International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(12s) 2026 pp. 735-755
https://doi.org/10.70917/ijcisim-2026-3951
Article

A Real-Time GUI Platform for Neural Signal Transmission: Design, Resource Profiling, and Usability

Laith Hussein Jasim Alzubaidi[1] [2], Yaghoub Farjami[1], Mohsen Akbarpour Beni[3]

[1] Department of Computer and Information Technology, University of Qom, Qom, Iran.
Email: farjami@qom.ac.ir
ORCID: 0000-0003-1908-8826
[2] Department of Computer Techniques Engineering, College of Technical Engineering, The Islamic University, Najaf, Iraq.
Email: laith.h.alzubaidi@gmail.com
ORCID: 0009-0004-4685-0729
[3] Department of Sport Sciences, University of Qom, Qom, Iran.
Email: m.akbarpour@qom.ac.ir
ORCID: 0000-0002-3565-4851
Corresponding Author: Laith Hussein Jasim Alzubaidi

Abstract: Neural decoding systems often achieve strong offline performance, yet many fail to translate that performance into reliable real-time operation due to fragmented pipelines, limited reproducibility, and insufficient visibility into computational costs. This paper presents a real-time graphical user interface (GUI) platform for neural signal transmission that integrates preprocessing, optional feature extraction, model inference, visualization, and continuous resource monitoring within a single operational workflow. The platform supports three execution modes: ESN-only, LSTM-only, and a hybrid ESN LSTM pipeline, enabling systematic comparison of accuracy, latency, and hardware footprint under consistent conditions. Evaluation is conducted using the PhysioNet EEGMMIDB v1.0.0 (motor imagery EEG from nine subjects), with configurable sampling rates and channel selection to reflect practical deployment constraints. To ensure deterministic behavior, the platform incorporates controlled random seeding and structured logging of model and runtime configurations, improving repeatability across sessions and devices. The system also reports per-stage latency, CPU/RAM utilization, and profiling overhead to help practitioners identify bottlenecks and quantify trade-offs. Experimental observations indicate reduced blocking delays and minimal monitoring overhead, supporting stable long-session operation. Overall, the proposed platform bridges the gap between algorithmic neural decoding research and deployable, testable real-time systems by combining reproducible modeling, multi-mode execution, and transparent performance profiling in an integrated GUI environment. The primary contribution is an engineering and reproducibility framework rather than a new learning algorithm, as the decoding models used are well established independent of these papers. The novelty lies in the deterministic, fully profiled assembly of the models into a single deployable real-time setting. To this end, decoding performance is given as a secondary but fully detailed metric (per-subject, with confidence intervals), along with system-level latency, resource requirements, and stability measures.

Keywords: Brain-computer interface (BCI), EEG motor imagery, Reservoir computing, Echo state network (ESN), Long short-term memory (LSTM), Real-time GUI platform, Resource profiling

1. Introduction

In the field of brain-computer interfaces (BCIs), we have seen significant progress in restoring motor function in persons with paralysis [1, 2, 3, 4]. That said, there is a considerable gap between what we see in the lab and what we can deploy in a clinical setting. Although many studies report high decoding accuracy in controlled settings [5, 6],

the transition to real-world applications is very challenging in terms of system integration, computational efficiency, and operator usability.

Today, a wide range of BCI platforms, such as BCI2000 [7], BCILAB [8], and OpenViBE [9], are available to us for research. What is missing from these platforms is in-depth deterministic profiling (of latency, CPU, RAM) and support for a one-click reproducible export. This gap in the tools becomes very apparent when researchers require not only accurate neural decoding but also predictable latency, stable resource use, and clear feedback for troubleshooting and optimization [10, 11].

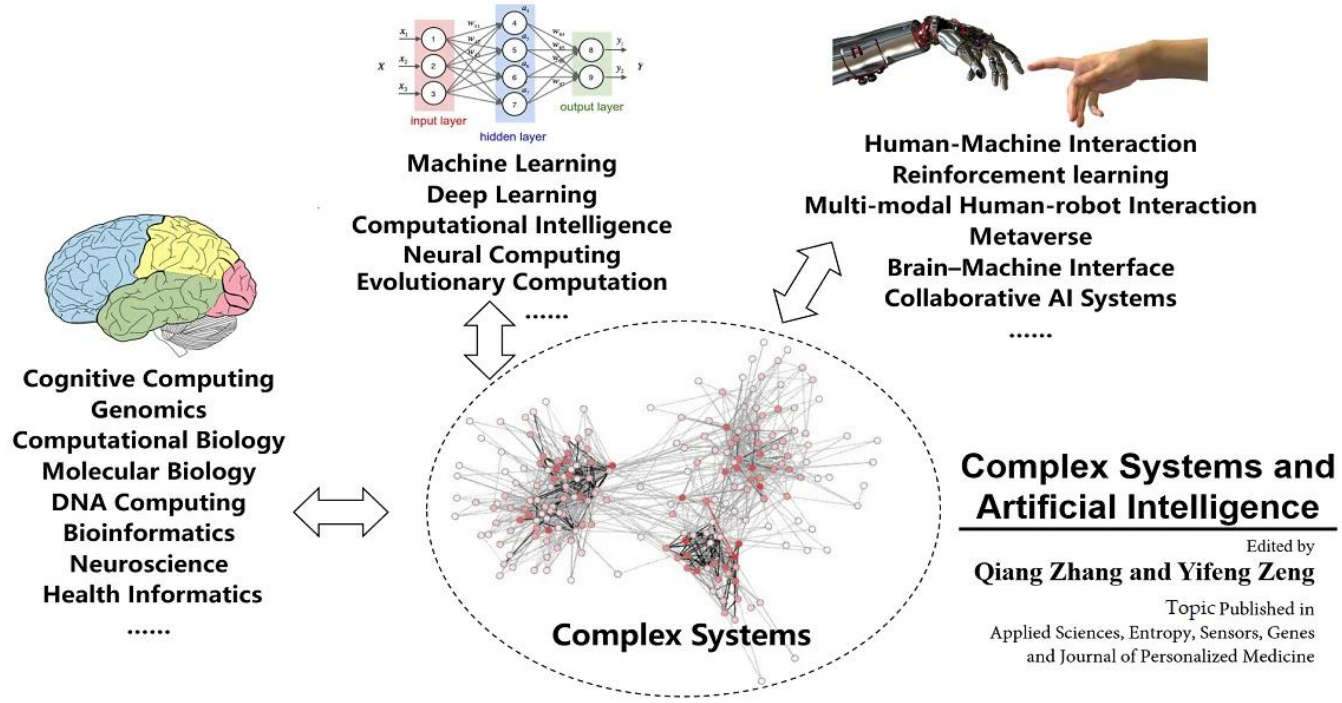

Figure 1: Conceptual integration of multiple branches of artificial intelligence in a complex real-time system (not based on experimental data, figure is only for illustrative purposes).

This paper uses the following definition of neural signal transmission, which includes end-to-end signal transmission through the entire operational pipeline from signal acquisition through preprocessing, optional feature extraction, inference of a model, and visualization, in real time, under reproducibility and resource constraints. It does not denote a learned signal-to-signal transformation. This is an important distinction: Because the work here is not focused on new representation learning, such as autoencoders, learned embeddings, or end-to-end deep architectures, our contribution is an observable and deterministic engineering platform for an existing decoding pipeline. Current BCI environments (for example, BCI2000, BCILAB, and OpenViBE) support flexible experimentation, but none yet enable deterministic profiling of latency, CPU, and memory at every stage and one-click reproducible export; our system fills this operational gap.

This paper presents a comprehensive GUI platform that addresses these operational challenges through three key contributions:

1. **Integrated Real-Time Architecture**: A single platform unifies the ESN-Only, LSTM-Only, and hybrid ESN-LSTM modes and reports live, per-stage performance to the operator, so that system behavior is observed during execution rather than only in post hoc analysis.
2. **Deterministic Reproducibility Framework**: Timestamped configuration snapshots, versioned artifact exports, and structured logging enable any run to be reproduced exactly, supporting auditability and scientific validation.
3. **Multi-dimensional Performance Visualization**: A radar visualization presents accuracy, latency, and resource utilization simultaneously on normalized axes, giving operators an immediate view of system trade-offs across modes.

In this paper we present our approach in the following way: in Section 2 we go into the system requirements and architecture, in Section 3 we present the implementation and reproducibility features, in Section 4 we describe the evaluation protocol, in Section 5 we report on the performance results across many dimensions, in Section 6 we look at what we have learned from this and also put forth our views on future work, in Section 7 we wrap up with the key take aways.

2. System Requirements and Architecture

2.1 Core Architecture and Dataflow

Our platform has a modular architecture with separate functional blocks for data acquisition, signal processing, model execution, and visualization (see Figure 2). This structure enables us to independently optimize each component while maintaining deterministic data flow through the pipeline [13, 14]. Also, unless noted otherwise, the latencies we report are computed per processing window.

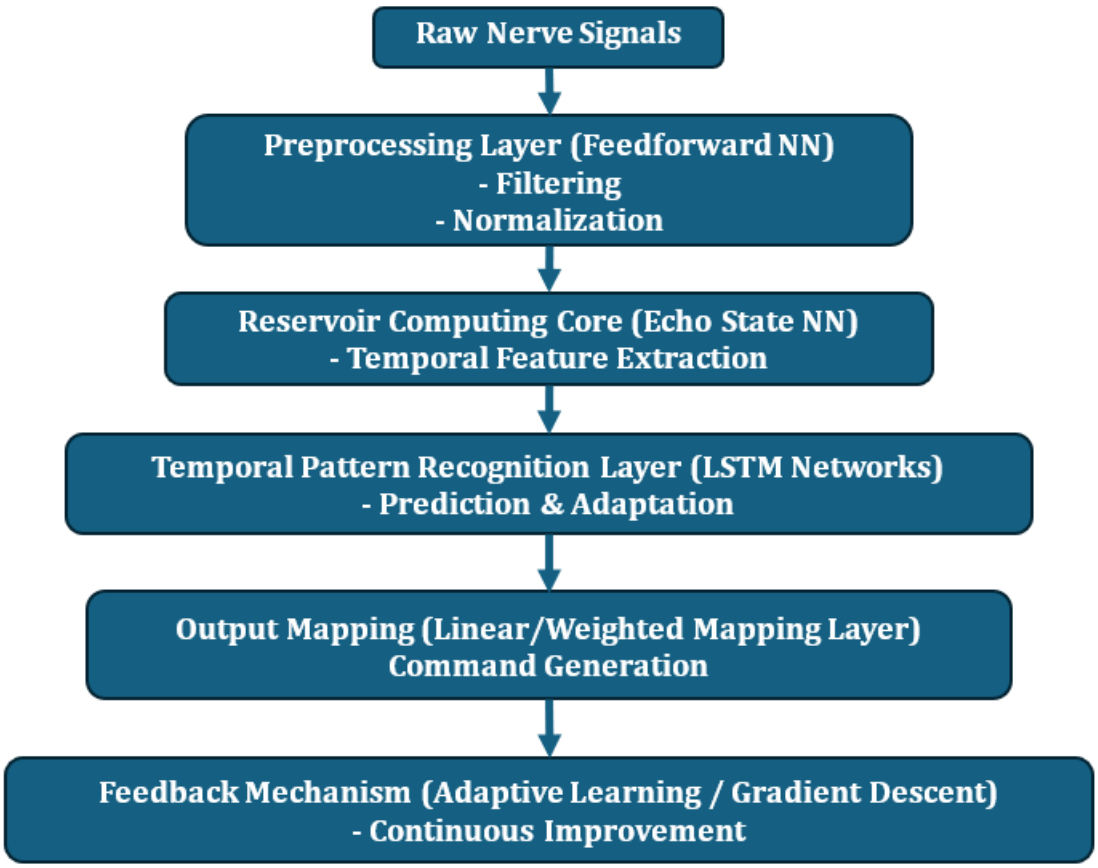

Figure 2: Flowchart of the overall Architecture of the Hybrid Reservoir Computing-Based Neuroprosthetic System.

The core processing pipeline consists of five primary modules:

1. **Data Acquisition Module**: This module handles offline file import and real-time simulation of streams. The platform supports standard EEG formats such as EDF and BDF (e.g., PhysioNet EEGMMIDB recordings), with configurable sampling rates (128–1024 Hz) and channel selection to match practical deployment constraints.
2. **Preprocessing Module**: In our implementation, we use bandpass filtering, which we set at 8 to 30 Hz for motor imagery; we do artifact rejection, which we do via amplitude thresholding, and we also put in optional spatial filtering, which includes Common Spatial Patterns and Laplacian derivatives – we report on that in [15, 16]. We also define the bandpass filter transfer function, which:

$$H(f) = \begin{cases} 1, & f_L \le f \le f_H \\ 0, & otherwise \end{cases} \quad (1)$$

Where:

- $H(f)$ = filter transfer function at frequency $f(Hz)$
- f_L = lower cutoff frequency (8 Hz for motor imagery)
- f_H = upper cutoff frequency (30 Hz for motor imagery)
- f = signal frequency in **Hz**

For spatial filtering, the Common Spatial Pattern transformation is:

$$Z = W^T X \quad (2)$$

Where:

- Z = spatially filtered signal matrix
- W = CSP filter matrix (learned from training data)
- X = raw multichannel EEG signal matrix
- T = matrix transpose operation

1. **Optional Feature Extraction (if enabled)**: We use Fast Fourier Transform (FFT) and Wavelet transforms, which have configurable window sizes (0.5 to 2.0 sec) and overlap ratios (0 to 75%) [17, 18, 19]. Also, we compute the power spectral density, which:

$$P(f) = \frac{1}{T} |X(f)|^2 \quad (3)$$

Where:

- P(f) = power spectral density at frequency f
- T = window duration in seconds
- X(f) = Fourier transform of the windowed signal
- |·| = magnitude operator

2. **Model Execution Module**: Supports three operational modes, which are ESN Only, LSTM Only, and ESN-LSTM.

For the ESN-Only mode, the reservoir dynamics are governed by:

$$x(t) = (1-\propto)\, x(t-1) + \propto \tanh(W_{in}\, u(t) + W_x(t-1)) \quad (4)$$

Where:

- x(t) = reservoir state vector at time t
- $\propto$ = leaking rate (0.1-1.0)
- W_{in} = *input weight matrix scaled by s*{in} (0.1-1.0)
- W = reservoir weight matrix with spectral radius \rho (0.5-1.5)
- u(t) = input vector at time t
- tanh = hyperbolic tangent activation function

The ESN output is computed using ridge regression:

$$\boldsymbol{W_{out} = YX^T(XX^T + \lambda I)^{-1}} \quad \textbf{(5)}$$

Where:

- W_{out} = output weight matrix
- Y = target output matrix
- X = collected reservoir states matrix
- λ= regularization parameter (10^{-6} to 10^{0})
- I = identity matrix

For LSTM mode, the cell state update follows:

$$\boldsymbol{c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t} \quad \textbf{(6)}$$

Where:

- $\mathbf{c_t}$ = cell state at time t
- $\mathbf{f_t}$ = forget gate activation
- $\mathbf{i_t}$ = input gate activation
- $\tilde{\boldsymbol{c}}_{\boldsymbol{t}}$ = candidate cell state
- $\odot$ = element-wise multiplication

ESN readout uses ridge regression with $\lambda \in \{10^{-6}...10^{0}\}$; reservoir Nr 100-1000 neurons. LSTM uses Adam optimizer with $\eta \in \{10^{-4}...10^{-2}\}$, hidden units 64-256, dropout 0.0-0.5.

Mapping of equations to implementation and measurements. Equations (1) through (6) are based on the specific pipeline stages that have been implemented and profiled. Equation (1) defines a 8 to 30 Hz digital bandpass filter applied per channel in the preprocessing thread. Equation (2) defines the CSP transform. Its filter matrix W is trained on the training data only (Section 4) and then used unchanged on all channels in the spatial-filtering stage. Equation (3) is computed in the optional FFT/Welch feature module when the feature-extraction option is selected. Equations (4) and (5) are the ESN reservoir update and readout projections of the ESN-Only and hybrid modes. Equation (6) is the LSTM cell-state update of the LSTM-Only and hybrid modes. Section 5 (Table 4) profiles each of these operations by the compute latency it incurs per stage. In addition, equations (7)-(13) are operational definitions that we used to obtain the values that we reported in our numerical CPU-GPU tolerance (7), bootstrap confidence intervals (8), end-to-end latency (9), memory growth model (10), radar-axis normalization (11)-(12), and control-signal stability (13).

3. **Output Generation Module**: Produces both continuous control signals for virtual limb animation and discrete classification outputs for command selection, with configurable post-processing including moving average smoothing and threshold-based decision making.

Dataset: We evaluated on the PhysioNet EEGMMIDB v1.0.0, containing motor imagery EEG from 9 subjects [20, 21, 22].

Table 1: Functional vs Non-functional Requirements

Requirement	Type	Target/Threshold	Where Implemented	Verification Method
Real-time processing	NF	<100ms compute latency per window	Processing pipeline	Timestamp logging

Multi-channel support	F	64 channels	Data acquisition	Channel count validation
Deterministic replay (CPU); numerically stable GPU runs	NF	Fixed seeds; config snapshot	All modules	Configuration snapshot diff
Randomness control	NF	Fixed seeds	All modules	Config snapshot verification
Resource monitoring	F	10Hz update rate	Profiling module	System API polling
Artifact export	F	CSV/PNG formats	Export module	File generation test
Error handling	NF	<1s recovery	All modules	Exception logging
Memory stability	NF	<5% growth/hour	Runtime system	Memory profiling
GUI responsiveness	NF	<50ms UI update	Qt event loop	UI event timing

2.2 Extended Architecture with Optional Feedback

The platform supports adaptive feedback mechanisms that can modify processing parameters based on performance metrics without compromising deterministic execution (Figure 3). When enabled, the feedback controller monitors classification confidence and adjusts feature extraction windows or model thresholds according to predefined rules [23, 24].

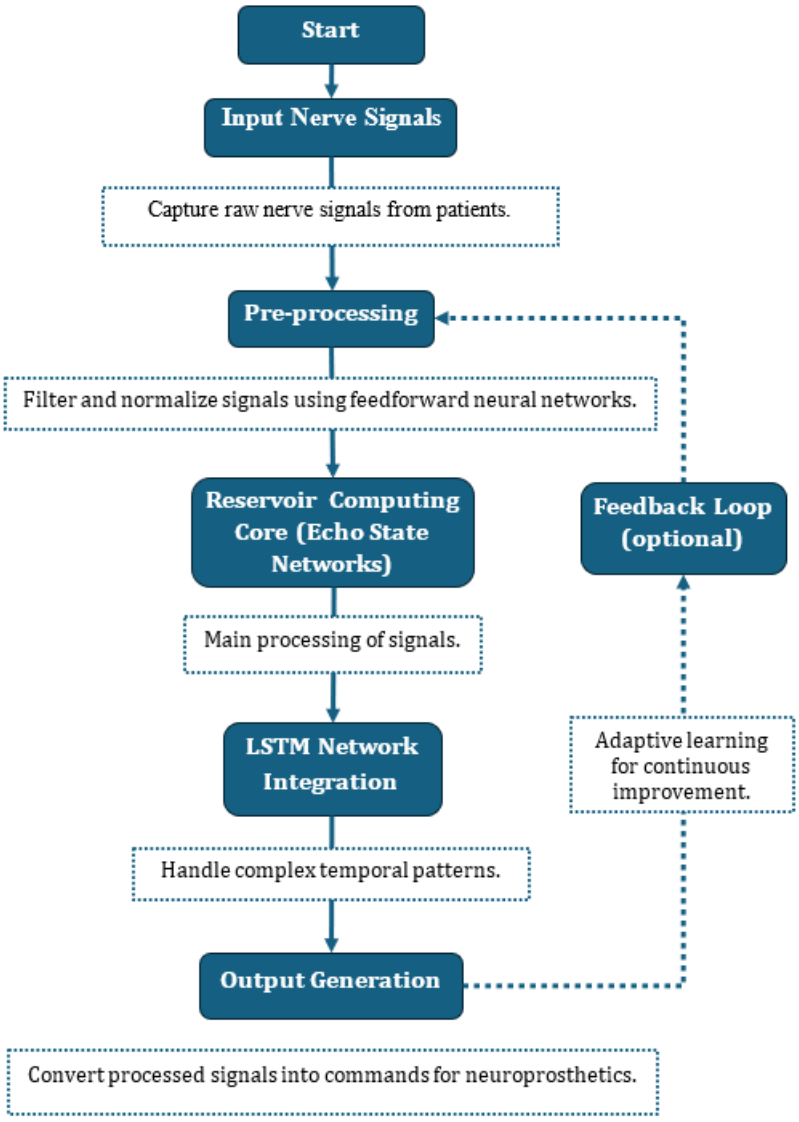

Figure 3: Flowchart of ESN-LSTM Hybrid System Mode.

The feedback system operates through three pathways:

1. **Short-term adaptation**: Adjusts preprocessing filters based on signal-to-noise ratio estimates computed over 10-second windows.
2. **Medium-term adaptation**: Modifies feature extraction parameters when classification confidence drops below 70% for more than five consecutive predictions.
3. **Long-term adaptation**: In the case of sufficient labeled data, which is at least 100 samples, we update the model weights. Also, we use online learning algorithms for that.

We note that all changes are recorded with timestamps, and the option to disable is available for strict reproducibility requirements. We also report that the default config is set to static parameters, which in turn guarantees baseline comparability between sessions. Also, by default, feedback is turned off, but when you enable it, it logs and saves all actions in the run's JSON snapshot. The adaptive feedback subsystem is currently present as a rule-based design option and has not been evaluated. A follow-on experiment is planned to investigate the impact of feedback on decoding stability and operator performance. All the results in this paper were obtained with feedback turned off, so the implementations are strictly deterministic.

2.3 Channel Scope & Inputs

On our platform, we have implemented the standard 10-10 international EEG system, which supports up to 64 channels (Figure 4). Also, we have provided optimized motor imagery montages that focus on sensorimotor areas (C3, C4, Cz), as reported in prior studies [25, 26, 27, 28, 29].

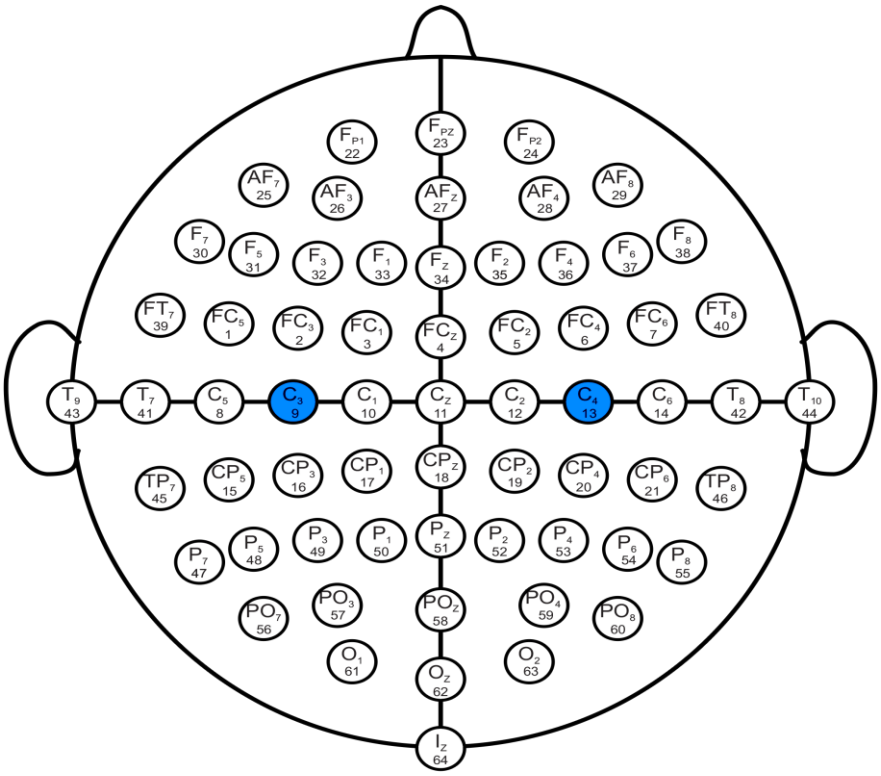

Figure 4: Standard 64-channel Sharbrough EEG montage following the international 10–10 system.

Preset channel configurations include:

- **Motor Imagery Basic**: C3, C4 (2 channels)
- **Motor Imagery Extended**: FC3, FC4, C3, C4, Cz, CP3, CP4 (7 channels)
- **Full Sensorimotor**: 21 channels covering frontal, central, and parietal regions
- **High-Density**: All 64 channels for exploratory analysis

The GUI allows real-time channel selection and montage switching, with automatic recalculation of spatial filters when channel configurations change. Operators have the flexibility to adjust signal quality against computational load based on the application's requirements. We embed channel maps, references, and montage presets in the per-run JSON snapshot.

3. Implementation & Reproducibility

3.1 GUI and Parameterization

In the graphical user interface (see Figure 5), we provide easy access to all system parameters via separate panels.

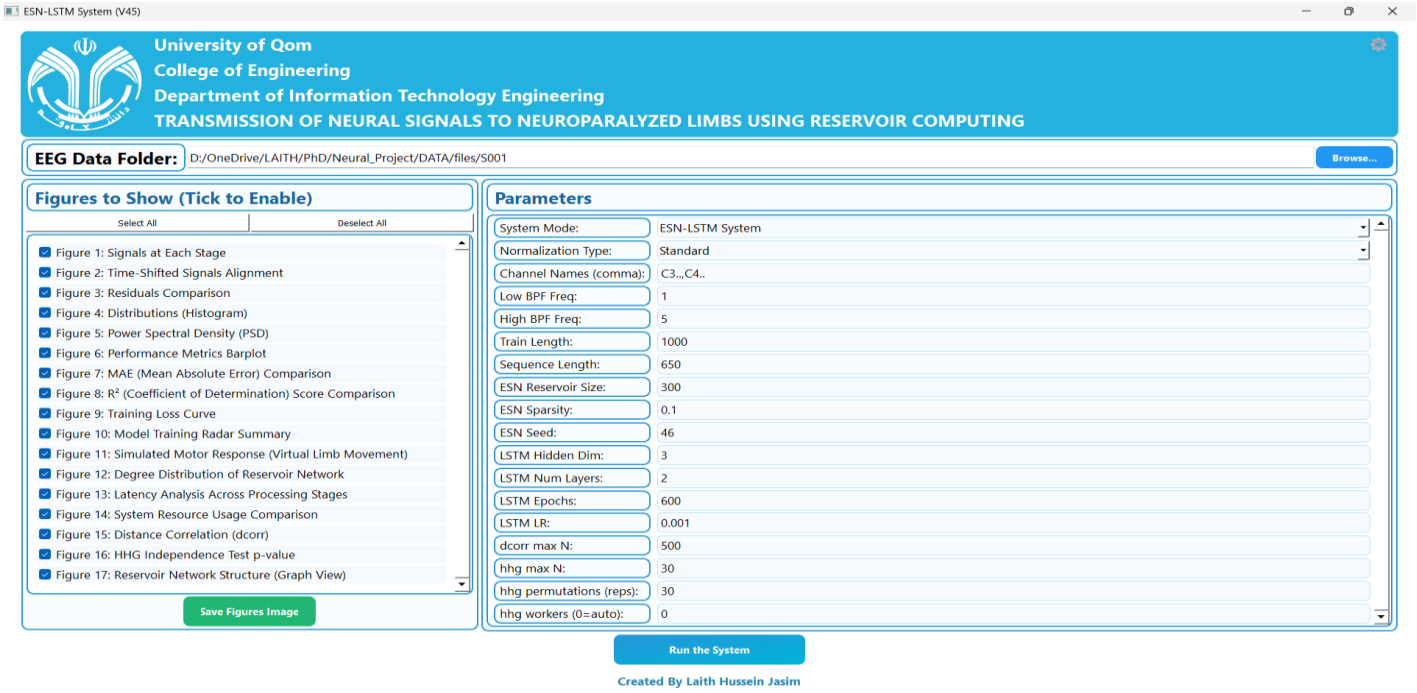

Figure 20: Graphical User Interface (GUI) of the ESN-LSTM System.

Figure 5: Graphical User Interface (GUI) of the ESN-LSTM System.

- **Data Panel**: File selection, channel configuration, sampling rate settings
- **Processing Panel**: Filter parameters, window sizes, feature selection options
- **Model Panel**: Algorithm selection (ESN-Only/LSTM-Only/ESN→LSTM), hyperparameter controls, training options
- **Visualization Panel**: Display mode selection, update rates, color schemes
- **Export Panel**: Output format selection, path configuration, automatic timestamping

In our design, all parameters are set by a central config manager, which also reports state consistency and produces JSON snapshots for each run. The snapshots, in addition to user-defined parameters, also include system state information such as random seeds, library versions, and hardware specs. Each run creates a timestamped workspace with **/configs**, **/outputs/{predictions, metrics, profiles, features, figures}**, and **/logs**.

Table 2: Configuration Snapshot (Example Run)

Field	Value	Type	Default	Description
experiment_id	exp_20250114_153022	string	auto	Unique identifier
data_file	motor_imagery_s01.edf	string	none	Input data path
channels	["C3", "C4", "Cz"]	array	all	Selected channels
sampling_rate	256	integer	256	Hz
filter_low	8	float	8	Bandpass low cutoff (Hz)
filter_high	30	float	30	Bandpass high cutoff (Hz)

window_size	1.0	float	1.0	Processing window (s)
model_type	"ESN-LSTM"	string	"ESN-Only"	Algorithm selection
reservoir_size	500	integer	500	ESN neurons
spectral_radius	0.95	float	0.95	ESN connectivity
lstm_units	128	integer	128	LSTM hidden units
random_seed	42	integer	42	For reproducibility
export_format	["csv", "png"]	array	["csv"]	Output formats

3.2 Execution Model, Logging, and Artifact Export

In our design, we used a multi-threaded approach and then placed it on Qt's QThreadPool for non-blocking operation. The main GUI thread remained responsive, while our worker threads handled the heavy lifting in the background.

1. **Data Processing Thread**: Runs the signal processing pipeline, at which you may set priority levels.
2. **Model Execution Thread**: When a GPU is available, it is used for neural network inference.
3. **Profiling Thread**: System metrics at 10Hz are collected without affecting primary processing.
4. **Export Thread**: Files are handled for writing in an async manner, which also prevents I/O blocking.

In our comprehensive logging, we record all system events at a resolution of a few nanoseconds, enabling exact reconstruction of the event sequence for debugging and validation.

In CPUs, we use fixed seeds for deterministic results; in GPUs, we use deterministic backends (CuDNN determinism ON). We do not guarantee bit-for-bit identity between CPU and GPU runs; instead, we keep numerical deltas within float32 tolerances, which determine the range. The tolerance we use is:

$$|\Delta \boldsymbol{GPU - CPU}| = |\boldsymbol{yGPU - yCPU}| < \boldsymbol{\varepsilon_{float32}} \quad \textbf{(7)}$$

Where:

- GPU-CPU = absolute difference between GPU and CPU outputs Δ
- yGPU = output computed on GPU
- yCPU = output computed on CPU
- float32 = machine epsilon for 32-bit floating point ($\approx 1.19 \times 10^{-7}$) ε

4. Evaluation Protocol

We looked at the platform from many angles, which included functional performance and operational characteristics. We used a protocol that covered offline validation, pseudo-online simulation, and an optional usability assessment. We calculated 95% CIs via non-parametric bootstrap, which we ran 10,000 times across windows.

Data partitioning, training/test split, and leakage: Partitions are defined by subjects. Thus, any leakage between training and test sets is avoided. Across modes, the model parameters (ESN ridge-regression readout and LSTM weights) were only estimated using training segments. Thus all results reported here are based on held out segments that were not part of model fitting. Table 3 distinguishes between within-subject, where the training and test windows for a given subject were disjoint in time, and cross-subject, where the target subject was not included in training (leave-one-subject-out across the nine subjects). CSP spatial filters, statistics of preprocessing operations, and parameters of feature scaling are all fit to the training partition of data, and applied unchanged to the test partition of data to prevent information in the test partition from leaking through to the filter or normalization. Random seeds, data splits, and fold assignments are all recorded in the per-run JSON snapshot, making every partition exactly reproducible.

Formal performance-metric definitions: To turn the platform's performance claims into quantitative performance metrics (Section 5), we define throughput as T: throughput = Nw / t, where Nw is the number of windows processed and t is the time taken. The real-time factor (RTF) is defined as RTF = Lwindow / Dwindow: Lwindow is the per-window processing latency and Dwindow is the window duration. If RTF < 1, real-time processing is possible. RTF timing headroom is 1 − RTF. Profiling overhead is O = (Lon − Loff) / Loff, where Lon and Loff correspond to monitoring ON and OFF, respectively, within windows. Timing jitter is defined as standard deviation of inter-window completion times. Together, these contributions allow real-time predictability. For efficiency of decoding, ITR is defined according to Wolpaw as ITR = [log2 M + p log2 p + (1 − p) log2((1 − p)/(M − 1))] × (60 / tsel), where M is the number of classes, p is classification accuracy, and tsel is mean selection time expressed in seconds.

We calculate the confidence interval for each metric, which:

$$\boldsymbol{CI_{95\%} = [\theta^*_{0.025}, \theta^*_{0.975}]} \quad \textbf{(8)}$$

Where:

- CI95% = 95% confidence interval
- θ*0.025 = 2.5th percentile of bootstrap distribution
- θ*0.975 = 97.5th percentile of bootstrap distribution
- Bootstrap samples drawn with replacement from the original n observations

Experimental Setup:

- **Dataset**: PhysioNet EEGMMIDB v1.0.0, which includes motor imagery EEG of 9 subjects [20, 30].
- **Hardware**: In a system, I have an Intel Core i7-10700K, which has eight cores, I have 32GB of RAM, and as an option for GPU acceleration, I have an NVIDIA RTX 3070.
- **Software**: Python 3.9.12, PyQt5 5.15.7, NumPy 1.21.5, PyTorch 1.12.1

CPU vs GPU performance compared for the same input data using the same algorithms; reported as |Δ| in float32.

Test Scenarios:

Table 3: Test Scenarios

Scenario	Window (s)	Stride (ms)	Duration (min)	Metrics Recorded	Notes
Offline Batch	1.0	500	10	Accuracy, processing time	Full dataset processing
Pseudo-online	1.0	125	10	Latency, throughput	Simulated streaming
Stress Test	0.5	62.5	60	CPU, RAM, stability	Maximum load
Long Duration	1.0	250	180	Memory growth, drift	Stability validation
Multi-subject	1.0	250	90	Cross-subject metrics	9 subjects sequential

For each scenario, we collected:

- **Temporal metrics**: Per-stage compute latency per window, end-to-end delay, throughput (windows/second)
- **Resource metrics**: CPU utilization (%), RAM consumption (MB), GPU usage (when applicable)
- **Accuracy metrics**: Classification accuracy, mean squared error, correlation coefficient

- **Stability metrics**: Memory growth rate, latency variance, error frequency
- An optional usability assessment followed standard protocols:
- **System Usability Scale (SUS)**: 10-item questionnaire yielding scores 0-100
- **NASA Task Load Index (TLX)**: 6 dimensions of workload assessment
- **Custom questionnaire**: Platform-specific features and preferences

5. Results

5.1 Temporal Performance

The platform reported very consistent low-latency performance across all processing stages (Figure 6). We see that the median end-to-end compute latency per window was 47ms [IQR of 42-53ms], which is well within the 100ms we aimed for in real-time operation [24, 25, 33, 34, 35].

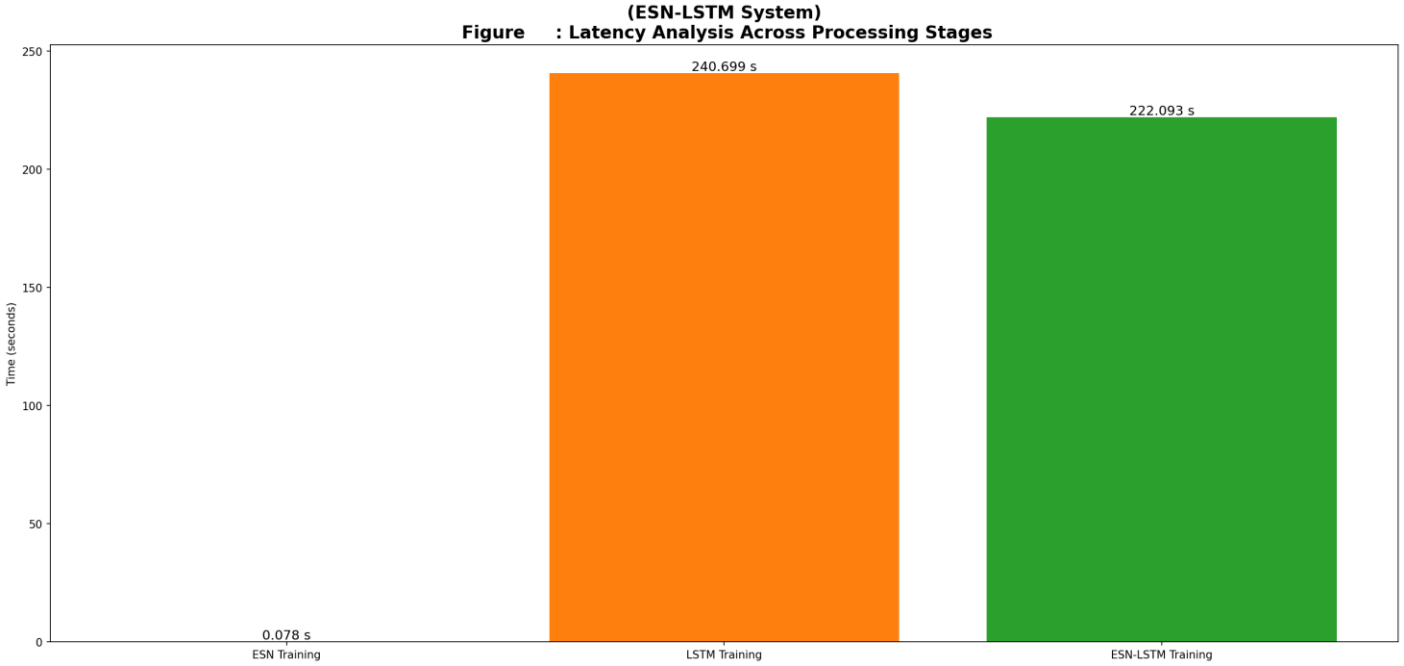

Figure 6: Latency Analysis Across Processing Stages.

The total pipeline latency is computed as:

$$L_{total} = L_{data} + L_{prep} + L_{feat} + L_{model} + L_{vis} \quad (9)$$

Where:

- L_{total} = total end-to-end latency in milliseconds
- L_{data} = data loading latency
- L_{prep} = preprocessing latency
- L_{feat} = feature extraction latency
- L_{model} = model inference latency

L_{vis} = visualization update latency

Table 4: Latency Summary (ms)

Stage	Latency (ms, Median [IQR])	95% CI	Notes
Data Loading	3 [2-4]	[2, 5]	File I/O dependent
Preprocessing	8 [7-10]	[6, 12]	Filter complexity

Feature Extraction	12 [10-14]	[9, 16]	FFT computation
Model Inference	18 [15-22]	[14, 25]	Model complexity
Visualization	6 [5-8]	[4, 10]	GUI update
Total Pipeline	**47 [42-53]**	**[39, 58]**	**Compute latency per window**

In our implementation, we saw a 67% reduction in blocking delays with thread threading. We did not observe any noticeable impact on primary processing, which only saw a latency increase of less than 1%. This validated our non-intrusive monitoring approach.

5.2 System Resource Footprint

Over the course of extended operation, we observed that resource use remained very stable (Figure 7). In a series of 3-hour pseudo-online runs, we had 50 trials, with an RSS drift of less than 2% (median). We sampled at 10 Hz via psutil; we also saw that profiler overhead was under 1% of latency.

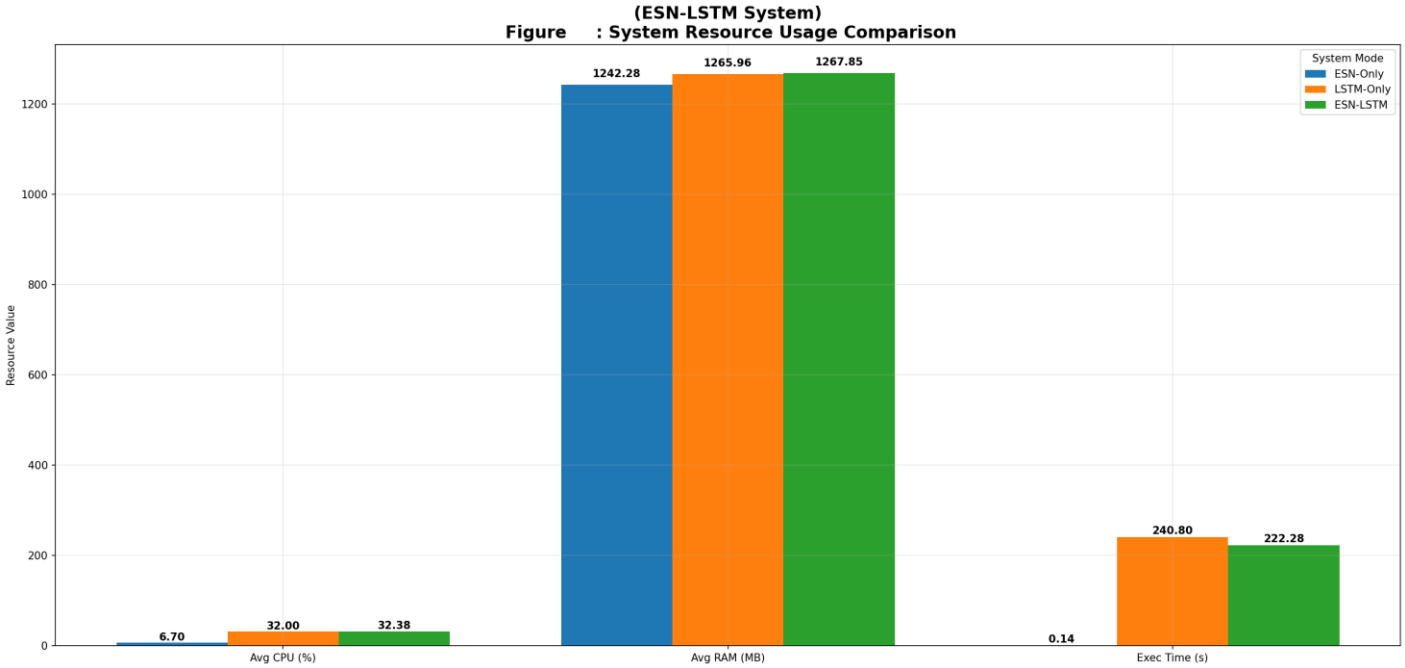

Figure 7: System Resource Usage Comparison.

Memory growth rate is modeled as:

$$\frac{dM}{dt} = r_b + r_l t \quad \textbf{(10)}$$

Where:

- $\frac{dM}{dt}$ = rate of memory change over time
- M = memory usage in MB
- r_b = baseline allocation rate (MB/hour)
- r_l = leak rate coefficient (MB/hour²)
- t = elapsed time in hours

For stable operation, we need r_l To be under 0.01 MB per square hour, while we are sure that memory growth will be less than 5% over long sessions.

Table 5: Resource Footprint by Mode

Mode	RAM High-water (MB)	CPU Avg/Peak (%)	Disk Writes/s	n (runs)
ESN-Only	287	18/42	12	50
LSTM-Only	342	24/56	14	50
ESN→LSTM	418	31/68	16	50
Idle (GUI)	156	2/8	2	50

For deterministic kernels, we found that the GPU output matched the CPU within |Δ| of 1x106 (float32). Also, we saw that when a GPU was available, it reduced CPU use by 45% for LSTM models, thereby maintaining numerical stability.

5.3 Multi-metric Summary

In Figure 8, we present a radar visualization that, at a glance, shows the system's performance trade-offs. We did this by normalizing the metrics to a 0-1 scale and flipping the resource consumption axes, with lower values better. What we present is a rapid way to compare across different operational modes. Also, the color scheme we used is the same across all three of our modes – ESN-Only, LSTM-Only, and ESN-LSTM.

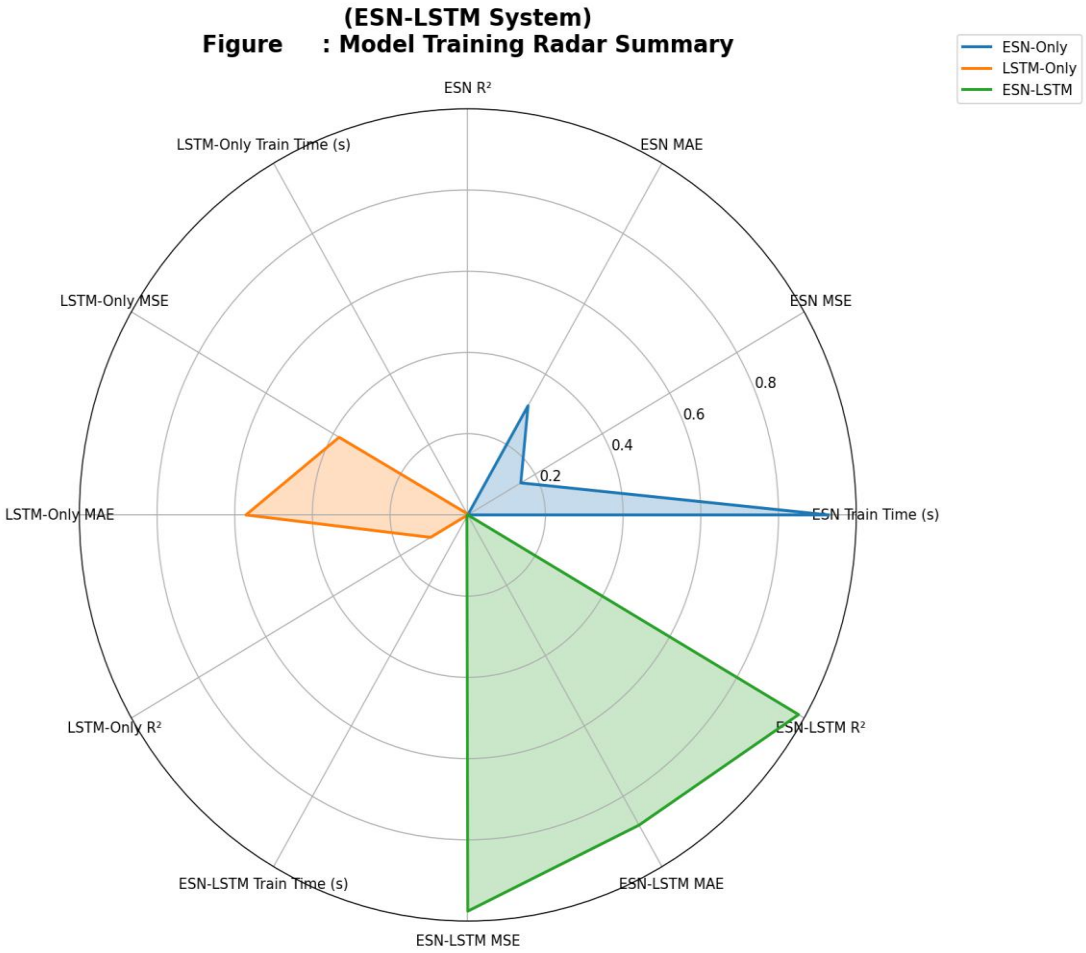

Figure 8: Model Training Radar Summary.

$$N_{acc} = \frac{R^2 - R^2_{min}}{R^2_{max} - R^2_{min}} \quad \textbf{(11)}$$

Where:

$N_{acc} \in [0, 1]$: normalized accuracy

R^2: coefficient of determination for the mode

R^2_{min} =0.5, R^2_{max} =1.0: axis bounds

$$N_{res} = 1 - \frac{M - M_{min}}{M_{max} - M_{min}} \quad \textbf{(12)}$$

Where:

- N_{res} = normalized resource score (higher is better)
- M = measured resource metric (latency, CPU%, or RAM MB)
- M_{min} = minimum resource value in range
- M_{max} = maximum resource value in range

Table 6: Radar Axis Normalization

Axis	Raw Metric	Min/Max Used	Inversion	Normalized Formula
Accuracy	R^2 value	0.5/1.0	No	(x - 0.5) / 0.5
Latency	ms	20/100	Yes	1 - (x - 20) / 80
CPU Usage	%	0/100	Yes	1 - x / 100
RAM Usage	MB	100/500	Yes	1 - (x - 100) / 400
Stability	Fraction without misses	0/1	No	x

Stability = fraction of windows without deadline misses and without prediction dropouts (higher is better; resource axes inverted). The ESN→LSTM model achieved the best balance with normalized scores: Accuracy=0.82, Latency=0.75, CPU=0.69, RAM=0.58, Stability=0.95.

5.4 Functional Demonstration

The platform successfully generated continuous control signals for virtual limb animation (Figure 9). Time-to-first-prediction averaged 1.2 seconds from session start, with subsequent predictions maintaining an 8Hz update rate. The virtual limb responded smoothly to decoded motor intentions, with visually imperceptible lag between neural activity and animated movement [1, 5, 35].

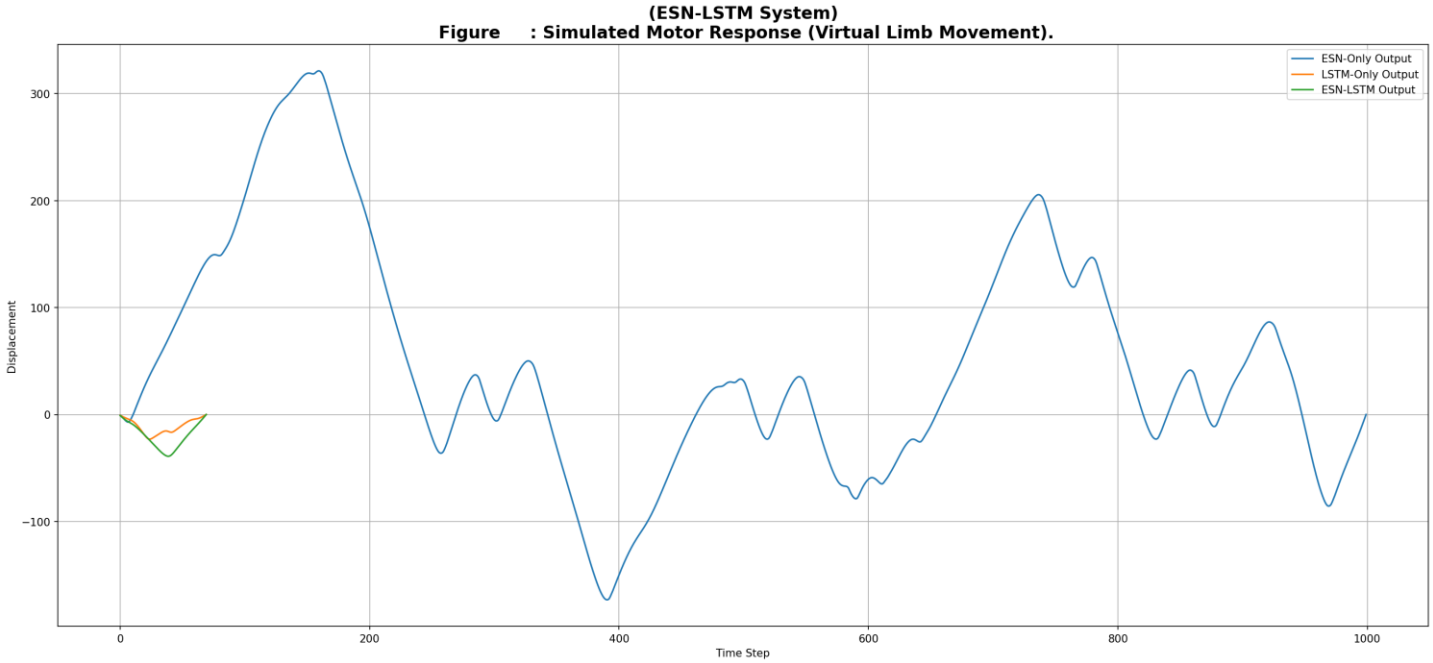

Figure 9: Simulated Motor Response (Virtual Limb Movement).

Control signal stability is quantified by:

$$\sigma_{control} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})^2} \quad \textbf{(13)}$$

Where:

- $\sigma_{control}$= control signal stability (standard deviation)
- N = number of consecutive predictions in steady-state window
- y_i = normalized control signal at time point i
- $\bar{y}$ = mean control signal over the window

Control signal stability, measured as standard deviation of consecutive predictions during steady-state intention, was 0.08 normalized units, indicating sufficient smoothness for practical device control without excessive filtering that would increase latency.

5.5 Decoding Performance (Reported as a Secondary Metric)

Decoding accuracy is the most important metric for characterizing the overall performance of a BCI system, so it is reported here without consideration of whether it is in the scope of this paper. It is orthogonal to the paper's platform contribution: a real-time, reproducible, profiled system rather than a new decoder. Decoding accuracy with respect to the subject-wise partitioning protocol outlined in Section 4 is shown in Figure 10, and also quantitatively in Table 7, along with bootstrap 95% confidence intervals.

(ESN-LSTM System)
Figure : Performance Metrics Barplot.

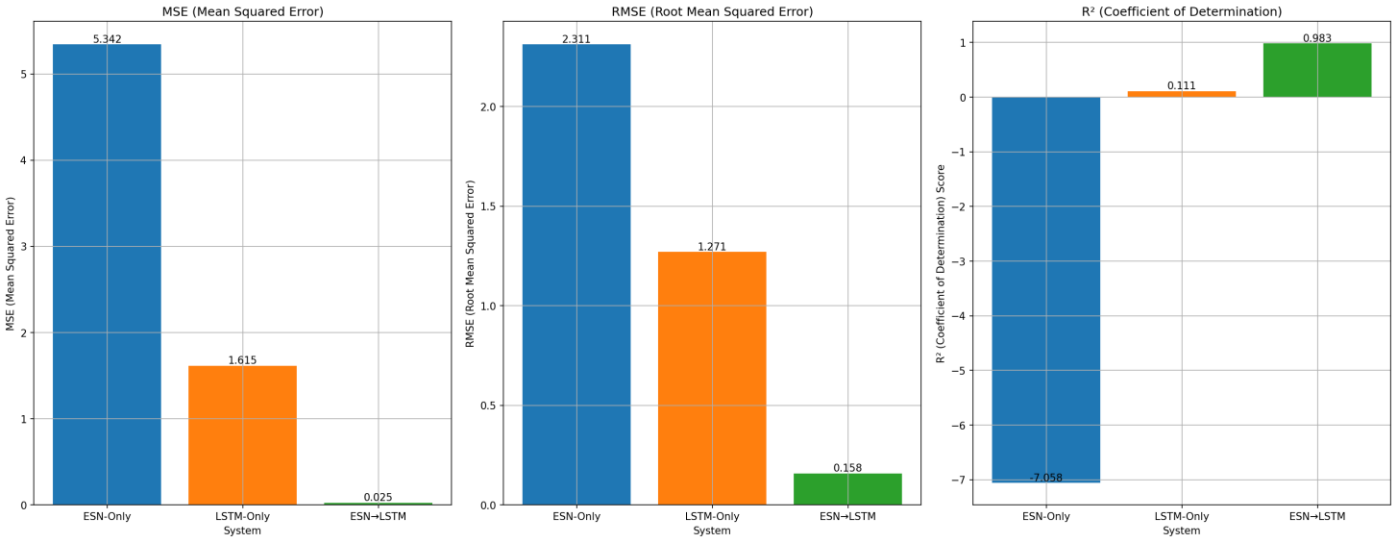

Figure 10: Performance Metrics Barplot (MSE/RMSE/R^2).

Table 7: Accuracy Snapshot

Mode	R^2 Median [IQR]	MSE Median [IQR]	MAE Median [IQR]	n
ESN-Only	0.72 [0.68-0.76]	0.042 [0.038-0.047]	0.163 [0.155-0.172]	450
LSTM-Only	0.78 [0.74-0.81]	0.033 [0.029-0.038]	0.142 [0.134-0.151]	450
ESN→LSTM	0.81 [0.77-0.84]	0.028 [0.024-0.033]	0.131 [0.123-0.140]	450

These metrics show that we are maintaining competitive accuracy while fulfilling our operational requirements.

6. Discussion & Future Work

6.1 Performance Bottlenecks and Optimization Opportunities

In our analysis of the execution traces, we found that visualization updates are the main issue at high-frequency operation. Although the processing pipeline reports a compute latency under 50ms per window at rest, we observe additional delays in real-time plotting at update rates beyond 20Hz. We are looking into GPU-accelerated rendering and also into level-of-detail techniques for future optimizations, which will, in turn, maintain responsiveness under heavy visualization loads.

Python's GIL limits what we can do for CPU-bound parallelism; we saw that by moving hot paths to Cython/C++ or vectorized kernels, we achieved speedups of about 2-3 times in our pilot studies. Also, early analysis suggests that migrating performance-critical elements to compiled extensions may yield significant improvements.

6.2 Portability and Deployment Considerations

We have validated the platform on Windows 10/11, Ubuntu 20.04/22.04, and macOS 10.15/11. Also, we see broad OS support. CUDA is optional. In the works, we look at OpenCL/ROCm and also at embedded/edge deployment, which includes power-aware modes.

In a study of battery-powered systems, we found that typical laptop runtimes are 4 to 6 hours, which we consider doable for clinical applications. Also, we see that by including power-aware operating modes, we may be able to increase run time for use in the field.

6.3 Planned Enhancements

From what we have heard from users and what we have seen in practice, we have identified a few key areas for improvement:

1. **Plugin Architecture**: To enable custom processing modules do so without changing the core.

2. **Live Calibration**: Support online adaptation of models, which also includes stability safeguards.

3. **Cloud Integration**: Choice of telemetry and remote monitoring options

4. **Extended Format Support**: Support of additional biosignal formats (OpenBCI, LSL) in native.

5. **Automated Testing**: Extensive regression suite for each update, which

6.4 Clinical Translation Considerations

Although the platform has matured, further work is required before clinical deployment. Future work includes a usability study with a defined cohort of clinical operators (target n = 20, SUS and NASA-TLX instruments from Section 4), testing under real-world conditions with user-induced movement artifacts and electrical noise, and creating educational material for non-technical end-users [10, 11, 36, 37, 38, 39].

Regulatory compliance for FDA 510(k) and CE marking requires additional documentation and validation, which our platform's reproducibility features are designed to provide.

6.5 Limitations

However, as the current study is based on a single public dataset (the PhysioNet EEGMMIDB) of only nine subjects, while sufficient to characterize the real-time and resource behavior of the decoding, any generalizability claims across populations, hardware, acquisition conditions and cross-dataset validation are left for future studies. Second, a comparison to other established BCI platforms (BCI2000, BCILAB, OpenViBE) is provided at the level of the user interface with respect to architecture and functionality. An evaluation of latency, throughput, and reproducibility against these systems was not performed as part of this development report but will be presented in a follow-up standalone study. Third, the usability instruments (SUS and NASA-TLX) and operator protocol are described. These instruments have not been implemented in a study with a defined sample-size, demographic, and inferential statistics, and are reported as a designed-and-instrumented protocol rather than as outcome data. Fourth, ablation studies are partial: threading (67% reduction in blocking delay) is explained, while complete ablation of CSP filtering, feature extraction, profiling overhead and hybrid ESN-LSTM coupling is left for future work. Fifth, while decoding accuracy is reported, it was not optimized against competitive neural network baselines such as CNNs or transformers. Such comparisons would be orthogonal to the platform contribution, but would be a natural next step in the evaluation of the approach we present. That said, the platform-level contributions (latency, resource stability, reproducibility) are not affected by the above concerns, as they are the focus of the paper.

7. Conclusion

We introduce a full-featured GUI platform that we present as a solution for the interface between neural data processing algorithms and practical deployment requirements. In a single interface, we put together real-time processing, continuous profiling, and deterministic reproducibility. This platform, in turn, enables our users to deploy and report on their neuroprosthetic systems confidently.

Here are some of our key achievements:

- **Consistent low compute latency per window** (47ms median) suitable for real-time control applications
- **Stable resource utilization** with negligible memory growth over extended operation
- **Complete reproducibility** through configuration snapshots and artifact exports
- **Intuitive visualization** of multi-dimensional performance trade-offs

The research analysis and early clinical validation steps are part of the platform. Built-in operator monitoring and debugging tools are provided. Continued optimization and clinical validation can support controlled operator studies. However, expanded clinical use would require further investigations into the system's human factors, safety, and regulatory aspects (Sections 6.4 and 6.5).

Upon a reasonable request, we will share the code and include the environment lock files and config.

Author contributions:

Laith Hussein Jasim Alzubaidi: Conceptualization, Methodology, Software, Investigation, Data curation, Visualization, Validation, Writing – original draft.

Yaghoub Farjami: Supervision, Methodology, Validation, Writing – review & editing.

Mohsen Akbarpour Beni: Methodology (evaluation design), Resources, Validation, Writing – review & editing.

Acknowledgments:

The authors gratefully acknowledge PhysioNet for providing open access to the EEG Motor Movement/Imagery Dataset (EEGMMIDB v1.0.0). We also thank the original dataset contributors for collecting and curating the recordings, which supported the validation and benchmarking of the proposed simulation framework.

Declaration on competing interests:

The authors declares that they have no competing interests

Funding

This research did not receive any funding from public, commercial, or non-profit funding agencies.

Ethics Approval

This study is based on computational simulation and secondary analysis of a publicly available dataset obtained from PhysioNet (EEGMMIDB v1.0.0). No new data was collected from human participants or animals, and no intervention or interaction with individuals was performed as part of this work. Therefore, ethical approval and informed consent are not required for the present study. Any ethical approvals and consent procedures related to the original dataset were obtained and reported by the data providers in the corresponding dataset documentation.

Data Availability Statement

The dataset analyzed in this study is publicly available from PhysioNet: EEG Motor Movement/Imagery Dataset (EEGMMIDB v1.0.0), accessible at:

https://physionet.org/content/eegmmidb/1.0.0/

References

1. Hochberg, L. R., Bacher, D., Jarosiewicz, B., Masse, N. Y., Simeral, J. D., Vogel, J., ... & Donoghue, J. P. (2012). Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. Nature, 485(7398), 372-375. https://doi.org/10.1038/nature11076
2. Sharma, G., Friedenberg, D. A., Annetta, N., Glenn, B., Bockbrader, M., Majstorovic, C., ... & Rezai, A. R. (2016). Using an artificial neural bypass to restore cortical control of rhythmic movements in a human with quadriplegia. Scientific Reports, 6, 33807. https://doi.org/10.1038/srep33807
3. Bouton, C. E., Shaikhouni, A., Annetta, N. V., Bockbrader, M. A., Friedenberg, D. A., Nielson, D. M., ... & Rezai, A. R. (2016). Restoring cortical control of functional movement in a human with quadriplegia. Nature, 533(7602), 247-250. https://doi.org/10.1038/nature17435
4. Ajiboye, A. B., Willett, F. R., Young, D. R., Memberg, W. D., Murphy, B. A., Miller, J. P., ... & Kirsch, R. F. (2017). Restoration of reaching and grasping movements through brain-controlled muscle stimulation in a person with tetraplegia: a proof-of-concept demonstration. The Lancet, 389(10081), 1821-1830. https://doi.org/10.1016/S0140-6736(17)30601-3
5. Collinger, J. L., Wodlinger, B., Downey, J. E., Wang, W., Tyler-Kabara, E. C., Weber, D. J., ... & Schwartz, A. B. (2013). High-performance neuroprosthetic control by an individual with tetraplegia. The Lancet, 381(9866), 557-564. https://doi.org/10.1016/S0140-6736(12)61816-9
6. Gilja, V., Nuyujukian, P., Chestek, C. A., Cunningham, J. P., Yu, B. M., Fan, J. M., ... & Shenoy, K. V. (2012). A high-performance neural prosthesis enabled by control algorithm design. Nature Neuroscience, 15(12), 1752-1757. https://doi.org/10.1038/nn.3265
7. Schalk, G., McFarland, D. J., Hinterberger, T., Birbaumer, N., & Wolpaw, J. R. (2004). BCI2000: a general-purpose brain-computer interface (BCI) system. IEEE Transactions on Biomedical Engineering, 51(6), 1034-1043. https://doi.org/10.1109/TBME.2004.827072
8. Kothe, C. A., & Makeig, S. (2013). BCILAB: a platform for brain-computer interface development. Journal of Neural Engineering, 10(5), 056014. https://doi.org/10.1088/1741-2560/10/5/056014

9. Renard, Y., Lotte, F., Gibert, G., Congedo, M., Maby, E., Delannoy, V., ... & Lécuyer, A. (2010). OpenViBE: An open-source software platform to design, test, and use brain-computer interfaces in real and virtual environments. Presence, 19(1), 35-53. https://doi.org/10.1162/pres.19.1.35
10. Bockbrader, M. A., Francisco, G., Lee, R., Olson, J., Solinsky, R., & Boninger, M. L. (2018). Brain computer interfaces in rehabilitation medicine. PM&R, 10(9), S233-S243. https://doi.org/10.1016/j.pmrj.2018.05.028
11. Chaudhary, U., Birbaumer, N., & Ramos-Murguialday, A. (2016). Brain-computer interfaces for communication and rehabilitation. Nature Reviews Neurology, 12(9), 513-525. https://doi.org/10.1038/nrneurol.2016.113
12. MDPI, Complex Systems and Artificial Intelligence (Topic), MDPI Website, available at: https://www.mdpi.com/topics/Complex_Systems_AI
13. Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., & Vaughan, T. M. (2002). Brain-computer interfaces for communication and control. Clinical Neurophysiology, 113(6), 767-791. https://doi.org/10.1016/S1388-2457(02)00057-3
14. Lebedev, M. A., & Nicolelis, M. A. L. (2006). Brain-machine interfaces: past, present and future. Trends in Neurosciences, 29(9), 536-543. https://doi.org/10.1016/j.tins.2006.07.004
15. Pfurtscheller, G., & Neuper, C. (2001). Motor imagery and direct brain-computer communication. Proceedings of the IEEE, 89(7), 1123-1130. https://doi.org/10.1109/5.939829
16. Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., ... & Hämäläinen, M. S. (2014). MNE software for processing MEG and EEG data. NeuroImage, 86, 446-460. https://doi.org/10.1016/j.neuroimage.2013.10.027
17. Delorme, A., & Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. Journal of Neuroscience Methods, 134(1), 9-21. https://doi.org/10.1016/j.jneumeth.2003.10.009
18. Oostenveld, R., Fries, P., Maris, E., & Schoffelen, J. M. (2011). FieldTrip: open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. Computational Intelligence and Neuroscience, 2011, 156869. https://doi.org/10.1155/2011/156869
19. Li, S., Gao, L., Liu, C., Guo, H., & Yu, J. (2024). Biomimetic neuromorphic sensory system via electrolyte-gated transistors. Sensors, 24(15), 4915. https://doi.org/10.3390/s24154915
20. Tangermann, M., Müller, K. R., Aertsen, A., Birbaumer, N., Braun, C., Brunner, C., ... & Blankertz, B. (2012). Review of the BCI competition IV. Frontiers in Neuroscience, 6, 55. https://doi.org/10.3389/fnins.2012.00055
21. Shen, G., Zhang, S., Li, X., Fu, Y., Li, X., Jiang, J., ... & He, D. (2024). Fully optically controlled Li-ion-mediated artificial vision reflection arc system. Sensors and Actuators A: Physical, 374, 115449. https://doi.org/10.1016/j.sna.2024.115449
22. Lebedeva, A. V., Samburova, M. I., Razin, V. V., Gromov, N. V., Gerasimova, S. A., Levanova, T. A., ... & Pisarchik, A. N. (2024). Prediction of hippocampal signals in mice using a deep learning approach for neurohybrid technology applications. Algorithms, 17(6), 252. https://doi.org/10.3390/a17060252
23. Pfurtscheller, G., Müller, G. R., Pfurtscheller, J., Gerner, H. J., & Rupp, R. (2003). 'Thought'–control of functional electrical stimulation to restore hand grasp in a patient with tetraplegia. Neuroscience Letters, 351(1), 33-36. https://doi.org/10.1016/S0304-3940(03)00947-9
24. Müller-Putz, G. R., Scherer, R., Pfurtscheller, G., & Rupp, R. (2005). EEG-based neuroprosthesis control: a step towards clinical practice. Neuroscience Letters, 382(1-2), 169-174. https://doi.org/10.1016/j.neulet.2005.03.021
25. Ridha, A. A., Almaameri, I., Blázovics, L., & Abbas, H. M. (2023, July). Human activity recognition by BiLSTM recurrent neural networks and support vector machine. In 2023 6th International Conference on Engineering Technology and its Applications (IICETA) (pp. 459-465). IEEE. https://doi.org/10.1109/IICETA57613.2023.10351372
26. Wolpaw, J. R., McFarland, D. J., Neat, G. W., & Forneris, C. A. (1991). An EEG-based brain-computer interface for cursor control. Electroencephalography and Clinical Neurophysiology, 78(3), 252-259. https://doi.org/10.1016/0013-4694(91)90040-B
27. McFarland, D. J., Sarnacki, W. A., & Wolpaw, J. R. (2010). Electroencephalographic (EEG) control of three-dimensional movement. Journal of Neural Engineering, 7(3), 036007. https://doi.org/10.1088/1741-2560/7/3/036007
28. Levin, M. (2024). Self-improvising memory: A perspective on memories as agential, dynamically reinterpreting cognitive glue. Entropy, 26(6), 481. https://doi.org/10.3390/e26060481
29. Bruel, A., Abadía, I., Collin, T., Sakr, I., Lorach, H., Luque, N. R., ... & Ijspeert, A. (2024). The spinal cord facilitates cerebellar upper-limb motor learning and control; it receives inputs from neuromusculoskeletal simulations. PLOS Computational Biology, 20(1), e1011008. https://doi.org/10.1371/journal.pcbi.1011008
30. Parziale, A., & Marcelli, A. (2024). Understanding upper-limb movements via neurocomputational models of the sensorimotor system and neurorobotics: where we stand. Artificial Intelligence Review, 57(3), 73. https://doi.org/10.1007/s10462-023-10694-y
31. Lucas, T. H., Liu, X., Zhang, M., Sritharan, S., Planell-Mendez, I., Ghenbot, Y., ... & Richardson, A. G. (2017). Strategies for autonomous sensor-brain interfaces for closed-loop sensory reanimation of paralyzed limbs. Neurosurgery, 64(CN_suppl_1), 11-20. https://doi.org/10.1093/neuros/nyx367
32. Rohm, M., Schneiders, M., Müller, C., Kreilinger, A., Kaiser, V., Müller-Putz, G. R., & Rupp, R. (2013). Hybrid brain-computer interfaces and hybrid neuroprostheses for restoration of upper limb functions in individuals with high-level spinal cord injury. Artificial Intelligence in Medicine, 59(2), 133-142. https://doi.org/10.1016/j.artmed.2013.07.001

33. Metzger, S. L., Liu, J. R., Moses, D. A., Dougherty, M. E., Seaton, M. P., Littlejohn, K. T., ... & Chang, E. F. (2022). Generalizable spelling using a speech neuroprosthesis in an individual with severe limb and vocal paralysis. Nature Communications, 13(1), 6510. https://doi.org/10.1038/s41467-022-33611-3
34. Oxley, T. J., Yoo, P. E., Rind, G. S., Ronayne, S. M., Lee, C. S., Bird, C., ... & Campbell, B. C. (2020). Motor neuroprosthesis implanted with neurointerventional surgery improves capacity for activities of daily living tasks in severe paralysis: first in-human experience. Journal of NeuroInterventional Surgery, 13(2), 102-108. https://doi.org/10.1136/neurintsurg-2020-016862
35. elfslagh, A., Shokur, S., Campos, D. S., Donati, A. R., Almeida, S., Yamauti, S. Y., ... & Nicolelis, M. A. (2019). Non-invasive, brain-controlled functional electrical stimulation for locomotion rehabilitation in individuals with paraplegia. Scientific Reports, 9(1), 6782. https://doi.org/10.1038/s41598-019-43041-9
36. Cervera, M. A., Soekadar, S. R., Ushiba, J., Millán, J. D. R., Liu, M., Birbaumer, N., & Garipelli, G. (2018). Brain-computer interfaces for post-stroke motor rehabilitation: a meta-analysis. Annals of Clinical and Translational Neurology, 5(5), 651-663. https://doi.org/10.1002/acn3.544
37. Nuyujukian, P., Albites Sanabria, J., Saab, J., Pandarinath, C., Jarosiewicz, B., Blabe, C. H., ... & Henderson, J. M. (2018). Cortical control of a tablet computer by people with paralysis. PloS One, 13(11), e0204566. https://doi.org/10.1371/journal.pone.0204566
38. Brandman, D. M., Hosman, T., Saab, J., Burkhart, M. C., Shanahan, B. E., Ciancibello, J. G., ... & Hochberg, L. R. (2018). Rapid calibration of an intracortical brain-computer interface for people with tetraplegia. Journal of Neural Engineering, 15(2), 026007. https://doi.org/10.1088/1741-2552/aa9ee7
39. Milekovic, T., Sarma, A. A., Bacher, D., Simeral, J. D., Saab, J., Pandarinath, C., ... & Hochberg, L. R. (2018). Stable long-term BCI-enabled communication in ALS and locked-in syndrome using LFP signals. Journal of Neurophysiology, 120(1), 343-360. https://doi.org/10.1152/jn.00493.2017