Signal Acquisition Team

Spring 2025 Semester Project
Real-Time Data Validation Pipeline

Longhorn Neurotech - Software Division

January 2025

1 Project Overview

Your mission is to implement a **real-time data validation layer** that catches hardware and signal quality issues *before* data enters the processing pipeline. This is the first line of defense against poor data quality.

1.1 Deliverables

- `data_validation.py` - New module with all validation functions
- Integration into `eeg_processor.py` at specified lines
- No modification of existing filtering code
- Documentation of validation metrics

2 Technical Background

2.1 Why Data Validation Matters

In real-time BCI systems, bad data causes:

- False classifications (user thinks left, system reads right)
- Model training on garbage data
- Inconsistent performance across sessions

Your validation layer detects:

1. **Hardware failures** - disconnected electrodes, broken channels
2. **Signal anomalies** - flatlines, extreme noise, NaN values
3. **Data integrity** - duplicate channels, missing samples

2.2 Validation Happens in Two Places

- **Per-sample validation** - Fast checks on every new data chunk (Lines 151-152 in eeg_processor.py)
- **Buffer validation** - Comprehensive checks on accumulated data (Line 177 in eeg_processor.py)

3 Component 1: NaN and Inf Detection

3.1 What It Is

NaN (Not a Number) and Inf (Infinity) values appear when:

- Division by zero in calculations
- Hardware read errors
- Buffer overflow

3.2 Mathematical Definition

For data matrix $X \in R^{C \times T}$ where C = channels, T = time points:

$$\text{Valid}(X) = \begin{cases} \text{True} & \text{if } \forall i, j : X_{ij} \in R \text{ and } |X_{ij}| < \infty \\ \text{False} & \text{otherwise} \end{cases} \tag{1}$$

3.3 Implementation

Listing 1: Function: detect_nan_inf

```python
def detect_nan_inf(data):
    """
    Detect NaN or Inf values in EEG data.

    INPUT:
        data: np.ndarray, shape (channels, samples)
              Raw EEG data from board

    OUTPUT:
        is_valid: bool
                  True if no NaN/Inf found
        nan_channels: list
                      Indices of channels containing NaN/Inf

    EXAMPLE:
        >>> data = np.array([[1.0, 2.0, np.nan], [3.0, 4.0, 5.0]])
        >>> is_valid, bad_ch = detect_nan_inf(data)
        >>> print(is_valid)   # False
        >>> print(bad_ch)     # [0]
    """
    # YOUR IMPLEMENTATION HERE
    # Hint: Use np.isnan() and np.isinf()
    pass
```

Your Task:

1. Check each channel for NaN values using `np.isnan()`
2. Check each channel for Inf values using `np.isinf()`
3. Return list of bad channel indices
4. Set is_valid = False if any bad values found

4 Component 2: Flatline Detection

4.1 What It Is

A **flatline** occurs when a channel outputs constant values, indicating:

- Disconnected electrode
- Hardware malfunction
- Saturated amplifier

4.2 Detection Method

A channel is flatline if its standard deviation is below threshold:

$$\text{Flatline}(x) = \begin{cases} \text{True} & \text{if } \sigma(x) < \theta_{\text{flat}} \\ \text{False} & \text{otherwise} \end{cases} \tag{2}$$

where $\sigma(x)$ is standard deviation and $\theta_{\text{flat}} = 0.1\ \mu\text{V}$ (typical threshold).

4.3 Implementation

Listing 2: Function: detect_flatline

```
def detect_flatline(data, threshold=0.1):
    """
    Detect flatline channels (constant/near-constant signal).

    INPUT:
        data: np.ndarray, shape (channels, samples)
              EEG data segment
        threshold: float
                   Standard deviation threshold in microvolts
                   Default: 0.1 uV

    OUTPUT:
        flatline_channels: list
                           Indices of flatline channels
        channel_stds: np.ndarray
                      Standard deviation of each channel

    REASONING:
        Healthy EEG has std > 1 uV typically
        Flatline means electrode disconnected or broken
```

```
    """
    # YOUR IMPLEMENTATION HERE
    # Hint: Calculate std per channel, compare to threshold
    pass
```

5 Component 3: Extreme Noise Detection

5.1 What It Is

Extreme noise appears as:

- Muscle artifacts (EMG contamination)
- Movement artifacts
- Electrical interference spikes

5.2 Detection Method

Use z-score to detect outliers:

$$z_{ij} = \frac{X_{ij} - \mu_i}{\sigma_i} \tag{3}$$

Spike detected if $|z_{ij}| > \theta_z$ where $\theta_z = 5$ (typical threshold).

5.3 Implementation

Listing 3: Function: detect_extreme_noise

```
def detect_extreme_noise(data, z_threshold=5.0):
    """
    Detect extreme noise/spike artifacts.

    INPUT:
        data: np.ndarray, shape (channels, samples)
        z_threshold: float
                     Z-score threshold for spike detection
                     Default: 5.0 (5 standard deviations)

    OUTPUT:
        has_spikes: bool
                    True if spikes detected in any channel
        spike_channels: list
                        Channels containing spikes
        spike_percentage: float
                          Percentage of samples that are spikes

    DETECTION LOGIC:
        For each channel:
            1. Calculate mean and std
            2. Compute z-scores: z = (x - mean) / std
            3. Flag samples where |z| > threshold
            4. If > 1% of samples are spikes, mark channel bad
```

```python
    """
    # YOUR IMPLEMENTATION HERE
    pass
```

6 Component 4: Channel Duplication Check

6.1 What It Is

Sometimes hardware errors cause:

- Same channel duplicated in multiple positions
- Copy of previous buffer instead of new data

6.2 Detection Method

Use correlation between channels:

$$\rho(x_i, x_j) = \frac{\mathrm{cov}(x_i, x_j)}{\sigma_{x_i}\sigma_{x_j}} \tag{4}$$

If $\rho > 0.99$, channels are duplicates.

6.3 Implementation

Listing 4: Function: detect_channel_duplication

```python
def detect_channel_duplication(data, correlation_threshold=0.99):
    """
    Detect if channels are duplicates of each other.

    INPUT:
        data: np.ndarray, shape (channels, samples)
        correlation_threshold: float
                              Pearson correlation threshold
                              Default: 0.99

    OUTPUT:
        duplicate_pairs: list of tuples
                        [(ch_i, ch_j), ...] duplicate channel pairs
        correlation_matrix: np.ndarray, shape (channels, channels)
                           Full correlation matrix

    USAGE:
        If duplicates found, data is invalid - hardware issue
    """
    # YOUR IMPLEMENTATION HERE
    # Hint: Use np.corrcoef(data) to get correlation matrix
    pass
```

7 Component 5: Validation Package Class

7.1 What It Is

A **ValidationPackage** bundles data with its quality metrics.

Listing 5: Class: ValidationPackage

```
class ValidationPackage:
    """
    Container for validated EEG data with quality metrics.

    ATTRIBUTES:
        data: np.ndarray
              The validated EEG data
        is_valid: bool
                  Overall validity flag
        timestamp: float
                   Timestamp when data was collected
        quality_metrics: dict
                         Dictionary of quality measurements
                         {
                             'has_nan': bool,
                             'has_inf': bool,
                             'flatline_channels': list,
                             'noisy_channels': list,
                             'duplicate_pairs': list,
                             'sampling_rate': float
                         }

    USAGE:
        package = ValidationPackage(data, timestamp)
        if package.is_valid:
            # Safe to process
            process(package.data)
        else:
            # Log the issue
            log_error(package.quality_metrics)
    """
    def __init__(self, data, timestamp):
        self.data = data
        self.timestamp = timestamp
        self.is_valid = True
        self.quality_metrics = {}
```

8 Integration Points

8.1 File: eeg_processor.py

Line 1-10: Add import statement:

```
# Add after existing imports
from data_validation import (
    detect_nan_inf,
```

```
    detect_flatline,
    detect_extreme_noise,
    detect_channel_duplication,
    ValidationPackage
)
```

Line 151-152: Add per-sample validation in `get_recent_data()`:

```
# Current line 151:
eeg_data = data[self.eeg_channels, :]

# INSERT AFTER LINE 151:
# Validate new data chunk
is_valid, nan_channels = detect_nan_inf(eeg_data)
if not is_valid:
    print(f"Warning: NaN/Inf detected in channels {nan_channels}")
    # Skip this chunk or use last known good data
    return self.processed_data_buffer[:, -int(duration * self.sampling_rate):]
```

Line 177: Add buffer validation before returning:

```
# Current line 177 returns model_input
# INSERT BEFORE RETURN:

# Comprehensive validation on buffer
flatline_ch = detect_flatline(recent_data)
if len(flatline_ch) > 0:
    print(f"Warning: Flatline detected in channels {flatline_ch}")

has_spikes, spike_ch, spike_pct = detect_extreme_noise(recent_data)
if has_spikes and spike_pct > 5.0:
    print(f"Warning: Excessive noise ({spike_pct:.1f}%) in channels {spike_ch}")

# Then continue with existing return statement
```

9 Testing Your Implementation

9.1 Test Cases

Listing 6: Test Script - test_validation.py

```
import numpy as np
from data_validation import *

def test_nan_detection():
    """Test NaN detection"""
    # Create test data with NaN
    data = np.random.randn(8, 100)
    data[2, 50] = np.nan

    is_valid, bad_ch = detect_nan_inf(data)
    assert not is_valid, "Should detect NaN"
```

```python
    assert 2 in bad_ch, "Should identify channel 2"
    print("    NaN detection works")

def test_flatline():
    """Test flatline detection"""
    data = np.random.randn(8, 100)
    data[5, :] = 0.0  # Flatline channel 5

    flat_ch, stds = detect_flatline(data)
    assert 5 in flat_ch, "Should detect flatline in channel 5"
    print("    Flatline detection works")

def test_noise():
    """Test noise detection"""
    data = np.random.randn(8, 100)
    data[3, 10:20] = 50.0  # Large spike

    has_spike, spike_ch, pct = detect_extreme_noise(data)
    assert has_spike, "Should detect spike"
    print("    Noise detection works")

if __name__ == "__main__":
    test_nan_detection()
    test_flatline()
    test_noise()
    print("\nAll tests passed!")
```

10 Expected Outcomes

10.1 Performance Metrics

- **Validation time:** $<$ 5ms per chunk
- **False positive rate:** $<$ 1%
- **True positive rate:** $>$ 95% on simulated bad data

10.2 Documentation Required

1. Comments explaining each validation function
2. Test results showing validation catches errors
3. Performance benchmarks (timing measurements)
4. Integration checklist confirming all insertion points completed

11 Grading Rubric

Component	Points	Criteria
NaN/Inf Detection	15	Correctly identifies bad values, returns channel indices
Flatline Detection	15	Uses std threshold, identifies constant channels
Noise Detection	20	Implements z-score method, reports percentage
Duplication Check	15	Calculates correlations, identifies duplicates
Integration	20	Code properly inserted at specified lines
Testing	10	Test script runs and passes all tests
Documentation	5	Clear comments and docstrings
Total	**100**	

12 Timeline

- **Week 1-2:** Implement validation functions
- **Week 3:** Integration into eeg_processor.py
- **Week 4:** Testing with synthetic bad data
- **Week 5:** Hardware testing with real OpenBCI board
- **Week 6:** Documentation and final report

13 Resources

- NumPy documentation: `https://numpy.org/doc/`
- EEG artifact detection paper: Delorme et al. (2007) - EEGLAB artifact rejection
- BrainFlow API: `https://brainflow.readthedocs.io/`