} \kappa_\ell(n), \quad (9)$$

which is defined for all time positions $n \in [-\infty : n_0 + N]$ and has access to all kernels κ_ℓ for $\ell \in [n-N : n_0]$. For normalization, we use the constant C from Equation 7, which ensures that the values $\Gamma_{n_0}(n)$ lie within the range of $[-1 : 1]$. Additionally, for the real-time PLP function, we choose to skip half-wave rectification and preserve a more sinusoidal state to better capture dynamic pulse changes over time. Note that the kernels κ_ℓ are computed based on $\mathcal{F}'(n, \omega)$ (Equation 8), denoted with $\mathcal{T}'(n, \tau)$ for the Fourier tempogram (Equation 2), τ'_n for the tempo parameter (Equation 4), and φ'_n for the phase (Equation 5). The PLP buffer, depicted in Figure 1e, displays only the section of $\Gamma_{n_0}(n)$ for $n \in [n_0 - N : n_0 + N]$, containing all the necessary information to compute the subsequent time position n_1. Note that for this buffer, our perspective shifts from a linear time scale to a centric viewpoint, where the center of the buffer represents the current time position n_0 (which corresponds to the physical buffer time position $t = 0$, as illustrated in Figure 2).

3.2 AUDIO STREAMING

The main difference between offline and real-time processing lies in how data is delivered. In offline processing, the entire audio track is available at any time. Conversely, in real-time processing, audio data is streamed continuously as it becomes accessible. Streamed audio is commonly processed in small blocks of data known as *frames*, which directly correspond to the time positions n introduced in Section 2. Each frame contains a fixed frame size $M \in \mathbb{N}$ of recent audio data sampled at a fixed audio sampling rate F_s. The duration of a single frame is called frame period T, as given by the equation

$$T = \frac{M}{F_s}. \quad (10)$$

Consequently, when we receive a frame of data, we are inherently operating with a *latency* equivalent to one frame period T. For example, if a real-time system runs with an audio sampling rate $F_s = 44100$ Hz and a frame size $M = 512$ samples, the system latency based on the frame period is $T = 512/44100 = 11.61$ ms. Reducing the frame size M to minimize system latency is often not feasible because the frame period T also determines the available computation time per frame.

3.3 CAUSAL DATA PROCESSING

PLP operates with centric kernels κ_n, analyzing a specific time position n within a larger context window $\mathcal{W}$ by utilizing both past and future data, with local time indices $m \in [n-N : n+N]$. However, for the real-time procedure, only causal data is available, limiting the indices to $m \in [n-N : n]$. While delaying the computation to await future data is possible, this approach is not feasible for real-time applications. For this reason, in our real-time model, we only utilize past and present data for PLP computation. This helps avoid additional latency but also

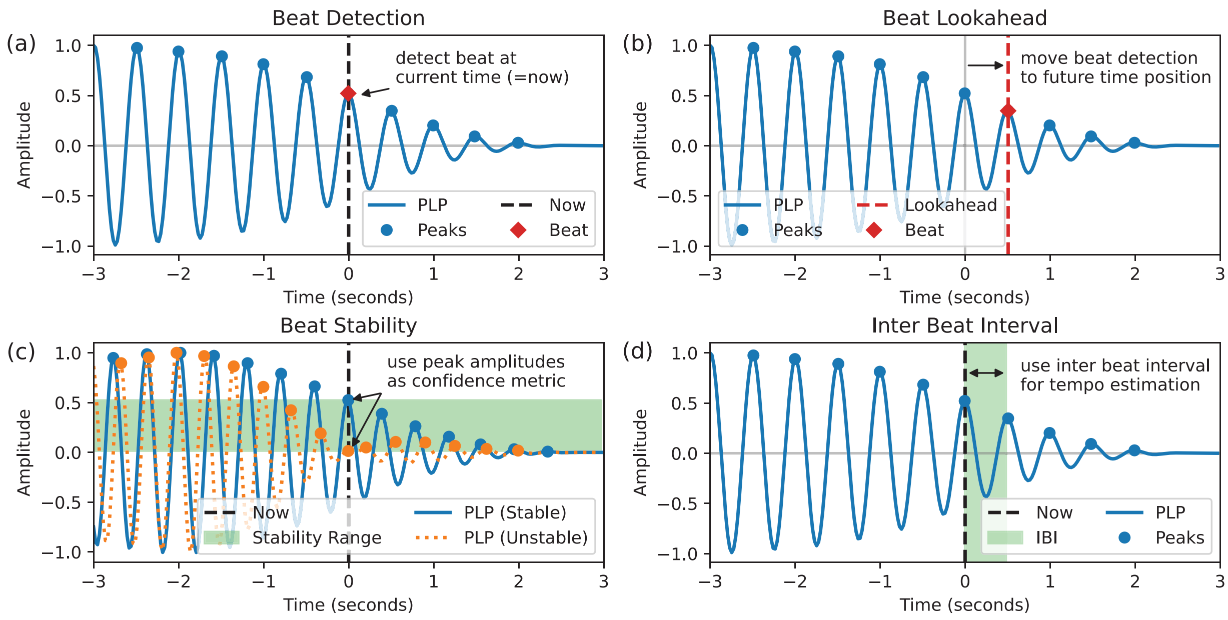


Figure 2 The output of the real-time beat tracking system: **(a)** Beat Detection (Section 4.1). **(b)** Beat Lookahead (Section 4.3). **(c)** Beat Stability (Section 4.4). **(d)** Inter Beat Interval (Section 4.5).

impacts other aspects of the beat tracking pipeline, as we will discuss in the following.

The tempogram $\mathcal{T}'(n,\tau)$ is based on only past and present activation data, which has multiple effects. First, the data for calculation is essentially half the size and therefore less accurate. Second, the lack of future data makes it harder to adapt to upcoming tempo changes. Third, the activation data displays a discontinuity, abruptly dropping to zero due to the absence of future data. This discontinuity introduces artifacts, manifesting as vertical lines in the tempogram, as depicted in Figure 1c.

The real-time PLP function Γ_{n_0} incorporates only past and present kernels κ_ℓ, as we restrict the kernel summation in Equation 9 to local time indices $\ell \in [n-N : n_0]$. Consequently, the right half of the PLP buffer, which represents future pulse data, consistently exhibits a falling slope due to the absence of overlapping kernels on that side of the buffer. This leads to less prominent peaks in the PLP buffer $\Gamma_{n_0}(n)$ for $n \in [n_0 - N : n_0 + N]$ characterized by overall lower amplitude values, as illustrated in Figure 1e.

3.4 DATA BUFFERING

Compared to a single frame size M, the kernel window $\mathcal{W}$ for PLP is relatively long, typically utilizing kernel sizes K of 4–12 s. Therefore, a single frame of audio data is insufficient for computing PLP, and it is necessary to collect and store frames in a buffer. To achieve this in a memory-efficient manner, we buffer only as much data as necessary to compute the current frame. For this purpose, a First-In-First-Out circular buffer is utilized, where the oldest values are dropped as new values are added. For real-time PLP, there are two computations where buffers are needed. First, to compute a kernel, an *activation buffer* with half kernel size K is required ($K_{\mathrm{Act}} = N + 1$), supporting $[-N : 0]$. Second, to overlap-add kernels with previous kernels, a *PLP buffer* with full kernel size K is needed ($K_{\mathrm{PLP}} = K = 2N + 1$), supporting $[-N : N]$. The PLP buffer is fully described by $\Gamma_{n_0}(n)$ for the section $n \in [n_0 - N : n_0 + N]$ in Equation 9.

4 SYSTEM OUTPUT

The primary objective of the real-time procedure (Section 3) is to update the PLP buffer with every new frame of data, thereby generating all the system output, as we will discuss in the following subsections.

4.1 BEAT DETECTION

With Figure 2a, we demonstrate the process of *beat detection* in our real-time beat tracking system. For offline processing, the goal of beat detection is to compile a list of beat positions, indicating when beats occur relative to the start of an audio track. For real-time audio streaming, the objective of beat detection is to determine if a beat should be triggered at the current time position n_0. This is achieved by analyzing the current state of the PLP buffer and using a simple peak picking method to identify peak positions $P = \{p_1, p_2, \ldots\}$ within the section $n \in [n_0 - N : n_0 + N]$. If a peak $p_i \in P$ falls at the center time position of the PLP buffer ($p_i = n_0$), a beat occurs at the current time position n_0 and should be triggered immediately.

4.2 BEAT CONTEXT

Even though the beat detection method outlined in Section 4.1 primarily focuses on the center time position n_0 of the PLP buffer, every peak $p_i \in P$ provides additional valuable information. They create a form of *beat context* around the current time position n_0, giving insights into both past and potential future beat positions. This functionality proves useful for real-time applications, as demonstrated in Section 7.2. The extent to which past and future information is available can be adjusted by modifying the PLP kernel size K. However, this also affects how sensitive the beat tracker is to tempo variations in the music: Increasing the kernel size K reduces the tracker's sensitivity to tempo fluctuations. Conversely, decreasing the kernel size K enhances the tracker's responsiveness to tempo changes.

4.3 BEAT LOOKAHEAD

For beat detection, as discussed in Section 4.1, we typically use the center position n_0 of the PLP buffer as the "decision line." However, by considering the beat context (Section 4.2), we can shift this decision line to any time position $n \in [n_0 - N : n_0 + N]$, including future time positions $n > n_0$, as illustrated in Figure 2b. To achieve this, we introduce the *beat lookahead*, an input parameter influencing the real-time beat tracking system's behavior. Instead of detecting beats at the center time position n_0 of the PLP buffer, the beat lookahead defines an offset, to move the decision line by a fixed number of frames to detect and trigger beats ahead of time.

Although triggering beats earlier may not appear intuitive, it becomes a valuable solution in addressing significant latency issues inherent in most real-time applications. These delays often arise from various sources, including audio processing, network communication, or input controller lags. Therefore, the beat lookahead is crucial for compensating for these latencies and ensuring synchronization between the analyzed input signal and the generated output signal.

However, predicting beats ahead of time comes with a trade-off: the accuracy of beat detection diminishes to some extent, which we elaborate on in Section 6.4.

4.4 BEAT STABILITY

Every time beat detection (Section 4.1) occurs, the PLP function Γ_{n_0} can have varying amplitude values $\Gamma_{n_0}(p_i)$ for peak positions $p_i \in P$, which directly fall at the decision line, such as the center time position of the PLP buffer ($p_i = n_0$). These amplitude values serve as indicators of *beat stability*, as depicted in Figure 2c. A high peak amplitude $\Gamma_{n_0}(p_i)$ signifies a *stable beat* situation, indicating that the neighboring optimal kernels $\varkappa_\ell$ were similar in tempo and constructively added up over time (constructive interference). Conversely, a low peak amplitude indicates an *unstable beat* situation, where neighboring optimal kernels $\varkappa_\ell$ with different tempi have canceled each other out (destructive interference). To this end, beat stability values can be utilized to control parameters in real-time applications, such as the volume of an accompaniment track, as detailed in Section 7.1.

Between the minimum and maximum peak amplitudes lies an entire *stability range*. Since the PLP function Γ_{n_0} is normalized (see constant C from Equation 7), the beat stability can have values between $[0 : 1]$. In this way, the peak amplitudes act as a *confidence metric* that can be used in addition to beat detection. For example, if an application requires consistent beat output sequences without unstable and noisy beat sections, a *stability threshold* can be introduced. This threshold can serve as an optional beat filter that allows beats to be processed only when they surpass a certain amplitude value, as we will discuss in Section 7.2.

4.5 INTER BEAT INTERVAL AND LOCAL TEMPO

Providing a local tempo measure for the current time position n_0 is particularly beneficial for interactive music making (Section 7.1), where the tempo can serve as an input parameter for time-based music instruments and effects plugins (e.g., sampler, reverb, delay, or echo). With PLP, we have two different approaches for determining local tempo. First, on the frame level, the PLP kernel $\varkappa_\ell$ is calculated based on the tempo parameter τ'_n, which directly yields a local tempo value. Second, the PLP function Γ_{n_0} offers a more consistent tempo measure, defined by the inter beat interval of two consecutive peaks, as illustrated in Figure 2d.

The tempo output can be bounded by setting minimum and maximum tempo values for the PLP procedure, thereby defining a specific tempo range Θ. Modifying this tempo range allows us to influence the pulse level at which the beat tracker should operate, such as the normal tactus level (quarter notes) or one tempo octave higher with double tactus level (eighth notes). For instance, if the expected tempo for a beat tracking application is around 100 BPM, we can focus the tempo output on the range $\Theta = [80 : 120]$ for normal tempo or choose $\Theta = [180 : 220]$ to force double tempo output.

5 EXPERIMENTS

To evaluate the described methods, we carry out multiple experiments. First, we compare our method with various low-latency beat trackers under specific oracle conditions and against methods from the existing literature for beat performance, latency, and tempo range. With the second experiment, we concentrate on an assessment of context-sensitive beat evaluation on different tempo ranges. Third, we evaluate our real-time procedure across different kernel sizes. Last, in the fourth experiment, we investigate the impact of the lookahead parameter on beat detection performance.

5.1 DATASETS

For our experiments, we employ a diverse set of commonly used datasets. We report on the average track duration, tempo, and stability in Table 1. To calculate tempo stability, we convert all inter beat intervals to tempo values and normalize them by dividing each by its respective average track tempo, maintaining a tolerance interval of ±4% for stable tempi. For a formal definition of tempo stability, we refer to Schreiber et al. (2020).

The `Ballroom` dataset, introduced by Gouyon et al. (2006), provides audio excerpts categorized by different dance music styles, displaying a wide variety of tempo ranges. `GTZAN`, described by Tzanetakis and Cook (2002), comprises audio excerpts spanning a diverse range of genres, including Country, Metal, Hiphop, Reggae, Jazz, and Classical. The `Rock` dataset, as introduced by Clercq and Temperley (2011), features songs listed in *Rolling Stone* magazine's "500 Greatest Songs of All Time" and generally exhibits lower overall tempo stability compared to the other datasets. Additionally, the `RWCPop` dataset, described by Goto et al. (2002), provides a collection of pop songs with full audio recordings and high tempo stability.

Figure 3 offers an overview of the datasets, showing that `Rock` and particularly `GTZAN` include tempo values exceeding the range $\Theta = [30 : 240]$ handled by our online model, see also Section 5.5. Given that online models typically function within a more limited tempo spectrum compared to offline models, it is important to note that the wider tempo span of these datasets could potentially affect the overall performance of beat estimation.

5.2 ACTIVATION FUNCTIONS

For the activation functions, we explored two distinct online methods. The first method (`RNN`) involves a machine learning approach, utilizing an online Recurrent Neural Network (RNN) activation with a specifically chosen single Long Short-Term Memory (LSTM) model (Böck and Schedl, 2011). In particular, we adopted the approach utilizing `madmom`'s `RNNBeatProcessor (online=True)` along with their pre-trained LSTM model 4 (Böck et al., 2016a). It is worth noting that these LSTM models are trained on the `Ballroom` dataset and therefore may exhibit superior performance on it compared to other datasets. The second method (`GT`) is a "ground truth" activation, which is derived from

Dataset	**Dataset** (Total)			**Track** (Average)		
Name	Tracks	Length	Type	Duration	Tempo	Stability
`Ballroom`	698	6h 03m	Excerpt	31 ± 1 s	129.7 ± 39.7 BPM	89.0 ± 13.4 %
`GTZAN`	993	8h 16m	Excerpt	30 ± 0 s	119.4 ± 39.6 BPM	90.5 ± 17.2 %
`Rock`	200	12h 53m	Full	232 ± 91 s	115.7 ± 34.7 BPM	81.6 ± 15.0 %
`RWCPop`	100	6h 46m	Full	244 ± 41 s	111.7 ± 27.3 BPM	97.9 ± 10.8 %

Table 1 Overview of the datasets used for evaluation.

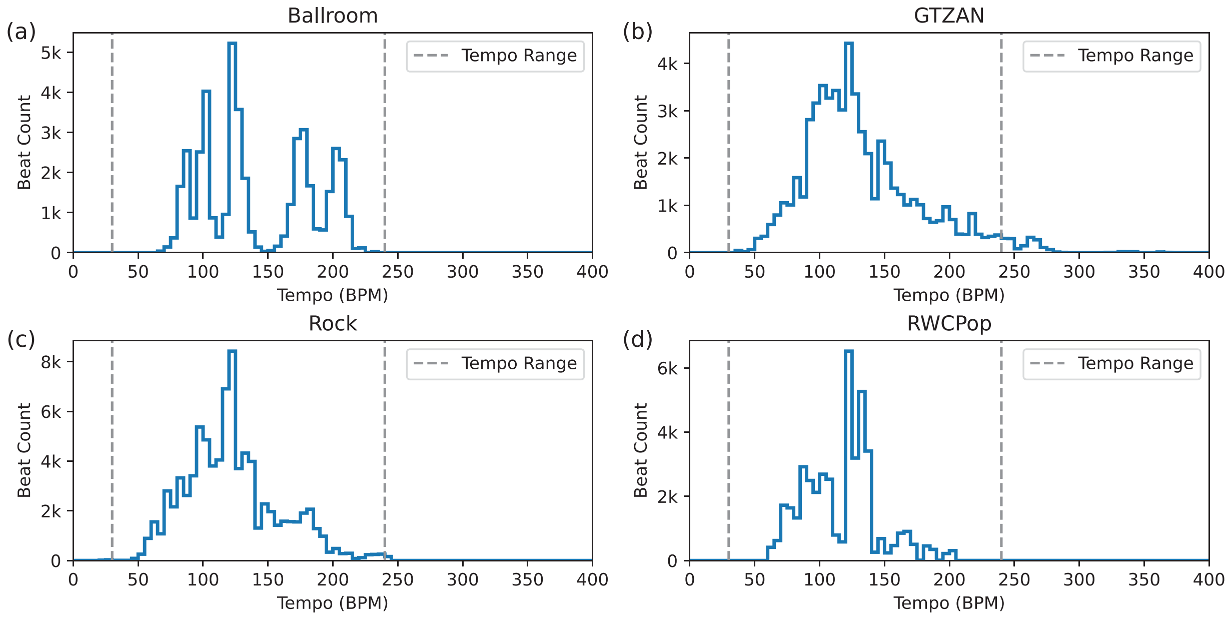


Figure 3 Beat-wise distribution of inter beat intervals (IBI) in various datasets, considering a tempo resolution bin size of 5 BPM. The tempo range for our online model (30–240 BPM) is indicated with dashed lines.

annotated beat positions. For each time position n, it outputs an amplitude value of 1 at annotated beat positions and 0 otherwise. To achieve this, we convert annotated beat positions (in seconds) into frame indices based on the frame period T (see Equation 10). This approach is valuable for analyzing post-processing methods independently of the activation, assuming best possible activation performance.

5.3 POST-PROCESSING METHODS

In this paper, the term "post-processing" is used for a beat tracking method that determines beat positions from an activation function. Specifically, our PLP-based post-processor takes an activation function as input, generates the PLP pulses as described in Section 2, and detects the beats accordingly. For our experiments, we outline the following post-processing methods: As a baseline reference, labeled as `PLP-Off`, we use the original PLP concept to directly compare the transition from offline to online. Our online adaptation of the original PLP method is referred to as `PLP-On`.

5.4 EVALUATION MEASURES

For evaluating beat estimation, we employ the standard F1-score metric with a tolerance window of ±70 ms, as implemented in `mir_eval` (Raffel et al., 2014). For context-sensitive evaluation, we apply the L-correct metric, as introduced by Grosche and Müller (2011). With this evaluation method, we consider not just a single beat but a series of consecutive estimated beats, each with a temporal context of length $L \in \mathbb{N}_{\geq 2}$. This is similar to how a listener would tap along to music, often needing a sequence of beats to adapt to tempo changes. Therefore, context-sensitive evaluation is valuable for evaluating how well our method can handle larger beat contexts (Section 4.2). To avoid initialization artifacts in the evaluation, we disregard all beats occurring before 5 s of each music track.

5.5 EXPERIMENT SETUP

For our experiments, we focus on an application-ready setup, using an audio sampling rate $F_s = 44100$ Hz and a frame size $M = 512$ samples. These configurations are typical for real-time audio hardware, such as an audio interface, resulting in a frame period $T = 11.61$ ms. The default tempo range for our experiments is $\Theta = [30 : 240]$, unless stated otherwise. With a tempo of 30 BPM, the beat events are 2 s apart, which can be considered as approaching the lower bound of human tempo perception. With an upper tempo of 240 BPM, we cover a full range of three tempo octaves (doubling the tempo of 30 BPM for three times). With `TR40`, we denote a tempo range of ±40% of the average track tempo, assuming that the mean tempo of a music recording is given. Throughout this paper, we consistently use the PLP method with a fixed kernel size $K = 2N+1$ of 6 s for all experiments and examples. Further discussion on this choice is provided in Section 6.3.

6 RESULTS AND DISCUSSION

6.1 OVERVIEW AND METHODS COMPARISON

In Table 2, we compare various beat tracking methods for F1-score, latency, and tempo range to assess the performance of our models on the `GTZAN` dataset. The latency values listed are based on the individual frame period T of each respective method (see Equation 10). Our default online model `RNN-PLP-On` uses the `RNN` activation as input for the `PLP-On` post-processing and achieves an F1-score of 74.72%, which falls within the range of most other online beat trackers. However, our approach has only a very small latency of 11.61 ms. The leading online model in our list, `BEAST-1`, achieves an F1-score of 80.04%, but it comes with a significantly higher latency of 46.44 ms. This delay could be perceived as distinct acoustic events by human ears, making it unsuitable for certain real-time audio applications where precise synchronization between input and output audio streams is important. Furthermore, we introduce `RNN-PLP-On-Zero` with an F1-score of 74.68%, which stands out as the only model in the table with a latency of 0 ms. Zero latency is achieved by offsetting the frame period $T = 11.61$ ms (see `RNN-PLP-On`) with a lookahead of 1 frame, allowing beats to be triggered 11.61 ms ahead of time. With `RNN-PLP-On-TR40` and `GT-PLP-On`, we showcase potential enhancements of our online model when utilizing oracle conditions, such as a known average tempo range (`TR40`) or a perfect ground truth activation function (`GT`). In addition, we report on offline PLP models `RNN-PLP-Off`, providing a more comprehensive comparison between offline to online processing.

6.2 CONTEXT-SENSITIVE EVALUATION ON TEMPO RANGE

With Figure 4 we report on results using the context-sensitive L-correct metric for different tempo ranges. When analyzing our online model `PLP-On` with the activation function `RNN`, we observe a decrease in F1-score from 74.72% (no context) to 52.66% ($L = 2$). This reflects the fact that detecting a series of beats accurately is more challenging than detecting a single beat. Further increasing the number of consecutive beats from $L = 2$ to $L = 8$, the F1-scores remain almost constant, with a slight decrease of about 3.09% (from 52.66% to 49.57%). This implies that the PLP method is inherently designed to effectively handle larger beat contexts, a characteristic that holds true for the online method as well.

When analyzing `RNN-PLP-On-TR40`, which utilizes a tempo range of ±40% of the average track tempo, we

Model	Mode	Comments	F1-score (%)	Latency (ms)	Tempo (BPM)
`RNN-PLP-On`	`Online`	our model	74.72	11.61	30 - 240
`RNN-PLP-On-Zero`	`Online`	our model (zero latency)	74.68	**0.00**	30 - 240
Exploratory Studies: Oracle Conditions					
`RNN-PLP-On-TR40`	`Online`	(`C1`) use avg. track tempo	75.11	11.61	track (mean) ±40%
`GT-PLP-On`	`Online`	(`C2`) use `GT` activation	91.93	11.61	30 - 240
`RNN-PLP-Off`	`Offline`	(`C3`) use non-causal data	79.07	–	30 - 240
`RNN-PLP-Off-TR40`	`Offline`	(`C4`) use avg. track tempo	82.00	–	track (mean) ±40%
`GT-PLP-Off`	`Offline`	(`C5`) use `GT` activation	97.83	–	30 - 240
Methods Overview: Comparing with Literature					
`BEAST-1`	`Online`	Chang and Su (2024)	80.04	46.44	55 - 215
`Novel-1D`	`Online`	Heydari et al. (2022)	76.48	20.00	55 - 215
`BeatNet`	`Online`	Heydari et al. (2021)	75.44	20.00	55 - 215
`Böck-FF`	`Online`	Böck et al. (2014)	74.18	46.44	55 - 215
`SpecTNT-TCN`	`Offline`	Hung et al. (2022)	88.7	–	–
`Transformer`	`Offline`	Zhao et al. (2022)	88.5	–	–
`TCN`	`Offline`	Böck and Davies (2020)	88.5	–	–

Table 2 Comparing various low-latency online beat trackers under specific conditions (`C1`, . . . , `C5`) and against existing literature for beat performance, latency, and tempo range, utilizing the `GTZAN` dataset. A tempo range of ±40% of the average track tempo is denoted by `TR40` and ground truth activation by `GT`.

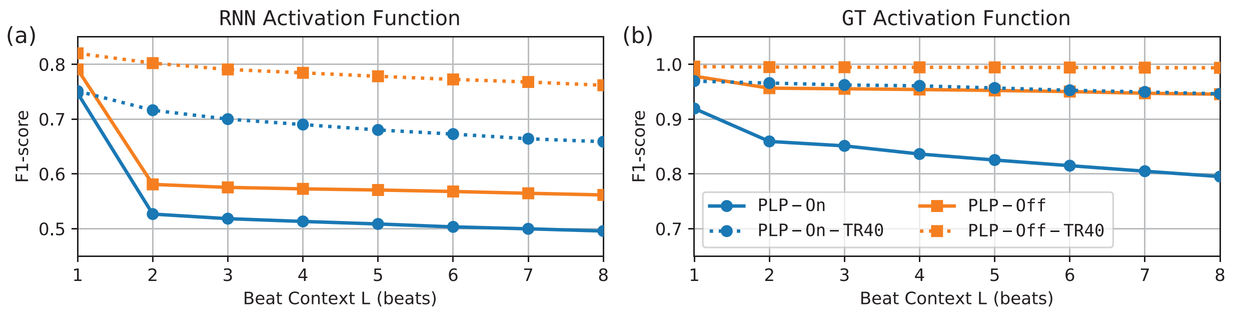


Figure 4 F1-score and L-correct metric for different activation functions and various post-processing methods on the `GTZAN` dataset. A tempo range of ±40% average track tempo is denoted by `TR40`.

notice a much smaller decrease in F1-score from 75.11% (no context) to 71.61% ($L = 2$). Furthermore, for all other consecutive beats from $L = 2$ to $L = 8$, the F1-scores remain consistently higher compared to `RNN-PLP-On`, with only a slight decrease of approximately 5.74% (from 71.61% to 65.87%). Similarly, for the activation functions `GT` depicted in Figure 4b, we observe consistent F1-scores, particularly evident for the tempo range `TR40`. This suggests that a notable improvement in context-sensitive beat performance can be achieved if the average tempo of the analyzed music is known and utilized for specific beat tracking tasks, thus highlighting the tempo range as a valuable hyperparameter for controlling real-time applications.

6.3 KERNEL SIZE EVALUATION

In Figure 5, we assess our real-time procedure across various kernel sizes. Considering the best possible activation function `GT` in Figure 5b, we observe consistent behavior across all datasets. When the kernel size is above 6 s, the F1-score remains stable; however, when it falls below 6 s, the F1-score starts to drop significantly. Based on this observation, we opted to fix the kernel size at 6 s for our experiments, aiming to find a balance between stability in beat estimation and responsiveness to tempo changes in the music. A more realistic scenario is depicted with the activation function RNN in Figure 5a. In this case, we observe minimal variation in F1-score across different kernel sizes across various datasets. The F1-score begins to decrease only when the kernel sizes fall below 3 s. This suggests that our real-time procedure can effectively accommodate a wide range of kernel sizes, enabling adjustment of the beat context with minimal impact on F1-score.

6.4 LOOKAHEAD IMPACT ANALYSIS

We now discuss the impact of the lookahead parameter on the beat detection performance and start with Figure 6a, focusing on RNN and GTZAN as an example. The corresponding data is reported in Table 3. For a lookahead of 0 frames (0 ms), the F1-score is at 74.72%, which we already reported for RNN-PLP-On in Table 2. Using a lookahead of 1 frame (11.6 ms), there is only a small difference (−0.04%) in F1-score for a total of 74.68%. Using a 10-frame lookahead (116.1 ms) results in 74.31% (−0.41%), which is useful for compensating network and processing delays, as demonstrated in Section 7.1. With 50 frames (580.5 ms), comparable to the inter beat interval size (500 ms) at 120 BPM, the F1-score drops to 72.39% (−2.33%). At 100 frames (1161.0 ms), the F1-score falls to 69.49% (−5.23%), and at 200 frames (2322.0 ms), it declines significantly to 61.54%

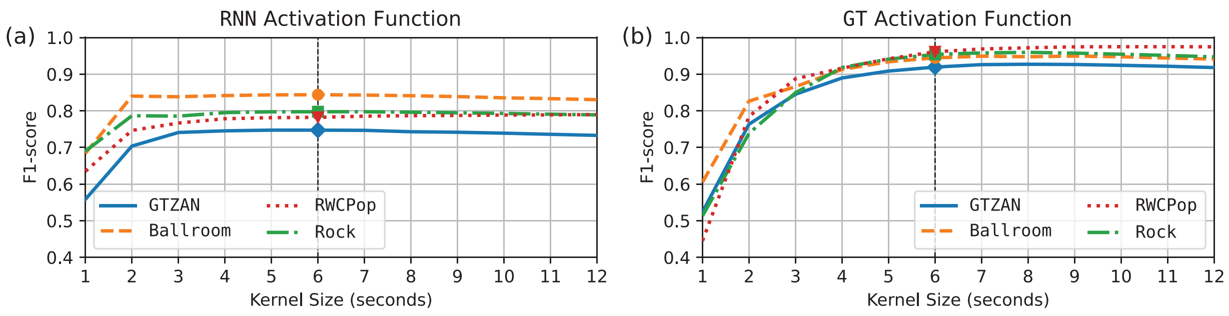


Figure 5 F1-score for different kernel sizes of PLP-On across various activation functions on different datasets.

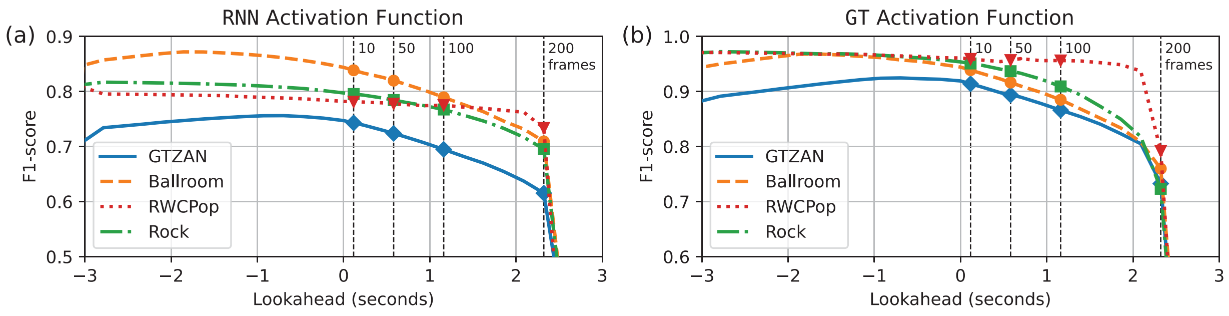


Figure 6 F1-score of various settings for lookahead of PLP-On for various activation functions across different datasets, see Table 3 for numbers.

Settings		F1-score (%) vs. Lookahead					
Lookahead in frames (ms)		0 (0.0)	1 (11.6)	10 (116.1)	50 (580.5)	100 (1161.0)	200 (2322.0)
RNN	GTZAN	74.72	74.68 (−0.04)	74.31 (−0.41)	72.39 (−2.33)	69.49 (−5.23)	61.54 (−13.18)
	Ballroom	84.39	84.34 (−0.05)	83.87 (−0.52)	81.98 (−2.41)	78.95 (−5.44)	70.94 (−13.45)
	RWCPop	78.22	78.21 (−0.01)	78.15 (−0.07)	77.80 (−0.42)	77.52 (−0.70)	73.38 (−4.84)
	Rock	79.74	79.72 (−0.02)	79.55 (−0.20)	78.45 (−1.29)	76.74 (−3.00)	69.55 (−10.19)
GT	GTZAN	91.93	91.87 (−0.05)	91.40 (−0.52)	89.39 (−2.54)	86.62 (−5.31)	73.27 (−18.65)
	Ballroom	94.44	94.39 (−0.06)	93.89 (−0.55)	91.63 (−2.81)	88.51 (−5.93)	75.98 (−18.46)
	RWCPop	96.10	96.08 (−0.02)	95.92 (−0.18)	95.71 (−0.39)	95.64 (−0.46)	79.20 (−16.90)
	Rock	95.39	95.36 (−0.03)	95.10 (−0.28)	93.70 (−1.69)	90.93 (−4.45)	72.33 (−23.06)

Table 3 The F1-score of lookahead settings in frames (and milliseconds) of PLP-On for different activation functions across different datasets, with each F1-score accompanied by the difference (in parenthesis) to the zero lookahead.

(−13.18%). Overall, the impact of lookahead on the F1-score is relatively small, especially for values below 50 frames (580.5 ms), emphasizing its importance as a valuable parameter for controlling and compensating latency in real-time beat tracking systems.

Comparing different datasets for lookahead performance, we observe a correlation with the tempo deviation of each dataset (see Table 1), which indicates the level of tempo variation across all songs. The dataset `RWCPop` exhibits the highest stability (97.9 ± 10.8%) and the lowest tempo deviation (±27.3 BPM) among the datasets analyzed, experiencing the smallest drop in F1-score (−4.84%) at 200 frames lookahead. In contrast, at 200 frames lookahead, `Rock` (±34.7 BPM), `GTZAN` (±39.6 BPM), and `Ballroom` (±39.7 BPM) experience drops in F1-score of −10.19%, −13.18%, and −13.45%, respectively. This suggests that lookahead can be employed with minor impact on beat detection performance, particularly for highly stable music genres such as Pop. This is particularly evident in the case of `GT`, Figure 6b, where the `RWCPop` dataset demonstrates stable F1-scores for up to 2 s of lookahead, distinguishing itself from other datasets.

Note that all datasets in Figure 6 show higher F1-scores for negative lookahead (adding latency to make a more accurate decision by waiting for future data). However, since our focus is more on compensating for latency rather than adding latency for real-time applications, we do not discuss this fact any further.

7 APPLICATIONS

In this section, we present two application prototypes for interactive music making (Section 7.1) and educational music gaming (Section 7.2) that utilize the system output of our real-time beat tracker in a creative way.

7.1 INTERACTIVE MUSIC MAKING

We start off with a demo for interactive music making, closely following the work by Meier et al. (2021). The terminal application, named "Beat Command Line Interface" (`beatcli.py`), is implemented in Python and built upon the real-time procedure detailed in Section 3.

With Figure 7, we show a block diagram of the application. It accepts input arguments (A) to select a device/channel of the audio input (B) and configure various parameters required for the audio analysis (C). It continuously executes a complete PLP real-time procedure and uses data buffering (Section 3.4) to update the PLP model for every new frame of audio provided by the input audio streaming (Section 3.2). Upon beat detection (Section 4.1) within the current time frame, both terminal output (D) and network output (E) with corresponding local pulse information are provided, as shown

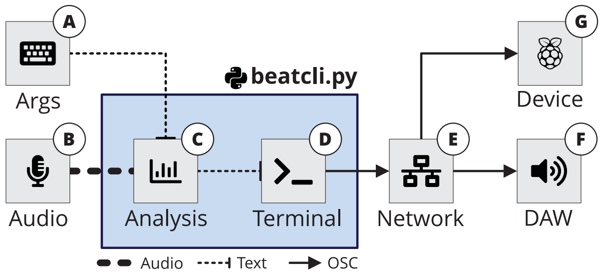
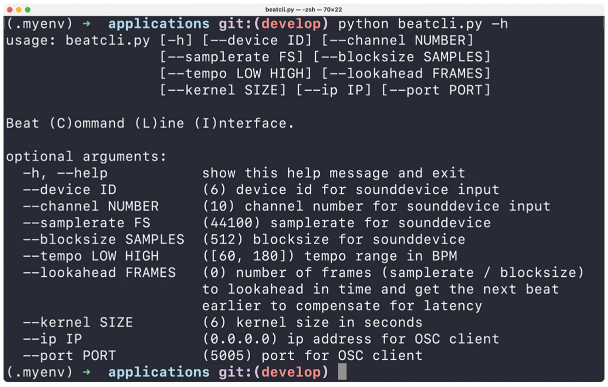


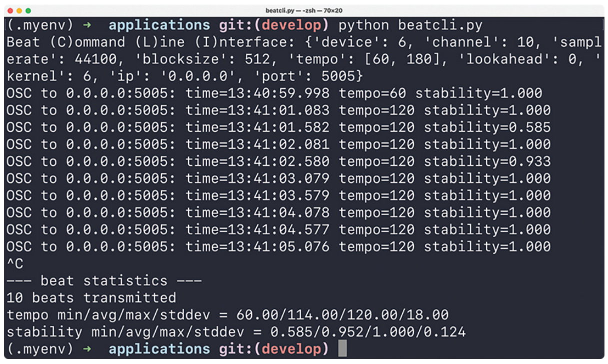
Figure 7 A block diagram of the `beatcli.py` terminal application. (A) Input arguments. (B) Audio input. (C) Audio analysis. (D) Terminal output. (E) Network output. (F) Receiving software. (G) Receiving hardware.


```
beatcli.py — -zsh — 70×22
(.myenv) ➜  applications git:(develop) python beatcli.py -h
usage: beatcli.py [-h] [--device ID] [--channel NUMBER]
                  [--samplerate FS] [--blocksize SAMPLES]
                  [--tempo LOW HIGH] [--lookahead FRAMES]
                  [--kernel SIZE] [--ip IP] [--port PORT]

Beat (C)ommand (L)ine (I)nterface.

optional arguments:
  -h, --help            show this help message and exit
  --device ID           (6) device id for sounddevice input
  --channel NUMBER      (10) channel number for sounddevice input
  --samplerate FS       (44100) samplerate for sounddevice
  --blocksize SAMPLES   (512) blocksize for sounddevice
  --tempo LOW HIGH      ([60, 180]) tempo range in BPM
  --lookahead FRAMES    (0) number of frames (samplerate / blocksize)
                        to lookahead in time and get the next beat
                        earlier to compensate for latency
  --kernel SIZE         (6) kernel size in seconds
  --ip IP               (0.0.0.0) ip address for OSC client
  --port PORT           (5005) port for OSC client
(.myenv) ➜  applications git:(develop) 
```
Figure 8 The help function of the `beatcli.py` application with information about input arguments.

in Figure 9. This system output (Section 4) can be received by software clients (F), such as a Digital Audio Workstation (DAW), or hardware devices (G), such as microcontrollers, to utilize the transmitted local pulse information.

The help function of `beatcli.py` (Figure 8) offers detailed explanations of the input arguments. The real-time audio streaming (Section 3.2) relies on the Python module `sounddevice`,[3] which receives the first four arguments to specify the desired hardware. The next three arguments control the settings of the beat tracker, including the `tempo` range in BPM to set the pulse level, the `lookahead` (Section 4.3) in `FRAMES` to compensate for latency, and the `kernel SIZE` in seconds to determine the beat context (Section 4.2). Last, two arguments configure the network output. `beatcli.py` serves as an Open Sound Control (OSC) server, sending pulse information over the network to a client with a specified `IP` address and `PORT` number.

With Figure 9, we illustrate the terminal output of `beatcli.py`, detailing the transmitted pulse information. Upon application launch, an overview of default settings is provided. Below this follows a table of detected beats, each row containing columns with local pulse information. Initially, the `IP` address and `PORT` of the OSC message being sent out are displayed. Subsequently, a `time` value provides a timestamp of when the beat detection (Section 4.1) occurred. Following this, a `tempo` value indicates the local tempo computed from the inter



```
(.myenv) ➜  applications git:(develop) python beatcli.py
Beat (C)ommand (L)ine (I)nterface: {'device': 6, 'channel': 10, 'sampl
erate': 44100, 'blocksize': 512, 'tempo': [60, 180], 'lookahead': 0, '
kernel': 6, 'ip': '0.0.0.0', 'port': 5005}
OSC to 0.0.0.0:5005: time=13:40:59.998 tempo=60 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:01.083 tempo=120 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:01.582 tempo=120 stability=0.585
OSC to 0.0.0.0:5005: time=13:41:02.081 tempo=120 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:02.580 tempo=120 stability=0.933
OSC to 0.0.0.0:5005: time=13:41:03.079 tempo=120 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:03.579 tempo=120 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:04.078 tempo=120 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:04.577 tempo=120 stability=1.000
OSC to 0.0.0.0:5005: time=13:41:05.076 tempo=120 stability=1.000
^C
--- beat statistics ---
10 beats transmitted
tempo min/avg/max/stddev = 60.00/114.00/120.00/18.00
stability min/avg/max/stddev = 0.585/0.952/1.000/0.124
(.myenv) ➜  applications git:(develop) 
```

Figure 9 The terminal output of the `beatcli.py` application showing the system in action.

beat interval (Section 4.5). Finally, a value for beat stability (Section 4.4) is displayed. A value of 1.0 represents a beat with a steady tempo and maximum stability, while values close to zero signify a relatively unstable tempo and beat structure.

To demonstrate a real-world use case of the application, we describe a real-time accompaniment system using `beatcli.py`. Suppose we are playing an instrument and intend to utilize our beat tracker to trigger samples in a DAW synchronized with the beat of our music. The stability parameter allows us to control the volume, ensuring that samples are only audible when beats are stable. With every beat we update the global tempo of the DAW, thereby adjusting the playback speed of the samples to match the tempo of our playing. With the tempo range settings, we can limit the beat triggers to certain tempo octaves, accommodating different tempo variations such as normal, or double tempo. The kernel size parameter allows us to adjust the overall beat context, making the tracker either more stable or more responsive to tempo changes. Finally, the lookahead value enables us to fine-tune the system to compensate for any noticeable network and processing delays, ensuring that the playback feels natural and not lagged.

7.2 EDUCATIONAL MUSIC GAMING

In this section, we introduced an application for educational music gaming, closely following the work by Meier et al. (2022).

We present a prototype jump-and-run game named "Rock Your Beats," which is illustrated in Figure 10. The player's goal is to tap along in sync with the beat of the music, by pressing a button on a touch screen, keyboard, or gaming controller. In doing so, the player must aim to hit moving "beat creatures" with a dropping rock positioned at the center of the game world. Each beat creature represents one beat, and one point is awarded for each hit.

The game world is generated in real-time from music in the player's environment by using the streamed audio

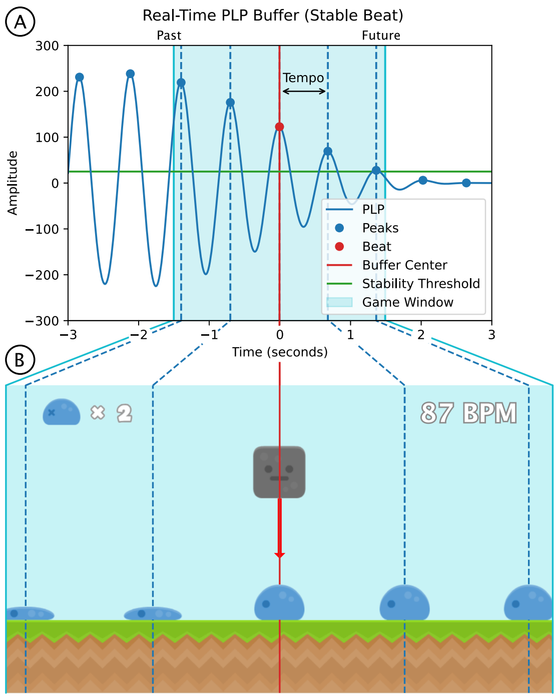


Figure 10 The educational music game "Rock Your Beats" (bottom) with the corresponding real-time PLP buffer (top), used to derive the positions of "beat creatures" in the game world.

(Section 3.2) of a microphone input signal. The center of the game world (the rock) represents the current point in time and thus relates to the center of the PLP buffer (at second zero), as discussed in Section 3.3. The beat creatures are placed at the peak positions of the PLP curve, using the beat context (Section 4.2). Therefore, the beat creatures follow the same movement from the right (future) to the left (past) positions as the peaks of the PLP curve and cross the central player position in sync with the beat of the music, as discussed in Section 4.1. If no stable beat structure is detected in the input signal, the peaks of the PLP curve have a rather low amplitude, as explained in Section 4.4. If this amplitude is below a certain stability threshold, no beat creatures are created in the game world, to avoid unpredictable and noisy beat illustrations. As a consequence, beat creatures only appear if music with a stable beat is played. The time range that is visually represented in the game, the game window, is closely related to the beat context (Section 4.2). In this way, the game window can also be chosen smaller to avoid the visualization of inaccurate beat predictions in the distant future and to determine how much visual support the player gets. The inter beat interval (Section 4.5) can be used to display the current local tempo in the game. Finally, with the beat lookahead (Section 4.3), the game can be adjusted to compensate for latency that might occur with the game visualization and controller input delays. For a more detailed discussion of this gaming application, we refer to Meier et al. (2022).

8 CONCLUSIONS

In this paper, we introduced a real-time beat tracking system designed to deliver zero latency and enhanced controllability for interactive music applications. In addition to beat detection, our model generates valuable supplementary outputs, including beat context, beat stability, and inter beat interval analysis for local tempo estimation. Leveraging the beat lookahead technique, our method effectively compensates for latency by up to several hundred milliseconds in real-time audio systems. Furthermore, our model demonstrates enhanced controllability, allowing real-time applications to adjust latency compensation, pulse level, and beat context. This versatility proves particularly beneficial for real-time beat tracking tasks, where maintaining synchronization between the analyzed input audio stream and the produced output audio stream is crucial, thereby preventing any noticeable delay in audio perception. We validated and tested the capability of our model through experiments and two real-world scenario applications focused on interactive music making and educational music gaming. As a result, our model serves as a practical and lightweight tool for musicians to fine-tune their real-time audio setups and achieve the desired latency perception, while also opening up new creative controllability for real-time beat tracking applications.

For future research, our goals include conducting additional experiments, such as evaluating the lookahead feature of our model on smaller tolerance windows and investigating beat performance on more challenging datasets. Additionally, we aim to enhance our real-time pipeline by integrating newly developed beat activation models into our post-processing method as they become available in the future. Recognizing the practical utility of our system, we are committed to continuing the development of interactive applications and demonstrations. Specifically, we plan to create a real-time beat tracking audio plugin to enable musicians to use our method for studio mixing or interactions live on stage.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers and the section editor for their helpful and constructive comments. The International Audio Laboratories Erlangen is a joint institution of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and Fraunhofer Institute for Integrated Circuits IIS.

FUNDING INFORMATION

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Grant No. 500643750 (MU 2686/15-1) and the National Science and Technology Council (NSTC), Taiwan, under Grant No. NSTC 112-2917-I-564-011.

COMPETING INTERESTS

Meinard Müller is a Co-Editor-in-Chief of the *Transactions of the International Society for Music Information Retrieval*. He was removed completely from all editorial decisions. The authors have no other competing interests to declare.

AUTHOR CONTRIBUTIONS

Peter Meier was the main contributor to writing the article, developing the real-time beat tracking system, running the experiments, and creating the applications. Ching-Yu Chiu shared her expertise on beat tracking and helped set up the datasets and experiments. Meinard Müller supervised the work and contributed to writing the article.

NOTES

1. https://audiolabs-erlangen.de/resources/MIR/2024-TISMIR-RealTime PLP
2. Note that the input to the PLP method can include both novelty functions (such as spectral flux) and activation functions (such as the output of a recurrent neural network).
3. https://pypi.org/project/sounddevice

AUTHOR AFFILIATIONS

Peter Meier https://orcid.org/0000-0002-3094-1931
International Audio Laboratories Erlangen, Am Wolfsmantel 33, 91058 Erlangen, Germany

Ching-Yu Chiu https://orcid.org/0000-0002-3671-8474
International Audio Laboratories Erlangen, Am Wolfsmantel 33, 91058 Erlangen, Germany

Meinard Müller https://orcid.org/0000-0001-6062-7524
International Audio Laboratories Erlangen, Am Wolfsmantel 33, 91058 Erlangen, Germany

REFERENCES

Böck, S., and Davies, M. E. P. (2020). Deconstruct, analyse, reconstruct: How to improve tempo, beat, and downbeat estimation. In **J. Cumming, J. H. Lee, B. McFee, M. Schedl, J. Devaney, C. McKay, E. Zangerle, and T. de Reuse** (Eds.), *Proceedings of the 21th International Society for Music Information Retrieval Conference (ISMIR)*, Montréal, Canada, pp. 574–582.

Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F., and Widmer, G. (2016a). Madmom: A new Python audio and music signal processing library. In *Proceedings of the ACM International Conference on Multimedia (ACM-MM)*, Amsterdam, The Netherlands, pp. 1174–1178.

Böck, S., Krebs, F., and Widmer, G. (2014). A multimodel approach to beat tracking considering heterogeneous music styles. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, pp. 603–608.

Böck, S., Krebs, F., and Widmer, G. (2016b). Joint beat and downbeat tracking with recurrent neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, New York City, New York, USA, pp. 255–261.

Böck, S., and Schedl, M. (2011). Enhanced beat tracking with context-aware neural networks. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, Paris, France, pp. 135–139.

Chang, C.-C., and Su, L. (2024). Beast: Online joint beat and downbeat tracking based on streaming transformer. In *ICASSP 2024–2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seoul, Korea, pp. 396–400.

Clercq, T., and Temperley, D. (2011). A corpus analysis of rock harmony. *Popular Music*, *30*, 47–70.

Davies, E. P. M., and Böck, S. (2019). Temporal convolutional networks for musical audio beat tracking. In *27th European Signal Processing Conference (EUSIPCO)*, A Coruña, Spain, pp. 1–5. IEEE.

Dixon, S., and Cambouropoulos, E. (2000). Beat tracking with musical knowledge. In **W. Horn** (Ed.), *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence*, pp. 626–630. IOS Press.

Durand, S., Bello, J. P., David, B., and Richard, G. (2015). Downbeat tracking with multiple features and deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015*, South Brisbane, Queensland, Australia, pp. 409–413. IEEE.

Ellis, D. P. (2007). Beat tracking by dynamic programming. *Journal of New Music Research*, *36*(1), 51–60.

Goto, M., Hashiguchi, H., Nishimura, T., and Oka, R. (2002). RWC music database: Popular, classical and jazz music databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, France, pp. 287–288.

Gouyon, F., Klapuri, A. P., Dixon, S., Alonso, M., Tzanetakis, G., Uhle, C., and Cano, P. (2006). An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(5), 1832–1844.

Grosche, P., and Müller, M. (2011). Extracting predominant local pulse information from music recordings. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(6), 1688–1701.

Heydari, M., Cwitkowitz, F., and Duan, Z. (2021). Beatnet: Crnn and particle filtering for online joint beat downbeat and meter tracking. In *22th International Society for Music Information Retrieval Conference (ISMIR)*, Online.

Heydari, M., McCallum, M., Ehmann, A., and Duan, Z. (2022). A novel 1d state space for efficient music rhythmic analysis. In *ICASSP 2022–2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 421–425. IEEE.

Holzapfel, A., and Grill, T. (2016). Bayesian meter tracking on learned signal representations. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR),* New York City, New York, USA, pp. 262–268.

Hung, Y., Wang, J., Song, X., Lu, W. T., and Won, M. (2022). Modeling beats and downbeats with a time-frequency transformer. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Virtual and Singapore, pp. 401–405.

Klapuri, A. P., Eronen, A. J., and Astola, J. (2006). Analysis of the meter of acoustic musical signals. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(1), 342–355.

Krebs, F., Böck, S., Dorfer, M., and Widmer, G. (2016). Downbeat tracking using beat synchronous features with recurrent neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR),* New York City, New York, USA, pp. 129–135.

Krebs, F., Böck, S., and Widmer, G. (2015). An efficient state-space model for joint tempo and meter tracking. In **M. Müller and F. Wiering** (Eds.), *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, pp. 72–78.

Meier, P., Krump, G., and Müller, M. (2021). A realtime beat tracking system based on predominant local pulse information. In *Demos and late breaking news of the International Society for Music Information Retrieval Conference (ISMIR)*, Online.

Meier, P., Schwär, S., Rosenzweig, S., and Müller, M. (2022). Real-Time MIR algorithms for music-reactive game world generation. In *Mensch und Computer 2022 - Workshopband*, Gesellschaft für Informatik e.V.

Müller, M. (2021). *Fundamentals of music processing using Python and Jupyter notebooks* (2nd ed.). Springer Verlag.

Raffel, C., McFee, B., Humphrey, E. J., Salamon, J., Nieto, O., Liang, D., and Ellis, D. P. W. (2014). MIR_EVAL: A transparent implementation of common MIR metrics. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, pp. 367–372.

Schreiber, H., Zalkow, F., and Müller, M. (2020). Modeling and estimating local tempo: A case study on Chopin's mazurkas. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Montréal, Canada, pp. 773–779.

Stefani, D., and Turchet, L. (2022). On the challenges of embedded real-time music information retrieval. In

Proceedings of the 25th International Conference on Digital Audio Effects (DAFx20in22), pp. 177–184.

Tzanetakis, G., and Cook, P. R. (2002). Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, *10*(5), 293–302.

Zhao, J., Xia, G., and Wang, Y. (2022). Beat transformer: Demixed beat and downbeat tracking with dilated self-attention. In *Proceedings of the 23rd International Society for Music Information Retrieval Conference (ISMIR)*, Bengaluru, India, pp. 169–177.

TO CITE THIS ARTICLE:
Meier, P., Chiu, C.-Y., & Müller, M. (2024). A Real-Time Beat Tracking System with Zero Latency and Enhanced Controllability. *Transactions of the International Society for Music Information Retrieval*, *7*(1), 213–227. DOI: https://doi.org/10.5334/tismir.189

Submitted: 2 March 2024 **Accepted:** 1 August 2024 **Published:** 1 October 2024