Tempo- and Time-Signature–Responsive Real-Time EEG Processing Architecture for Multimodal Music Analysis

Oğuzhan Tuğral

Abstract

This work introduces a tempo- and time-signature–responsive real-time EEG processing system built on OpenBCI hardware, Node.js networking, and browser-based JavaScript analysis. The system acquires 8-channel EEG via the ADS1299 front-end, which is a high-resolution biopotential amplifier, performs digital filtering in the OpenBCI GUI, streams data over UDP to a WebSocket server, and computes spectral features through a tempo-adaptive windowing mechanism whose duration is dynamically derived from musical tempo and meter. A circular buffer ensures continuous sample flow, and sliding-window DFT enables high-resolution spectral tracking. The system achieves stable end-to-end latency of approximately 6–11 ms, maintains synchronized EEG–music alignment across tempo changes, and generates interpretable delta, theta, alpha, beta, and gamma band trajectories for real-time visualization. By coupling EEG analysis windows directly to musical tempo and time signature, this framework provides a novel foundation for real-time multimodal corpus creation and opens new possibilities for studying neural entrainment, musical structure perception, and interactive performance systems.

November 16, 2025

Contents

1 Basics of Digital Signal Processing **3**
1.1 Shannon–Weaver Communication Model . . . 4
1.2 Analysis and Synthesis . . . 4
1.2.1 Sampling . . . 5
1.2.2 Quantization . . . 6

2 EEG Signal Processing: Hardware, Software, and Data Transfer **9**
2.1 Hardware Acquisition (ADS1299) . . . 10
2.2 OpenBCI GUI Digital Signal Processing . . . 12
2.2.1 Inverse Scaling to Microvolts . . . 13
2.2.2 Digital Filtering . . . 14
2.2.3 Common Filters Used in OpenBCI GUI . . . 14
2.2.4 Basic Signal Statistics . . . 15
2.2.5 timeSeriesRaw vs. timeSeriesFilt . . . 16
2.3 UDP Network Streaming . . . 18
2.3.1 UDP-Based Multi-Port and Multi-Channel EEG Streaming . . . 19
2.3.2 Network Packet Handling in Real-Time EEG Streaming . . . 20

3 Browser-Side EEG Spectral Analysis and Visualization **24**
3.1 Data Reception and Parsing . . . 25
3.1.1 WebSocket Message . . . 25
3.1.2 Parse Value . . . 26
3.1.3 Update Display . . . 26
3.1.4 Extract Sample . . . 27
3.2 Circular Buffer Management . . . 27
3.2.1 Store in Array . . . 27
3.3 Tempo-Adaptive Windowing . . . 29
3.3.1 Tempo and Time Signature Retrieval . . . 29
3.3.2 Window Size Calculation . . . 30
3.3.3 Window Boundary Detection . . . 30
3.4 Spectral Analysis via Discrete Fourier Transform . . . 31
3.4.1 Process Segment . . . 32
3.4.2 Compute DFT . . . 33
3.4.3 Extract Bands . . . 34
3.4.4 Sliding Windows . . . 35
3.5 Power Calculation and Canvas Visualization . . . 35
3.5.1 Select Last Window . . . 36

3.5.2 Calculate Power Bands 36
3.5.3 Draw Canvas 37
3.6 Summary of Browser-Side Pipeline 38

1 Basics of Digital Signal Processing

Building on the challenges outlined in the first section, this part introduces *tempo- and time-signature–responsive EEG windows* as a proposed contribution toward solving Problem 1[1]. This approach integrates harmonic and musical analysis with insights from music and health sciences, enabling a real-time tool that adapts dynamically to musical structure and physiological responses.

The basic concepts of Digital Signal Processing (DSP) form the foundation for understanding modern biosignal acquisition. A signal is a representation of how a physical quantity varies over time or space. Examples include electrical brain activity measured through EEG as a function of time, air-pressure fluctuations corresponding to sound waves, and light intensity distributed across a two-dimensional array in an image. Signals underlie all information-transmission and processing systems, serving as carriers of data from sensors, biological systems, communication channels, and countless other sources.

Extending these fundamental principles, the following section examines in greater detail how signals are processed, transformed, and interpreted within the context of real-time multimodal analysis.

[1] This section constitutes the second chapter of my ongoing master's thesis and builds upon the findings presented in the first chapter. That earlier section concluded that recent interdisciplinary music- and health-science research faces two major, interrelated challenges.

First, music-and-memory studies rarely incorporate detailed harmonic or structural musical analysis. This limitation stems from a second, deeper problem in both Music Information Retrieval (MIR) and music theory: current technologies used in music-and-memory research rely almost entirely on expert-annotated datasets and lack integrated, real-time systems capable of automatically tokenizing musical structure while simultaneously synchronizing it with behavioral, neural, and physiological multimodal measurements. (Details of the first chapter can be found at: https://oguzhantugral.com/research/musicTheory/dataAnalysisMusicHealth.html.)

Accordingly, while the present work aims to contribute to a potential solution for Problem 1, the sample project *Real-Time Roman Numeral Analysis from Live MIDI Performance Using a Distributed Multimodal Architecture* represents a contribution toward solving Problem 2.

1.1 Shannon–Weaver Communication Model

Before introducing the fundamental operations of signal processing, it is instructive to recall the Shannon–Weaver communication model, which provides a conceptual framework for understanding how information is generated, transmitted, distorted by noise, and recovered. This model highlights the entire lifecycle of a signal—from its origin at the information source to its final interpretation at the destination—and thereby connects directly to the motivation behind signal processing itself.

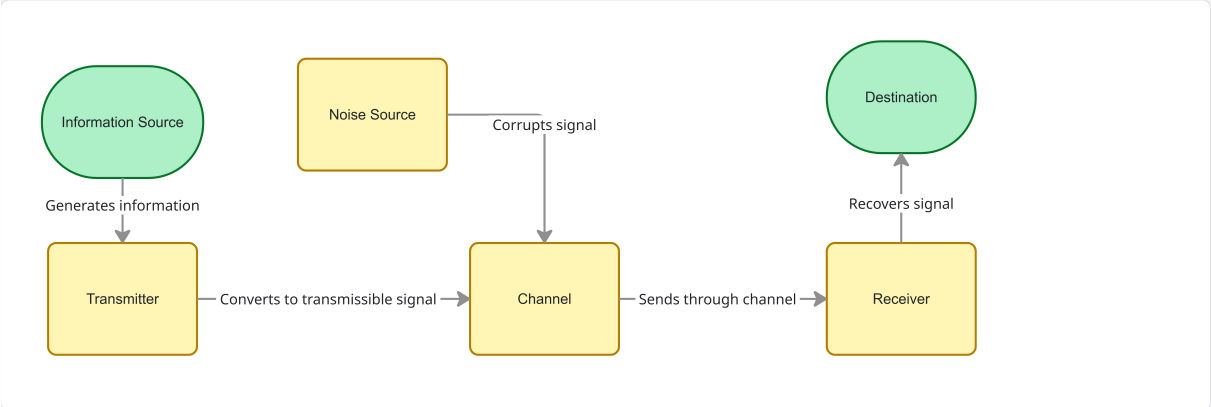

Figure 1: Shannon–Weaver communication model.

From a signal processing perspective, Fig. 1 emphasizes two critical observations. First, the information source generates the physical signal—such as brain activity, sound waves from an instrument, or visual patterns—that symbolize the information to be conveyed. Second, as the signal propagates through the channel, it becomes distorted through attenuation and the addition of noise. Signal processing emerges precisely as the mathematical and algorithmic framework designed to analyze such corrupted signals, extract the underlying information, and synthesize new signals for reliable communication and computation. Thus, the two key concepts—analysis and synthesis—must be introduced to clarify their roles within the broader context of signal processing.

1.2 Analysis and Synthesis

Signal processing encompasses two fundamental operations. The first is analysis, which involves understanding or extracting information from signals, including their features,

structure, and underlying patterns. Examples include analyzing EEG signals to detect neural rhythms, extracting spectral content from sound waves, or identifying edges in images. The second operation is synthesis, which involves generating signals that convey intended information, such as producing speech waveforms, musical tones, or control signals. Together, these operations form the foundation of modern communication, sensing, and computational systems.

Analog models describe signals as continuous functions of time or space, representing physical quantities that vary smoothly without discrete steps. In contrast, digital models represent signals as sequences of discrete numerical values that approximate the continuous waveform. This representation enables efficient storage, processing, and transmission using unified algorithms across digital hardware and software systems. The transition from analog to digital fundamentally transforms how signals can be manipulated: digital signals are built upon two ingredients—discrete time (sampling) and discrete amplitude (quantization).

1.2.1 Sampling

Sampling converts a continuous-time signal into a discrete sequence by measuring its value at uniformly spaced time instants:

$$x[n] := x(nT_s), \qquad n \in \mathbb{Z}.$$

Here, T_s is the sampling period and $f_s = 1/T_s$ is the sampling rate in samples per second. For example, if $T_s = 0.004\,\mathrm{s}$, then $f_s = 250\,\mathrm{Hz}$, and the system acquires 250 discrete samples per second. Each discrete value $x[n]$ represents the amplitude of the continuous-time signal at the instant $t = nT_s$. In other words, while n is a discrete index of the digitized signal (analogous to an array index), its multiplication with the sampling period gives the corresponding time value in the original analog signal. This relationship forms the essential link between continuous-time and discrete-time representations.

Having established the basic mechanism that maps continuous-time signals to their discrete-

time counterparts, we now turn to the fundamental principle that determines the conditions under which this conversion preserves all the information in the original signal. This provides the proper representation of the signal in the time domain, and before examining how quantization introduces additional distortions, it is helpful to consider how the sampling theorem directly influences the design and operation of EEG acquisition hardware.

For EEG applications, neural signals contain meaningful information primarily below 100 Hz, although high-frequency muscle artifacts and environmental noise can extend well beyond this range. The OpenBCI Cyton system samples at $f_s = 250\,\text{Hz}$, which satisfies the Nyquist criterion for signals bandlimited to 125 Hz. To ensure that frequency components above this limit do not cause aliasing, the ADS1299 analog front-end incorporates a hardware anti-aliasing filter before digitization, typically with a cutoff frequency near $f_s/2 \approx 125\,\text{Hz}$. This filter attenuates high-frequency content that would otherwise violate the sampling theorem, ensuring that the discrete samples faithfully represent the underlying EEG signal.

1.2.2 Quantization

The second step of digitization, *quantization*, maps each sample to one of L allowable *amplitude levels*, producing integer-valued data $\hat{x}[n]$. Whereas continuous amplitudes may take any real value, quantization restricts each sample to a finite set of levels, enabling digital storage and computation. Together, sampling and quantization transform an analog signal into a digital sequence.

While the sampling theorem explains how a clean, bandlimited signal can be captured without loss of information, real-world communication systems rarely operate under ideal, noise-free conditions. In the *Shannon–Weaver communication model*, noise enters the channel as an additive disturbance,

$$y(t) = x(t) + w(t),$$

where $y(t)$ is the measured signal formed by combining the clean signal $x(t)$ with random noise $w(t)$ arising from thermal effects, interference, and other disturbances. Such noise reduces the achievable data rate and the reliability of communication, and its influence differs substantially between analog and digital transmission systems.

In analog communication, noise accumulates continuously as the signal propagates through the channel. Each amplifier or repeater passes forward a slightly more corrupted version of the waveform, and no mechanism exists to restore the signal perfectly. By contrast, digital communication introduces a decision stage at every repeater: received symbols are compared against thresholds, and the nearest ideal symbol is regenerated. Because the digital repeater outputs clean, quantized symbols, noise does not accumulate indefinitely; instead, it is periodically removed at each decision point. However, because the present work focuses on the mechanisms underlying software for generating and processing multimodal data—including EEG measurements—it is more appropriate to consider modern biosignal acquisition systems rather than classical communication channels, where repeaters and symbol decisions play a central role.

Devices such as the OpenBCI Cyton board follow a fundamentally different architecture: rather than transmitting symbols over a noisy communication channel that would require digital regeneration, they condition the analog EEG signal locally—using differential amplification, analog filtering, and gain control—and then convert it directly into digital samples for short-range digital transmission. For example, each EEG channel measures the voltage difference between an active scalp electrode and a reference electrode, amplifies and filters this microvolt-level signal, and digitizes it before sending it wirelessly to a computer. Once digitized, these samples are transmitted as discrete integer packets rather than as analog waveforms. Thus, unlike classical analog chains, there is no progressive accumulation of noise during transmission: all critical noise-sensitive operations occur before digitization, and all subsequent communication is effectively noise-free except for occasional packet loss.

Consequently, the notion of a "channel" in the Shannon–Weaver sense plays a reduced role

in such systems. For biosignal acquisition hardware, the primary challenge lies in ensuring low-noise analog conditioning prior to sampling, not in symbol regeneration during transmission. For this reason, the following section focus on the internal signal-processing pipeline of the acquisition device itself, with emphasis on amplification, filtering, sampling, and quantization in the ADS1299-based OpenBCI Cyton system.

2 EEG Signal Processing: Hardware, Software, and Data Transfer

Within the present software framework developed in this work, multimodal synchronized data are generated by integrating EEG-derived features that are mapped to musical representations—such as automatically detected Roman numeral harmonic analyses—as well as behavioral markers including facial-expression tracking, go/no-go performance metrics, and estimates of valence and emotion. The EEG signals are acquired using an OpenBCI Cyton device. Accordingly, this section provides a technical overview of the Cyton 8-channel EEG data acquisition and processing pipeline, which converts cortical potentials into digital data through three principal stages: (1) hardware acquisition via the ADS1299 analog front-end, (2) digital processing in the OpenBCI GUI, and (3) network transmission via the UDP protocol. The OpenBCI Cyton 8-channel EEG system thus provides a complete pathway from tiny scalp voltages to real-time digital data that can be consumed by other software modules.

When neurons fire, they generate electrical signals on the order of 20–100 V— around 10,000 times weaker than a 1.5 V AA battery—so the first challenge is simply detecting them. In the hardware stage, electrodes pick up these faint potentials, and the ADS1299 chip on the Cyton board differentially measures, amplifies, filters, and converts them into high-resolution digital codes. In the software stage, the OpenBCI GUI converts these codes back into voltage values, applies standard digital signal processing (such as band-pass and notch filtering), and exposes both raw and filtered data streams for inspection and analysis. In the network stage, the GUI uses low-latency UDP streaming to send multi-channel EEG data in JSON format to environments like Python, MaxMSP, or JavaScript, enabling real-time neurofeedback, interaction, and multimodal performance. The following sections explain each of these stages in detail, linking hardware behavior, DSP fundamentals, and network streaming.

2.1 Hardware Acquisition (ADS1299)

The Cyton board uses a specialized chip called the ADS1299, manufactured by Texas Instruments. This chip is specifically designed for biological signals and provides eight input channels for simultaneous multi-site measurement, a 24-bit analog-to-digital converter (ADC) for high-resolution digitization, programmable-gain amplifiers for boosting weak signals, and built-in filtering stages for reducing noise. The complete acquisition process implemented by the ADS1299 can be understood as a sequence of seven stages.

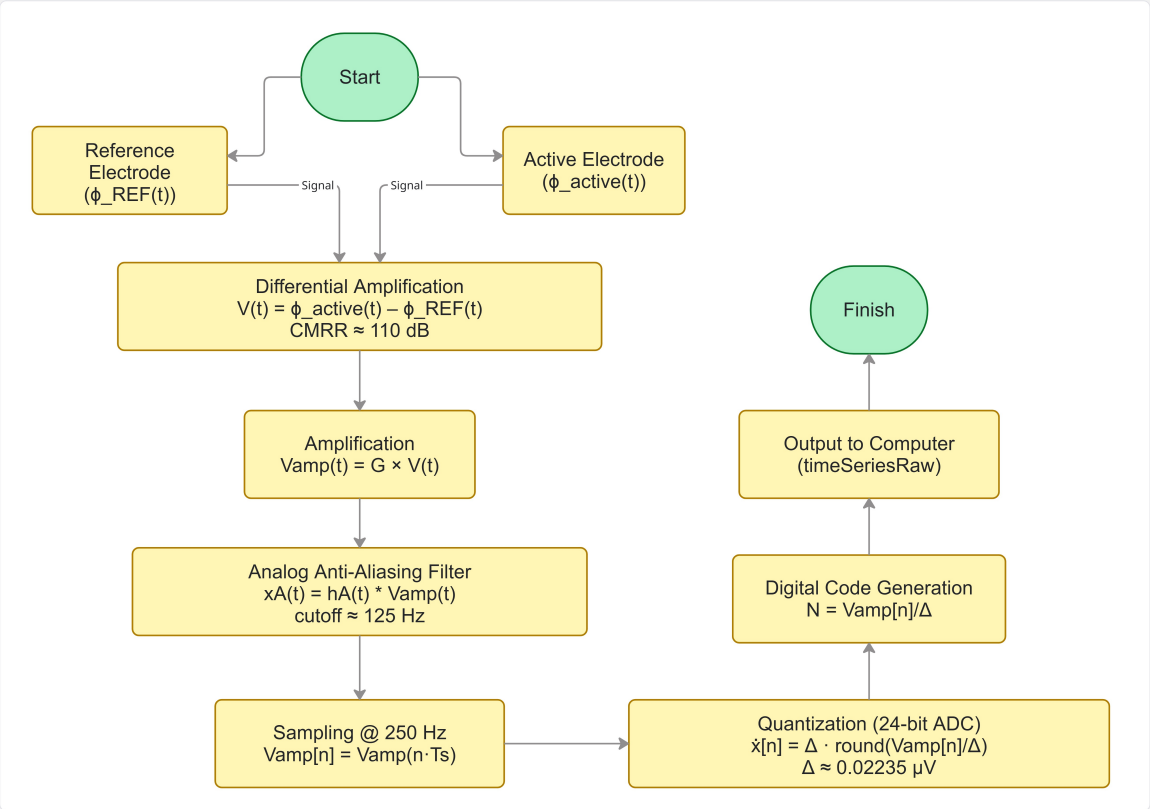

Figure 2: openBCI Hardware Acquisition

The signal acquisition chain begins with *differential amplification*, in which the system measures the potential difference between the active and reference electrodes rather than the absolute potential at either site. The differential input voltage is defined as

$$V(t) = \phi_{\text{active}}(t) - \phi_{\text{REF}}(t),$$

a formulation that effectively suppresses noise components that appear identically on both

electrodes. This noise rejection is governed by the ADS1299 front end, whose common-mode rejection ratio (CMRR) exceeds 110 dB, corresponding to a linear attenuation factor of approximately 3×10^5.

Following differential amplification, the signal is scaled by the programmable gain amplifier,

$$V_{\text{amp}}(t) = G\,V(t),$$

which increases the microvolt-level EEG fluctuations to a range suitable for digitization.

Before conversion to the digital domain, an *analog anti-aliasing filter* attenuates spectral components above the Nyquist limit. The filtered signal is expressed as

$$x_A(t) = h_A(t) * V_{\text{amp}}(t),$$

where $h_A(t)$ denotes the filter's impulse response.

The conditioned analog signal is then uniformly sampled with period $T_s = 0.004\,\text{s}$, corresponding to $f_s = 250\,\text{Hz}$. The resulting discrete-time sequence is

$$V_{\text{amp}}[n] = V_{\text{amp}}(nT_s).$$

Each sample is quantized by the 24-bit ADC using two's complement encoding. The quantized output is

$$\hat{x}[n] = \Delta\,\text{round}\left(\frac{V_{\text{amp}}[n]}{\Delta}\right),$$

where Δ denotes the quantization step size. For the ADS1299, this step size is

$$\Delta_{\text{LSB}} = \frac{2V_{\text{ref}}}{G\,2^{24}} \approx 0.02235\,\mu\text{V},$$

yielding a maximum quantization error of $\pm\Delta/2 \approx 0.011\,\mu\text{V}$.

Finally, the ADC outputs the corresponding signed integer code,

$$N = \frac{V_{\text{amp}}[n]}{\Delta_{\text{LSB}}},$$

which is transmitted to the host computer as the raw time-series data. In summary, the hardware pipeline—differential amplification, gain scaling, analog filtering, sampling, and quantization—transforms extremely weak biopotential signals into precise digital measurements suitable for downstream software processing.

2.2 OpenBCI GUI Digital Signal Processing

Having established how the hardware conditions, amplifies, and digitizes the EEG signal, we now transition to the software domain. In particular, the OpenBCI GUI (Graphical User Interface) represents the first stage of software-based signal interpretation. After the Cyton board transmits the integer-encoded data packets, the GUI performs two essential operations. First, it converts these integer codes back into meaningful voltage values expressed in microvolts, reestablishing the physical interpretation of the recorded EEG. Second, it applies digital filters that remove noise and enhance the clarity of the reconstructed signal. Together, these operations ensure that the incoming data stream is both interpretable and ready for visualization or higher-level analysis. Figure 3 illustrates the full signal acquisition flow from hardware to software.

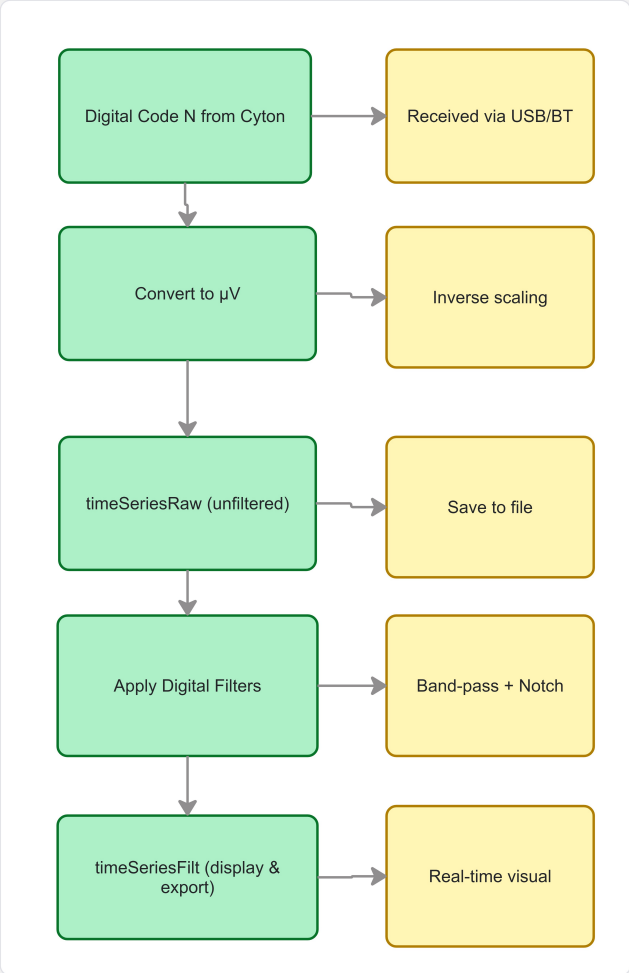

Figure 3: Digital signal reconstruction and filtering pipeline in the OpenBCI GUI.

As the data transitions from raw numerical form toward physiologically meaningful units, the first computational step involves restoring the original microvolt-scale signal amplitude.

2.2.1 Inverse Scaling to Microvolts

At this stage, the OpenBCI GUI reverses the hardware-side scaling by converting the incoming digital code back into a voltage expressed in microvolts. Since the ADS1299 first amplified the signal by the gain factor G and then quantized it into an integer code N, the GUI reconstructs the voltage using

$$\hat{x}[n] = \frac{N \cdot \Delta_{\text{LSB}}}{G},$$

where N is the transmitted digital code, Δ_{LSB} is the quantization step size (approximately $0.02235\,\mu\text{V}$), and G is the gain applied during amplification (typically 24). We divide by G because the hardware originally multiplied the signal by this factor; undoing this operation restores the voltage to its true physiological scale. This reconstructed signal, known as `timeSeriesRaw` in the GUI, corresponds to the unfiltered EEG trace displayed in microvolts.

Once the signal has been accurately reconstructed in the voltage domain, the next logical step is to remove unwanted frequency components that could obscure meaningful neural activity.

2.2.2 Digital Filtering

In this phase, the OpenBCI GUI applies *digital filters* to suppress noise and isolate the relevant components of the EEG. The filtered value at sample index n is given by

$$x_{\text{filt}}[n] = \sum_{k=-\infty}^{\infty} h_D[k] \cdot \hat{x}[n-k],$$

where $h_D[k]$ represents the digital filter coefficients and $\hat{x}[n-k]$ denotes earlier samples of the reconstructed signal. Conceptually, the filter outputs each sample as a weighted combination of multiple past inputs, allowing desired frequencies to pass while attenuating sources of interference.

With the filtering mechanism defined, we now turn to the specific types of filters most commonly employed in the OpenBCI GUI.

2.2.3 Common Filters Used in OpenBCI GUI

Digital filtering is central to maintaining clarity in EEG recordings, and the OpenBCI GUI implements several standard filter types to achieve this goal. Two of the most critical are the *band-pass filter*, which isolates the EEG-relevant frequency band, and the *notch filter*, which suppresses power line interference.

Band-Pass Filter

The *band-pass filter* preserves frequencies relevant for EEG interpretation, typically between *1 and 50 Hz*. This removes slow drifts (near 0 Hz) caused by electrode motion or thermal variation, and suppresses high-frequency components above 50 Hz generated by muscle activity or environmental electronics. The 1 Hz high-pass portion also acts as a *DC blocking filter*, removing constant voltage offsets accumulated during acquisition. In the OpenBCI GUI, the default configuration is a 1–50 Hz band-pass implemented using an *Infinite Impulse Response (IIR)* architecture, providing steep roll-off characteristics with minimal computational cost.

Notch Filter

A *notch filter* selectively attenuates power line interference at either *60 Hz* (North America) or *50 Hz* (Europe/Asia). This noise source is narrowband and strong, making it a frequent contaminant even under ideal electrode conditions. The notch filter removes this interference while preserving adjacent neural frequencies. The GUI implements it using an IIR structure with a quality factor $Q \approx 30$, producing a rejection bandwidth of about 2 Hz around the line frequency.

After filtering removes major noise sources, the GUI can extract simple statistical features that help evaluate channel quality and signal behavior in real time.

2.2.4 Basic Signal Statistics

Real-time EEG assessment often begins with *basic statistical measures* that reveal overall signal stability and channel behavior. Metrics such as the mean and variance provide quick insight into baseline drift, electrode contact quality, and neural activity. Evaluating these parameters ensures that the data is suitable for more advanced processing.

Channel Mean (DC Offset)

A fundamental indicator of signal quality is the *mean voltage* of each channel, computed as

$$\bar{x}_i = \frac{1}{N} \sum_{n=0}^{N-1} x_i[n],$$

where $x_i[n]$ is the n-th sample of channel i and N is the evaluation window size. This mean value, or *DC offset*, quantifies how far the baseline deviates from zero. The ADS1299 does not remove DC internally, so any electrode-based offset is digitized alongside the neural signal. When the 1 Hz high-pass portion of the GUI's band-pass filter is applied, this DC component is effectively suppressed in the `timeSeriesFilt` output. Large offsets in `timeSeriesRaw` may indicate poor electrode contact, drying gel, or hardware instability. Beyond the mean, understanding how much the signal fluctuates around this baseline provides further insight into neural activity and noise levels.

Channel Variance

Channel variance measures the degree of fluctuation in the EEG signal around its mean and is defined in real-time contexts using the *population variance*:

$$\sigma_i^2 = \frac{1}{N} \sum_{n=0}^{N-1} \left(x_i[n] - \bar{x}_i\right)^2.$$

Although the *sample variance* (with denominator $N-1$) is unbiased for statistical inference, the population variance is preferred in streaming applications due to its stability and computational consistency. Squaring the deviations ensures that both positive and negative fluctuations contribute equally to the overall measurement. A high variance may indicate strong neural activity, increased noise, or transient artifacts, making it a valuable parameter for assessing channel reliability.

2.2.5 timeSeriesRaw vs. timeSeriesFilt

The OpenBCI GUI provides access to two representations of the EEG data: `timeSeriesRaw` and `timeSeriesFilt`. The `timeSeriesRaw` signal, denoted by $\hat{x}[n]$, is the reconstructed

voltage obtained directly from the ADC without any filtering. It contains all frequency components—including noise—and is useful for diagnosing hardware issues or inspecting exactly what the ADC captured. In contrast, `timeSeriesFilt`, represented by $x_{\text{filt}}[n]$, is the version of the signal that has passed through digital band-pass and notch filters.

By default, `timeSeriesFilt` includes the following digital processing stages:

1. 1–50 Hz band-pass filter: Removes DC drift and slow baseline fluctuations below 1 Hz, as well as high-frequency noise and muscle artifacts above 50 Hz.
2. 60 Hz notch filter (North America) or 50 Hz notch filter (Europe/Asia): Attenuates power line interference.

Both filters are implemented as *Infinite Impulse Response (IIR)* structures within the GUI, providing computationally efficient real-time processing with minimal phase distortion. This filtered signal is cleaner, easier to interpret, and generally preferred for analysis and visualization.

Having reconstructed, filtered, and statistically characterized the EEG signal within the OpenBCI GUI, the processing pipeline now shifts from local computation to real-time data distribution. At this stage, the central objective is no longer signal transformation but the efficient transmission of these processed samples to external software environments. Consequently, the next step focuses on how the system streams continuous EEG data across the network in a reliable and low-latency manner.

2.3 UDP Network Streaming

Up to this point, the Cyton hardware and the OpenBCI GUI have collaboratively transformed raw EEG potentials into a cleaned, scaled, and digitally reconstructed signal suitable for software-level analysis. Having completed the essential steps of amplification, digitization, filtering, and statistical evaluation, the processing pipeline now shifts from local computation to the distribution of data across external systems. In this final stage of the acquisition chain, the objective is to stream the processed EEG samples to real-time clients with minimal delay. Efficient EEG streaming requires a network protocol capable of supporting continuous, low-latency transmission, particularly in scenarios where multiple software environments depend simultaneously on the same data stream. Figure 4 illustrates the complete transmission pathway linking the OpenBCI hardware and software to the target application in which further processing is performed.

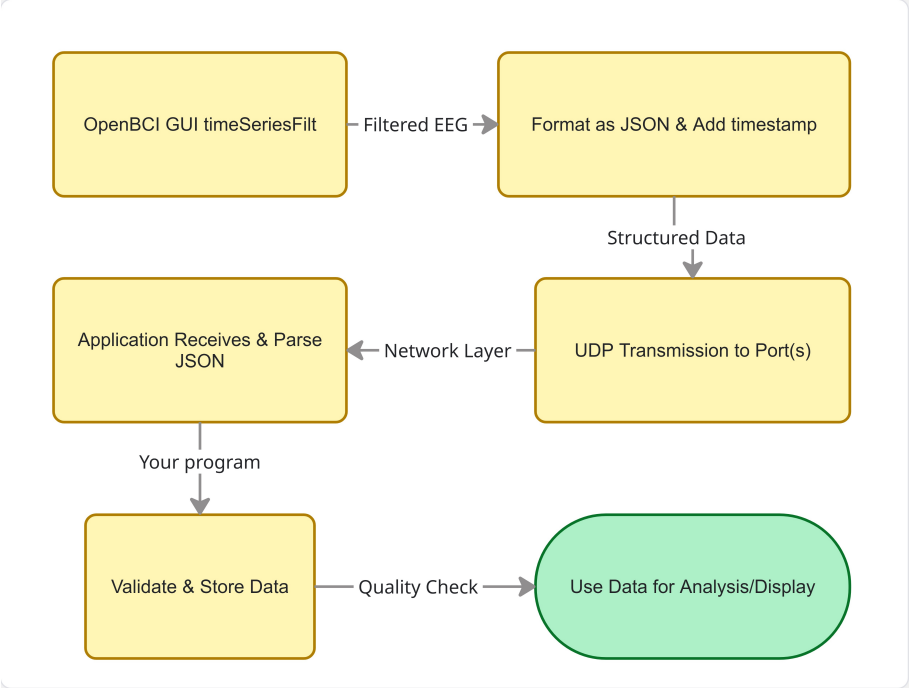

Figure 4: UDP Flow Chart

As the signal moves beyond local reconstruction and filtering, the OpenBCI system must ensure that the chosen communication protocol, output stream organization, and packet structure together support robust and timely data delivery. The following sections explain why the OpenBCI GUI employs UDP rather than TCP, how it enables simultaneous

multi-port streaming, and how multi-channel EEG samples are encapsulated within each transmitted JSON packet.

2.3.1 UDP-Based Multi-Port and Multi-Channel EEG Streaming

Real-time EEG streaming from the OpenBCI GUI requires a protocol capable of delivering continuous data with minimal delay. Although the Transmission Control Protocol (TCP) guarantees ordered delivery and error correction, it introduces notable latency due to connection overhead, congestion control, and retransmission delays. TCP also relies on a *handshake procedure*—an initialization step required to establish a reliable connection—which further increases delay.

In contrast, the User Datagram Protocol (UDP) transmits packets without establishing a connection, avoids handshake operations entirely—which are the setup exchanges required by TCP (i.e., the small initial messages sent before data transfer begins)—and achieves typical latencies of 1–5 ms, compared with TCP's 10–100 ms range. Because EEG data tolerate occasional packet loss—each packet containing only 40–50 ms of recorded activity (approximately 10–12 samples at 250 Hz)—brief interruptions affect only narrow time windows and are acceptable in real-time applications where responsiveness is prioritized over perfect reliability. For these reasons, UDP is the preferred protocol for smooth, low-latency EEG streaming.

The OpenBCI GUI also provides a multi-port streaming architecture, enabling up to three independent UDP streams to operate simultaneously. Each stream can target a different IP address and port, making it possible to send raw EEG data to Python, spectral features to Max/MSP, and band-power values to a browser-based interface, all in parallel and without significant delay.

Each UDP packet is formatted in *JSON* and contains a timestamp accompanied by an array of eight EEG channel samples. A typical packet structure appears as follows:

```
{
  "type": "eeg",
```

```
  "data": [
    [ch1_sample1, ch2_sample1, ..., ch8_sample1],
    [ch1_sample2, ch2_sample2, ..., ch8_sample2],
    ...
    [ch1_sampleN, ch2_sampleN, ..., ch8_sampleN]
  ],
  "timestamp": 1234567890
}
```

Conceptually, the incoming data form an $8 \times N$ matrix:

$$\mathbf{X} = \begin{bmatrix} x_1[0] & x_1[1] & \cdots & x_1[N-1] \\ x_2[0] & x_2[1] & \cdots & x_2[N-1] \\ \vdots & \vdots & \ddots & \vdots \\ x_8[0] & x_8[1] & \cdots & x_8[N-1] \end{bmatrix},$$

where each $x_i[n]$ denotes the microvolt-level EEG voltage at channel i and sample index n.

2.3.2 Network Packet Handling in Real-Time EEG Streaming

After the data have been transmitted, the receiving application must validate, decode, and integrate each UDP packet into its processing pipeline with minimal overhead. The receiver first checks whether the incoming payload contains valid JSON:

$$P(\text{packet}) = \begin{cases} \text{parse(packet)} & \text{if valid JSON,} \\ \text{reject} & \text{otherwise,} \end{cases}$$

ensuring that malformed packets do not propagate through downstream computations. Following validation, the application extracts the numerical arrays, converts string values to floating-point numbers when necessary, and stores them in buffers for real-time analysis.

Because UDP does not guarantee delivery, detecting packet loss is essential. This is

typically performed by comparing incoming sequence numbers to those expected. The loss rate is computed as

$$\text{Loss Rate} = \frac{N_{\text{expected}} - N_{\text{received}}}{N_{\text{expected}}} \times 100\%,$$

where loss rates below 5% often have negligible impact on real-time EEG interpretation.

Network latency also plays a critical role. Defined by

$$t_{\text{latency}} = t_{\text{receive}} - t_{\text{send}},$$

latency must typically remain below 1–5 ms for responsive neurofeedback or BCI applications. However, the *total system latency* includes three components:

1. Hardware acquisition delay: ≈ 4 ms at 250 Hz sampling.
2. GUI processing delay: typically 1–2 ms.
3. Network transmission delay: 1–5 ms under normal conditions.

This yields an end-to-end latency of *6–11 ms*, well below the 20 ms threshold at which delays become perceptible to users.

Finally, the *packet rate* determines how frequently data are transmitted. For eight channels sampled at 250 Hz with 4-byte samples, the data rate is 8 kB/s. Dividing by the packet size P_{size} yields the packet rate:

$$R_{\text{packet}} = \frac{8000}{P_{\text{size}}}.$$

Typical OpenBCI configurations send packets every 40–50 ms (20–25 packets per second), ensuring smooth and stable streaming.

Having outlined the overall processing pipeline—from analog signal acquisition in the hardware to digital reconstruction, filtering, and UDP-based transmission at the software level—we can now summarize the system-specific aspects described in this section. The OpenBCI Cyton platform integrates precision hardware, digital preprocessing, and real-time network streaming into a unified EEG acquisition pipeline. At the hardware level,

the ADS1299 provides eight simultaneous channels with 24-bit resolution, a sampling rate of $f_s = 250\,\text{Hz}$, a least-significant-bit voltage resolution of $\Delta_{\text{LSB}} = 0.02235\,\mu\text{V}$, and exceptionally low input-referred noise, enabling reliable detection of microvolt-level neural activity. Building on this foundation, the OpenBCI GUI applies essential preprocessing operations—including a 1–50 Hz band-pass filter and a 50/60 Hz notch filter—to produce clean and interpretable EEG traces suitable for further analysis.

Once filtered and reconstructed, the data are packaged into JSON structures and streamed via UDP, a low-latency communication protocol well suited for real-time neurotechnology applications. With support for simultaneous multi-port transmission and packet rates of 20–25 Hz, the system delivers high-resolution EEG signals efficiently to external clients for real-time analysis, interaction, or multimodal integration.

Table 1: OpenBCI Cyton 8-Channel EEG System Specifications

Parameter	Symbol	Value
Hardware (ADS1299)		
Number of Channels	C	8
ADC Resolution	N	24 bits
Sampling Rate	f_s	250 Hz
Sampling Period	T_s	4 ms (0.004 s)
Programmable Gain	G	1, 2, 4, 6, 8, 12, 24
Default Gain	G_{default}	24
Internal Reference	V_{ref}	4.5 V
Input Range (after gain)	—	$\pm V_{\text{ref}}$
LSB Step Size (G=24)	Δ_{LSB}	0.02235 µV
Common-Mode Rejection Ratio	CMRR	>110 dB (~300,000:1)
Anti-Aliasing Filter Cutoff	f_{cutoff}	~125 Hz
Signal Characteristics		
Typical EEG Amplitude	—	20-100 µV
Input-Referred Noise	—	<1 µV (RMS)
Quantization Error Range	e_q	±0.0112 µV
Digital Processing		
Band-Pass Filter (Default)	—	1-50 Hz (IIR)
Notch Filter (USA)	—	60 Hz (IIR, Q~30)
Notch Filter (Europe/Asia)	—	50 Hz (IIR, Q~30)
Network Streaming		
Protocol	—	UDP
Max Simultaneous Streams	—	3
Data Format	—	JSON
Typical Packet Rate	—	20-25 packets/s
Samples per Packet	—	10-12 samples (40-50 ms)
Data Rate	—	~8 kB/s
Network Latency	—	1-5 ms (typical)
Total System Latency	—	6-11 ms (hardware + GUI + network)

3 Browser-Side EEG Spectral Analysis and Visualization

The preceding sections have established how cortical potentials are transformed into digital data streams through hardware acquisition, GUI filtering, and UDP transmission. At this section, following UDP-to-WebSocket conversion by the Node.js server, the signal processing chain transitions into the browser environment, where the computationally intensive task of spectral analysis takes place. Here, raw microvolt-level voltage sequences are transformed into meaningful frequency-domain representations that reveal the underlying rhythms of neural activity.

This browser-based pipeline, illustrated in Figure 5 and implemented in javaScript files, consists of five interconnected stages. The process begins with WebSocket-based data reception, where incoming JSON packets are parsed and validated before individual EEG samples are extracted. These samples are then stored in a circular buffer that maintains temporal continuity while managing memory efficiently. Critically, the system implements tempo-adaptive windowing, wherein analysis segment durations are dynamically determined by musical tempo and time signature—enabling precise temporal alignment between neural responses and musical events. Once a complete window is available, the discrete Fourier transform decomposes the time-domain signal into frequency components, from which neurophysiologically relevant bands (delta, theta, alpha, beta, gamma) are extracted. Finally, band-specific power calculations drive real-time canvas-based visualization.

Importantly, each stage depends on the successful completion of the preceding one, forming a tightly coupled processing chain that must balance computational efficiency with robustness to network variability. The following subsections examine each component in detail, using the section headlines that correspond to the nodes in Fig. 4 for traceability. This structure illustrates how theoretical signal-processing principles translate into practical, browser-based implementations.

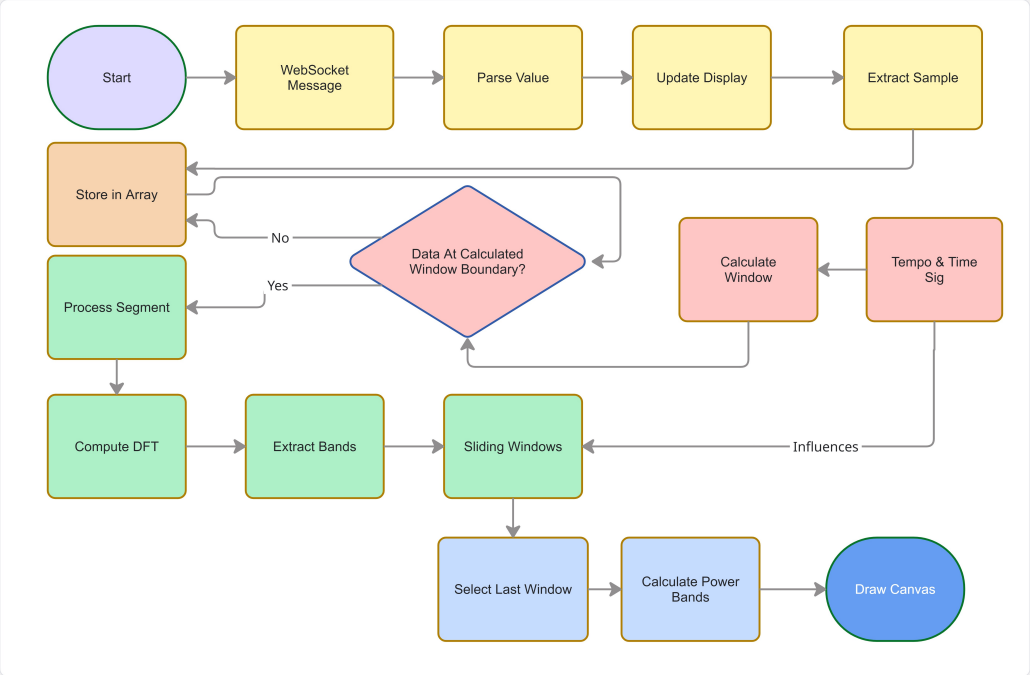

Figure 5: Browser-side EEG processing pipeline

3.1 Data Reception and Parsing

The yellow nodes in the Fig. 4 illustrates reception and parsing of the data in which the browser-side processing pipeline begins with establishing a reliable communication channel between the server and the web application to transmits this data for further processing. This initial stage demands careful attention to asynchronous event handling - where events are processed one at a time, in order, and each event must fully complete before the next one begins, data validation, and error recovery to ensure subsequent stages receive well-formed, temporally consistent data. As illustrated in the flowchart, this comprises four sequential operations: WebSocket message reception, value parsing, display updating, and sample extraction.

3.1.1 WebSocket Message

Modern real-time web applications require bidirectional, low-latency communication, which the WebSocket protocol provides through persistent connections. Unlike traditional HTTP request-response cycles, WebSocket allows the server to push data immediately upon availability. The browser initiates a connection to `ws://localhost:8081`, creating a direct

channel for continuous EEG packet transmission.

Because EEG data arrive at 250 Hz (every 4 milliseconds), the system cannot block on any operation. Each incoming message triggers an `onmessage` callback that processes the packet and immediately returns control to the event loop, ensuring subsequent packets arrive without delay. The implementation includes `onerror` and `onclose` handlers that detect connection failures and attempt automatic reconnection, maintaining continuous data flow despite network disruptions.

3.1.2 Parse Value

Once a WebSocket message fires, the raw string undergoes conversion to a JavaScript object via `JSON.parse()`. The expected packet format consists of three fields: a `type` identifier (`"eeg"`), a `data` array containing multichannel samples as nested arrays `[[ch1,...,ch8], ...]`, and a `timestamp`.

Network transmission is inherently unreliable. Consequently, parsing is wrapped in a `try-catch` block to intercept JSON syntax errors. If parsing fails, the packet is discarded with a logged error, but execution continues. Successfully parsed packets undergo structural validation: the system verifies expected fields exist, that `type` matches `"eeg"`, and that the `data` array contains eight-element subarrays. Packets failing these checks are discarded, ensuring only well-formed data propagate through the pipeline.

3.1.3 Update Display

Beyond accepting and validating data, the reception stage provides immediate visual feedback. As each packet is successfully parsed, the application extracts current microvolt values for all eight channels and updates corresponding HTML elements. This real-time display confirms correct data flow, allows operators to monitor signal quality, and provides color-coded indicators that flag abnormal conditions.

Channels exhibiting voltages near ADC saturation limits (± 187 µV) are highlighted in red, signaling potential clipping artifacts. Conversely, channels showing stable signals appear

in green, confirming optimal recording conditions. This immediate feedback ensures data quality before committing computational resources to analysis.

3.1.4 Extract Sample

Having validated the packet and updated the display, the system extracts individual EEG samples for buffering. The `data` field is a two-dimensional array: the first dimension indexes time (typically 10-12 samples per packet) and the second indexes channels (eight per sample). Access requires double-indexing: `data[sampleIndex][channelIndex]`. Before buffering, the system performs data type verification using `parseFloat()` to ensure numeric format. The validated sample then passes to the circular buffer's insertion method, completing the first stage of the browser-side processing pipeline.

3.2 Circular Buffer Management

Circular buffer is represented with the single brown node in Fig. 3 as the second stage in signal processing. Following successful extraction of validated EEG samples from incoming WebSocket packets, the processing pipeline transitions to data storage. At this point, the system employs a circular buffer architecture that maintains a fixed-size window of recent samples while automatically discarding older data. This approach balances two competing requirements: providing sufficient temporal context for spectral analysis while ensuring that memory consumption remains constant regardless of recording duration.

3.2.1 Store in Array

The circular buffer is implemented as a fixed-length JavaScript array initialized at system startup with predetermined capacity. This capacity is determined by the maximum window size required for spectral analysis, which depends on the slowest musical tempo the system expects to encounter. For instance, at 60 beats per minute with 4/4 time signature, one measure spans 4 seconds, corresponding to 1000 samples at 250 Hz sampling rate. To accommodate slower tempos and provide margin for sliding window operations, typical implementations allocate buffers holding 2000-4000 samples, representing 8-16 seconds of

continuous recording.

The buffer operates on a write-pointer mechanism that tracks the current insertion position. When a new sample arrives from the extraction stage, it is written to the array index indicated by the write pointer, which then increments by one. Critically, when the write pointer reaches the end of the array, it wraps around to index zero rather than expanding the array. This wraparound behavior—achieved through modulo arithmetic where `writePointer = (writePointer + 1) % bufferSize`—gives the circular buffer its name and memory-efficient properties. As new samples continuously overwrite old samples, the buffer always contains the most recent N samples, where N equals buffer capacity.

Reading data from the circular buffer requires careful index management to maintain temporal ordering. When the spectral analysis stage requests a window of M samples, the system must extract M consecutive samples accounting for the possibility that these samples span the wraparound boundary. This is accomplished by calculating a read-start index as `readStart = (writePointer - windowSize + bufferSize) % bufferSize`, then extracting samples sequentially with wraparound handling. For example, if the write pointer sits at index 150 in a 2000-sample buffer and 200 samples are requested, the system retrieves samples from indices 1950-1999 followed by 0-149.

The implementation maintains temporal integrity through synchronized operations. Because JavaScript executes in a single-threaded event loop, race conditions between writing new samples and reading existing samples do not occur. However, the system must ensure that read operations never request more samples than the buffer contains. This is enforced by tracking the total number of samples received since initialization: until this count exceeds the requested window size, spectral analysis is deferred, preventing attempts to analyze incomplete data.

Memory efficiency represents the primary advantage of this architecture. By maintaining fixed allocation, the buffer avoids garbage collection overhead associated with dynamic array resizing and eliminates memory leaks that could degrade performance during extended

recording sessions. The constant memory footprint enables reliable real-time operation, as processing latency remains predictable regardless of session duration. Following successful storage, samples remain available for retrieval when the tempo-adaptive windowing stage determines that sufficient data have accumulated for spectral analysis.

3.3 Tempo-Adaptive Windowing

One of the distinctive features of this system is the synchronization of EEG analysis windows with musical tempo, enabling multimodal correlation between neural dynamics and musical structure. This part of the program is illustrated with pink nodes in Fig. 3. Traditional EEG spectral analysis employs fixed-duration windows that bear no relationship to external stimuli. In contrast, the present implementation dynamically adjusts window duration to align precisely with musical measures, ensuring that each spectral analysis captures neural activity corresponding to complete musical phrases. This tempo-adaptive approach is essential for investigating how brain rhythms entrain to musical rhythms, a central question in music cognition research.

3.3.1 Tempo and Time Signature Retrieval

Before calculating appropriate window sizes, the system must obtain current musical tempo and time signature information. This information originates from MIDI data processed by the the javaScript module, which tracks musical timing in real-time as MIDI events arrive from Ableton Live. The counter.js module maintains two critical variables: `currentTempo` (expressed in beats per minute) and `timeSignature` (expressed as a string such as "4/4" or "3/4"). These values are exposed as global JavaScript variables accessible to the EEG processing pipeline.

The tempo retrieval mechanism operates continuously, querying the script at regular intervals—typically every 100 milliseconds—to detect tempo changes that may occur during performance. When MIDI data indicate a tempo change, the counter.js module updates its internal state, and the next query by the EEG pipeline retrieves the new value. This responsiveness ensures that window calculations remain synchronized even during prepared

accelerandi, ritardandi, or abrupt tempo shifts common in expressive musical performance.

3.3.2 Window Size Calculation

Having obtained tempo and time signature, the system calculates the appropriate window duration corresponding to user selected musical measure number. The fundamental relationship derives from the definition of tempo: in case of selected 1 bar duration, beats per minute indicates how many beats occur in 60 seconds, so the duration of one beat equals 60/tempo seconds. The time signature's numerator specifies how many beats comprise one measure. Therefore, one measure spans (60/tempo) × beatsPerMeasure seconds.

To convert this temporal duration to sample count, the formula multiplies by the sampling rate: `windowSize = (60 / tempo) × beatsPerMeasure × samplingRate`. For example, at 120 beats per minute with 4/4 time signature and 250 Hz sampling rate, the calculation proceeds as follows: one beat lasts 60/120 = 0.5 seconds, one measure contains 4 beats spanning 2 seconds, and 2 seconds at 250 Hz yields 500 samples. Similarly, at 60 BPM with 3/4 time, one measure spans 3 seconds corresponding to 750 samples.

Because array indices must be integers, the calculated window size undergoes rounding to the nearest whole number. Additionally, the system enforces minimum and maximum constraints to prevent pathological cases. The minimum constraint—typically 250 samples (1 second)—ensures sufficient frequency resolution for distinguishing EEG bands, as frequency resolution equals samplingRate/windowSize. Windows shorter than 250 samples would yield resolution coarser than 1 Hz, insufficient for separating theta (4-8 Hz) from alpha (8-13 Hz) bands. The maximum constraint—typically equal to buffer capacity—prevents requests for more samples than the circular buffer contains. When calculated windows exceed these bounds, they are clipped to the nearest valid value.

3.3.3 Window Boundary Detection

Once the appropriate window size is determined, the system must decide whether sufficient data have accumulated in the circular buffer to perform analysis. This decision occurs through a boundary detection check that compares the total number of samples

received against the calculated window size. The check employs a simple conditional: `if (totalSamplesReceived >= windowSize)`, then proceed to spectral analysis; otherwise, continue buffering.

This decision point appears in the flowchart as a diamond-shaped conditional box labeled "Data At Calculated Window Boundary?" with two branches. The "No" branch loops back to the storage stage, allowing additional samples to accumulate until the condition is satisfied. The "Yes" branch proceeds forward to segment processing and DFT computation. This gating mechanism ensures that spectral analysis never attempts to process incomplete windows, which would produce invalid frequency-domain representations.

The boundary detection operates continuously as new samples arrive. After each successful DFT computation, the system does not reset the sample counter but instead continues accumulating samples, checking the boundary condition with each new arrival. This enables sliding window analysis wherein successive windows overlap substantially, providing high temporal resolution of spectral dynamics while maintaining alignment with musical structure.

3.4 Spectral Analysis via Discrete Fourier Transform

When the buffer reaches the calculated window boundary, the system performs frequency-domain analysis to extract spectral features corresponding to established EEG frequency bands. This transformation from time-domain voltage sequences to frequency-domain power spectra constitutes the computational core of the entire pipeline, revealing the rhythmic oscillations that characterize neural activity. While raw EEG data appear as chaotic voltage fluctuations when plotted over time, spectral decomposition exposes the underlying periodic components—delta, theta, alpha, beta, and gamma rhythms—that reflect distinct neurophysiological states and cognitive processes.

3.4.1 Process Segment

Before applying the Fourier transform, the system extracts the appropriate segment from the circular buffer and applies preprocessing operations that improve spectral quality. Segment extraction employs the read-index calculation described previously, retrieving windowSize consecutive samples while accounting for buffer wraparound. These samples are copied into a separate processing array to prevent interference from ongoing write operations.

The extracted segment typically undergoes windowing to reduce spectral leakage—an artifact that occurs when the signal at window boundaries exhibits discontinuities. Abruptly truncating a sinusoidal signal creates artificial high-frequency components that contaminate the spectrum. To mitigate this, the system applies a window function that gradually attenuates samples toward the segment boundaries. The Hann window is commonly employed, defined as

$$w[n] = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right), \qquad n = 0, 1, \ldots, N-1.$$

Multiplying each sample by its corresponding window coefficient produces smooth transitions to zero at both boundaries, eliminating discontinuities while preserving central samples at full amplitude.

Additionally, the segment undergoes detrending to remove DC bias. EEG signals often contain slow voltage drifts unrelated to neural oscillations—arising from electrode polarization, amplifier offsets, or movement artifacts. These drifts manifest as large DC components (0 Hz) in the spectrum that can distort power calculations. Detrending typically involves computing the segment's mean value and subtracting it from all samples, centering the signal around zero. This simple linear detrending effectively removes the DC component without affecting higher-frequency neural rhythms of interest.

3.4.2 Compute DFT

The Discrete Fourier Transform decomposes the preprocessed time-domain segment into its constituent frequency components. Mathematically, the DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi kn/N} \tag{1}$$

where x[n] represents the input samples, N is the window length, k indexes frequency bins from 0 to N-1, and j is the imaginary unit. Each frequency bin k corresponds to a specific frequency f = k·fs/N, where fs is the sampling rate (250 Hz). The exponential term can be expanded using Euler's formula as cos(2kn/N) - j·sin(2kn/N), yielding real and imaginary components.

In JavaScript, this is implemented through nested loops. The outer loop iterates over frequency bins k, while the inner loop accumulates the sum over time samples n. For each bin, the algorithm maintains separate accumulators for real and imaginary parts:

Real[k] = x[n]·cos(2kn/N) Imaginary[k] = x[n]·sin(2kn/N)

This naive implementation exhibits $O(N^2)$ computational complexity: for each of N frequency bins, N multiply-accumulate operations are performed, totaling N^2 operations. For typical window sizes of 500-1000 samples, this corresponds to 250,000-1,000,000 operations per transform. Modern JavaScript engines execute these operations in 5-10 milliseconds on contemporary hardware, acceptable for real-time performance. However, production implementations often employ Fast Fourier Transform (FFT) algorithms—such as the Cooley-Tukey radix-2 FFT—that reduce complexity to O(N log N), providing substantial speedup for larger windows.

Once the DFT is computed, the magnitude spectrum is calculated from the real and imaginary components: $|X[k]| = \text{sqrt}(\text{Real}[k]^2 + \text{Imaginary}[k]^2)$. The magnitude spectrum represents the amplitude of each frequency component, with units of microvolts. The power spectrum—more commonly used in EEG analysis—is obtained by squaring

the magnitude: Power[k] = |X[k]|². Power has units of microvolts squared and directly represents the energy contribution of each frequency.

The frequency resolution of the DFT equals f = fs/N. For a 500-sample window at 250 Hz, resolution is 0.5 Hz, meaning each bin spans 0.5 Hz. This resolution determines the ability to distinguish nearby frequencies: narrow EEG bands like alpha (8-13 Hz) require sufficient resolution to separate them from adjacent theta (4-8 Hz) and beta (13-30 Hz) bands. Longer windows provide finer frequency resolution but reduce temporal resolution, embodying the fundamental uncertainty principle of time-frequency analysis.

3.4.3 Extract Bands

Having computed the power spectrum, the system extracts neurophysiologically relevant frequency bands. EEG research has established five canonical bands, each associated with distinct cognitive and behavioral states:

Delta (0.5-4 Hz) dominates during deep sleep and reflects unconscious processes. Elevated delta in waking states may indicate cortical lesions or pathological conditions. Theta (4-8 Hz) appears during meditation, creative thinking, and REM sleep, reflecting memory consolidation and emotional processing. Alpha (8-13 Hz) characterizes relaxed wakefulness with closed eyes, often called the "idling rhythm" of the visual cortex. Alpha suppression occurs during visual processing and mental effort. Beta (13-30 Hz) accompanies active thinking, focused attention, and anxiety, reflecting cortical activation. Gamma (30-50 Hz) supports high-level cognition, sensory binding, and consciousness itself, though its interpretation remains debated.

To extract band power, the system maps frequency ranges to DFT bin indices. The bin corresponding to frequency f is k = round(f·N/fs). For delta (0.5-4 Hz) in a 500-sample window at 250 Hz, the bin range is k = 1 to 8. The system iterates over this range, accumulating power values: DeltaPower = Power[k] for k = 1 to 8. This sum represents the total spectral energy within the delta band. The same procedure applies to all bands, yielding five scalar values representing the relative strength of each rhythm.

Band power can be expressed in absolute terms (microvolts squared) or normalized. Relative band power divides each band by the total power across all bands, yielding proportions that sum to 1.0. This normalization reduces inter-subject variability and facilitates comparisons. Logarithmic scaling (10·log(power)) converts power to decibels, compressing the dynamic range and emphasizing relative changes.

3.4.4 Sliding Windows

Rather than analyzing a single window and stopping, the system employs sliding window analysis to track temporal evolution of spectral features. After computing band powers for one window, the system advances by a hop size—typically 50

This sliding window approach generates a time-frequency representation wherein band powers are computed at regular intervals (every 1 second for 250-sample hops at 250 Hz). The resulting power trajectories reveal how neural rhythms wax and wane in response to musical events, task demands, or spontaneous fluctuations. For instance, alpha power may decline during eyes-open periods and recover during rest, while beta may surge during cognitive effort.

The trade-off between temporal and frequency resolution is fundamental. Longer windows yield finer frequency resolution but coarser temporal localization, while shorter windows provide better time resolution at the cost of frequency precision. The tempo-adaptive windowing approach navigates this trade-off by selecting window durations that align with musical structure, ensuring that each analysis window captures meaningful musical units while maintaining adequate frequency resolution for band separation.

3.5 Power Calculation and Canvas Visualization

The final stage computes band-specific power metrics and renders real-time visualizations on an HTML5 canvas element, providing immediate visual feedback that enables operators to monitor neural dynamics as they unfold. This visualization confirms that spectral analysis functions correctly, allows quality assessment of ongoing recordings, and provides

intuitive representations of complex frequency-domain data.

3.5.1 Select Last Window

Before computing displayable power values, the system must select the appropriate analysis window. In sliding window implementations that generate multiple overlapping analyses per second, the visualization displays the most recent complete window to maintain temporal coherence with ongoing neural activity. The system maintains a queue of completed spectral analyses, each tagged with a timestamp indicating when the corresponding EEG segment was recorded.

Window selection prioritizes recency while avoiding incomplete analyses. When the visualization update cycle triggers—synchronized with the browser's refresh rate via `requestAnimationFrame`—the system queries the analysis queue for the newest entry. If multiple analyses completed since the last frame, only the most recent is selected, preventing visualization lag. This ensures that displayed power values reflect current brain state rather than outdated data.

During initial startup or following tempo changes, the analysis queue may be temporarily empty while the system accumulates sufficient samples for the new window size. To prevent visualization artifacts, the system retains the previous frame's display until new analyses become available, maintaining smooth visual continuity.

3.5.2 Calculate Power Bands

Having selected the target window, the system computes summary statistics for each frequency band. Although spectral analysis already calculates power for each DFT bin, visualization requires aggregating these bins into five scalar values. The fundamental calculation sums power across bins comprising each band:

$$P_{\text{band}} = \sum_{k \in \text{band}} |X[k]|^2.$$

Raw power values often span several orders of magnitude, complicating visualization.

To address this, the system typically applies logarithmic scaling, converting power into decibels:

$$P_{\text{dB}} = 10 \log_{10}(P).$$

This compression maps the wide dynamic range of EEG power — varying from 10^{-2} to $10^{2}\,\mu V^2$ — into a manageable scale suitable for bar chart display. Decibel scaling also aligns with human perception, which tends to be logarithmic.

Alternatively, relative band power normalizes each band by total power:

$$P_{\text{relative}} = \frac{P_{\text{band}}}{\sum_{\text{all bands}} P_{\text{band}}}.$$

This yields proportions summing to 1.0, facilitating comparisons between bands and reducing sensitivity to absolute amplitude variations caused by electrode impedance changes. The choice depends on research goals: absolute power preserves overall signal strength information, while relative power emphasizes spectral energy distribution.

3.5.3 Draw Canvas

Visualization employs the HTML5 Canvas API, which provides a programmable drawing surface for real-time graphics. The canvas element exists as a DOM object with specified dimensions, and JavaScript accesses it through a 2D rendering context exposing drawing primitives.

The rendering process begins by clearing the previous frame using `context.clearRect()`, erasing all prior graphics. The coordinate system origin sits at the top-left corner, with x increasing rightward and y downward. To display five frequency bands as a bar chart, the system divides canvas width by five, allocating equal horizontal space to each band. Bar height is computed by mapping power values—after logarithmic scaling or normalization—to pixel coordinates through linear interpolation.

Color coding enhances interpretability: delta appears in blue, theta in green, alpha in yellow, beta in orange, and gamma in red. This rainbow-like progression leverages familiar visual metaphors. Each bar is drawn using `context.fillRect(x, y, width, height)`, with parameters positioning and sizing the rectangle. Text labels identifying bands and numerical values are rendered using `context.fillText()`.

The rendering cycle synchronizes with the browser's refresh rate—typically 60 Hz—through `requestAnimationFrame()`, which schedules updates to coincide with the display's vertical sync. This prevents screen tearing and ensures smooth animation. The visualization callback recursively calls `requestAnimationFrame()`, creating a continuous rendering loop that updates as new spectral analyses complete. When band powers change gradually, the visualization exhibits fluid motion; when they shift abruptly—during eyes-open to eyes-closed transitions—the bars jump accordingly, providing immediate visual feedback of neural state changes.

3.6 Summary of Browser-Side Pipeline

This section has described the complete browser-side EEG processing pipeline, which transforms UDP-transmitted JSON packets into real-time frequency-domain visualizations suitable for multimodal music-brain research. The pipeline comprises five sequential stages, each addressing a distinct aspect of signal processing while maintaining tight integration with the overall system architecture.

The data flow begins with WebSocket message reception and JSON parsing, where incoming packets undergo validation and error handling to ensure temporal consistency. Extracted samples then enter a circular buffer employing wraparound write-pointer management, maintaining fixed memory allocation while providing access to recent history. Critically, the tempo-adaptive windowing stage synchronizes analysis segments with musical structure by dynamically calculating window sizes based on MIDI-derived tempo and time signature information retrieved from counter.js. This synchronization represents the system's key innovation, enabling precise temporal alignment between neural dynamics

and musical events—a capability absent in conventional EEG analysis workflows.

Once sufficient samples accumulate, the spectral analysis stage performs discrete Fourier transformation, decomposing time-domain voltage sequences into frequency-domain power spectra. The naive $O(N^2)$ JavaScript implementation executes in 5-10 milliseconds for typical 500-1000 sample windows, meeting real-time performance requirements. Frequency band extraction then aggregates DFT bins into neurophysiologically relevant delta, theta, alpha, beta, and gamma bands, each reflecting distinct cognitive and behavioral states. Finally, the visualization stage renders band powers as color-coded bar charts using HTML5 Canvas API, synchronized with display refresh through requestAnimationFrame.

This entirely browser-based implementation demonstrates that sophisticated real-time signal processing traditionally confined to specialized environments like MATLAB can be accomplished using standard web technologies. The pipeline's modular architecture facilitates future extensions—such as incorporating FFT libraries for computational acceleration, implementing additional spectral features like spectral entropy or phase coherence, or expanding visualization capabilities to include time-frequency spectrograms. Integrated with the hardware acquisition and server components described previously, this browser-side processing completes the end-to-end pathway from cortical potentials to actionable spectral representations.