Department of Computer Science

Submitted in part fulfilment for the degree of MSc

Project reports

Fan Wu

22 August 2024

Supervisor: Christopher Crispin-Bailey

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Christopher Crispin-Bailey, for his unwavering support, guidance, and encouragement throughout the duration of this project. His expertise and insightful feedback were instrumental in shaping the direction and quality of my research.

I extend my sincere thanks to the organizations and individuals who collected and shared the open datasets that were crucial for my research. Your commitment to open data has significantly contributed to the advancement of this project.

On a personal note, I would like to thank my family and friends for their constant support and understanding throughout this journey. Their encouragement and patience have been my strength during challenging times.

Thank you all.

STATEMENT OF ETHICS

This research was conducted in accordance with the ethical guidelines and principles established by University of York, Department of Computer Science.

All datasets used in this study were obtained from publicly accessible open data sources, which are freely available for research and analysis. The data utilized are obtained from PhysioNet, a repository of freely-available medical research data, managed by the Massachusetts Institute of Technology Laboratory for Computational Physiology. These datasets are available to the public and were accessed in compliance with the terms and conditions stipulated by the data providers.

No new data was collected from human participants, and therefore, no ethical approval from an ethics committee or institutional review board (IRB) was required. The research did not involve any interaction with individuals or the collection of personal data beyond what is publicly available in the datasets.

The datasets were anonymized and any sensitive information was removed or de-identified prior to analysis to ensure privacy and confidentiality. The use of open data ensures that all data sources were appropriately credited, and the research adhered to all relevant data usage policies and licensing agreements.

In terms of data integrity, every effort was made to ensure the accuracy, reliability, and transparency of the data analysis and reporting process. The research was conducted with the utmost attention to maintaining the highest standards of academic integrity and rigor.

No conflicts of interest were identified in the course of this research. Any potential conflicts that may have arisen were managed according to the ethical guidelines of University of York.

The ethical implications of this research were considered with a focus on contributing valuable knowledge to the academic community and society. The study was designed to ensure that the benefits of the research outweigh any potential risks, and the findings are shared responsibly and ethically.

By adhering to these ethical principles and guidelines, this research aims to maintain the highest standards of academic integrity and contribute positively to the field.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Executive summary

This project aims at finding the best parameter set(s) and data preprocessing method of Log2 Sub-band (L2SB) algorithm. The L2SB algorithm can be used in wireless EEG recorders and other biomedical signal recorders. This algorithm deals with 12-bit binary data, achieves effective compression results while minimizing hardware resource consumption. Thus, this project can help choose the best parameter set and preprocess the data.

To find the best data preprocessing method, several common preprocessing methods, such as scaling and adding offset were explored and the compression ratio (CR) was used as the criteria. An iterative method was used to find the best parameter set(s). To give a better understanding of results, tables, line plots, box plots and raincloud plots were used to analyse the data. The compression result of the L2SB algorithm is compared with that of the Differential Pulse-Code Modulation (DPCM) combined with Huffman coding algorithm.

The results can be concluded as follows:

1. For integer type raw data, adding offset is the best preprocessing method and using the absolute value of the minimum as offset can lead to good CR. For float type raw data, choosing scaling factor according to the minimum and maximum can lead to the highest CR. Overall, making all data points non-negative while minimizing their range can lead to good results.
2. If the goal is to obtain the highest CR, then partitioning into three nibbles can achieve this in most datasets, but only a small number of combinations can outperform those when the nibble number is seven, and only by a small margin. However, no single combination can achieve the highest CR in every dataset. When compressing mixed EEG data, the combination (5, 3, 4) achieves the highest CR. When compressing unmixed EEG data, the combination (4, 3, 5) achieves relatively good, but not top, results.
3. If the goal is to obtain the highest average CR, then partitioning into seven nibbles can guarantee this across all datasets. Choosing a nibble number of seven can guarantee the algorithm consistently delivers a relatively high CR for a group of cases.
4. DPCM combined with Huffman coding obtains higher CR than the L2SB algorithm across all datasets. However, it requires storing and transmitting Huffman codebook, thus may increase the energy consumption. It remains uncertain which algorithm would be more efficient in practical applications.

After a thorough analysis, it is affirmed that this project does not present any significant legal, social, ethical, professional, or commercial issues.

All data was collected with proper consent and in compliance with relevant regulations, ensuring patient privacy and data security.

The algorithm enhances the accessibility and effectiveness of wearable medical devices without adversely affecting any social groups.

Ethical guidelines for the use of medical data are strictly followed, respecting patient rights and autonomy.

The work adheres to established standards and best practices in data science and medical research, maintaining high levels of academic and professional integrity.

Additionally, an analysis of market potential and regulatory landscape indicates no significant commercial barriers or risks associated with the L2SB algorithm's development.

1 Introduction

Electroencephalography (EEG) is an electrophysiological technique to record the electrical activity in the human brain[1]. It has applications in various medical fields, including anesthesia depth monitoring, seizure[1], Parkinson's disease[2], epilepsy[3], evoked potentials (EPs) and event-related potentials (ERPs) [1].

The traditional method of collecting biomedical data often poses inconvenience to patients or research subjects, as it typically limits their mobility. A wearable wireless signal recorder addresses this issue by enabling users to move freely. This technology enhances real-time healthcare delivery from caregivers to patients with neurological or other medical conditions.

However, designing a wireless biomedical signal recorder presents several challenges, with power consumption being a critical concern. For an EEG recorder to provide a better user experience, it needs to be compact, which reduces the space available for a battery. Meeting the requirements for long-term recording with a small battery is often difficult. In wireless EEG monitors, the transceiver is a major contributor to power consumption, so reducing the size of transmitted data could significantly address this power issue. Additionally, reducing data size can help save power when using onboard storage media, such as flash memory, by lowering the read/write rates. However, implementing a data reduction unit also consumes extra power, leading to trade-offs that must be carefully considered.

Log2 Sub-band (L2SB) is a data compression algorithm specifically designed for wireless EEG recorders and other biomedical signal recorders[4]. This algorithm achieves effective compression results while minimizing hardware resource consumption and inherently offers potential for seizure detection. This project will explore data preprocessing methods and parameter combinations of L2SB algorithm to determine the best preprocessing method and set(s) of parameters for a given type of biosignal from a wearable sensor. The compression outcomes will be compared to a traditional approach such as Differential Pulse-Code Modulation (DPCM) combined with Huffman encoding.

2 Background

2.1 EEG data compression technologies

EEG data compression is a critical task due to the high volume of data generated in continuous EEG recordings. Various techniques have been developed and can be broadly categorized into four categories: Huffman coding and its related methods, predictive compression techniques, transformation compression techniques and methods exploiting EEG-specific properties[5].

1. Huffman coding and its related methods

This category mainly includes traditional Huffman coding[6], collapsed Huffman tree (CHT) and repetition count compression.

Huffman coding is an entropy-based compression method that provides optimal prefix codes through a greedy algorithm exploiting a binary tree, known as a Huffman tree. It can efficiently approximate the entropy of the symbols and is widely used due to its simplicity. But it cannot control the maximum length of the code words and is not always optimal for data with small variations in symbol probabilities.

To address this, CHT was proposed. It can limit the maximum bit string length of the code-words by collapsing tree leaves. This approach achieves a compression ratio (CR) that approximates the entropy of the symbols, making it efficient for data with varying symbol probabilities.

When EEG data contains long sequences of repeated symbols, repetition count compression[7] can be used. It is a run-length compression variant that encodes symbols along with their repetition counts. After repetition count compression, Huffman coding can be used to further compress the data.

2. Predictive Compression Techniques

Predictive compression reduces entropy by predicting signal values based on past samples and encoding the prediction error. This category mainly contains Markov predictor[8], digital filtering predictor[8], [9], linear predictive coding[10], adaptive linear prediction[11] and artificial neural networks (ANNs) predictor[12], [13].

Markov Predictor assumes the signal has Markovian properties and uses a first-order Markov chain to model and predict the signal. The predictor utilizes a frequency matrix derived from training data to estimate conditional probabilities. It is efficient for signals exhibiting Markovian properties, but the performance is heavily dependent on the

accuracy of the frequency matrix and can struggle with non-Markovian data.

Digital filtering predictor (LP) uses a digital linear predictive filter designed to minimize the mean squared error. It is effective for signals with linear relationships. It assumes that the mathematical description of data is known and the data are generated via a stationary random process, while in reality these conditions are rarely available.

To address these limitations, linear predictive coding (LPC) is proposed. It assumes that the signal can be considered stationary within short frames. For each frame, coefficients are computed using LP and are used to encode the original signal.

Adaptive linear prediction is similar to LPC but adapts the prediction model parameters in real-time to better match the signal's characteristics. It’s more robust to changing signal properties and non-stationary data, but has increased computational complexity due to the need for continuous parameter updates.

Artificial neural network (ANN) predictor uses multilayer perceptron architecture and is trained with past samples to predict future values. It can capture complex non-linear relationships in the data, but it requires large amounts of training data and the training process can be computationally intensive and time-consuming.

3. Transformation Compression Techniques

Transformation-based techniques represent the signal in a different domain to achieve compression. This category includes Karhunen–Loeve transformation (KLT)[14] and discrete cosine transform (DCT)[11].

KLT is the optimum method for signal representation in terms of minimizing mean square error. It uses a transformed vector and a transformation matrix to apply an orthogonal transformation of original data. It is a lossless data compression technique, but its limitation is the significant computation time required for its implementation.

DCT is a suboptimal solution which has better performance in running time. It transforms the signal into the frequency domain, retaining only the most significant components to reduce data size. Unlike KLT, it is a lossy technique. It can be used in EEG and electrocardiogram (ECG) data compression and has frequently been reported that DCT achieves performance close to KLT.

4. Methods exploiting EEG properties

EEG data have a temporal correlation and a spatial correlation that can be exploited to design compression techniques. This category mainly includes vector quantization[8] and predictive compression with delayed inputs[15].

Vector quantization maps vectors of EEG samples to code vectors, encoding the error vectors separately. It exploits the spatial correlation of EEG data from multiple channels.

Predictive compression with delayed inputs includes delayed inputs to capture long-term dependencies in the signal. It takes advantage of the time correlation in EEG data, particularly around 10.6 Hz corresponding to alpha waves.

2.2 Comparison between L2SB algorithm and traditional algorithm

To evaluate the effectiveness of any compression method and its implementation, three key factors must be considered[16]. The first factor is the CR, as it directly impacts the reduction in data that needs to be transmitted or stored. The second factor is the power consumption involved in achieving this data reduction, which is crucial for energy efficiency. The third consideration is the circuit area, as large circuits may pose challenges when integrating the compression method into an SOC or FPGA design.

A useful benchmark for comparative performance analysis is Huffman coding, as it is well-established and widely adopted by researchers for evaluating their own compression algorithms. Huffman serves as a reliable reference standard, enabling novel techniques to be assessed, with Huffman performance providing a common point of comparison. When dealing with bio-signals, DPCM is often used as a pre-processing method before Huffman coding is applied.

When comparing the CR, the L2SB algorithm offers a CR of around 1.65 when applied to bio-signal data such as EEG, while Huffman coding achieves a higher compression ratio than L2SB, typically around 2.2 for EEG data. This is due to its ability to represent frequent data with shorter codes and less frequent data with longer codes.

Although the Huffman algorithm provides a higher CR, L2SB's power consumption per bit is significantly lower, especially in power-limited systems. In an 8-channel system, L2SB consumes 27 times less power than Huffman and 192 times less power in a single-channel system.

When comparing the area efficiency, L2SB achieves far greater area efficiency compared to Huffman encoding. For example, in terms of

compression per square micrometre (um²) of silicon, L2SB delivers superior data compression compared to other known lossless compression schemes. It offers a better balance between CR and silicon area utilization, making it suitable for implementation in compact, low-power hardware.

In conclusion, although L2SB delivers a more modest compression ratio compared to other algorithms, this simplicity comes with significant advantages. More complex algorithms typically result in higher power consumption, greater circuit complexity, and larger gate area requirements, which often limit their effectiveness in resource-constrained systems. In contrast, L2SB demonstrates a clear advantage when it comes to the trade-off between compression and transmission power, especially when factoring in its minimal silicon area footprint. For instance, even though L2SB's compression ratio may be lower, its ability to significantly reduce power consumption makes it particularly suited for applications where energy efficiency is paramount. This balance becomes even more critical in extreme resource-limited environments, such as those utilizing printed organic thin film semiconductors, where more complex alternatives may be impractical or unviable.

3 Methodology

L2SB algorithm is a technique that works on time-domain. Biomedical signals are often transferred into 8-bit to 12-bit samples on most recorders. Without loss of generality, the L2SB algorithm focuses on 12-bit binary data. Original L2SB algorithm separates 12 bits into 3 nibbles, each nibble has 4 bits. When compressing data, nibbles in each sample will be compared with the same part in its previous sample. Different nibbles will be kept and sent to the transceiver or the memory. A header is used to indicate how many nibbles are left after the comparison, as Figure 1 shows. The three nibbles can be called nibble 1, 2 and 3. If all nibbles are the same, no nibble is kept and the header is 00. If nibble 1 and 2 are the same, nibble 3 is kept and the header is 01. If only nibble 1 is the same, nibble 2 and 3 are kept and the header is 10. If all nibbles are different from the previous sample, all three nibbles are kept and the header is 11.

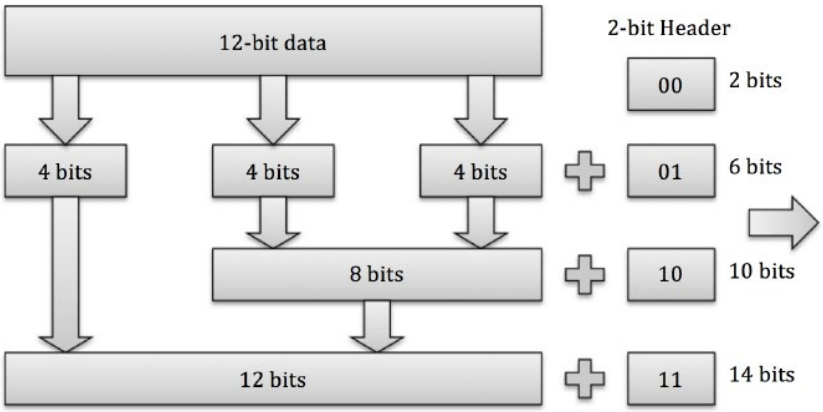

Figure 1 Log2 Sub-band Algorithm

3.1 Explore parameter combinations

In this project, different ways of separating 12-bit data were explored. A 12-bit binary data can be separated into 2 to 12 nibbles. Different length of headers is required for these separations, as indicated in Table 1.

Table 1 Header Length

Number of nibbles	Header length
2, 3	2
4, 5, 6, 7	3
8, 9, 10, 11, 12	4

When evaluating data compression efficiency, the impact of nibble number on the CR must be considered. On the one hand, increasing nibble number means the data can be compared in finer granularity, which may lead to a possible higher CR. On the other hand, more nibbles mean longer header, which leads to lower CR. Intuitively, a nibble number of three or four, which is neither too small nor too large, might result in the highest CR.

For most separations, there are dozens to hundreds of combinations. An iterative approach was used to calculate CR using these combinations in the selected dataset. The results are shown in the next chapter.

To compare the CR of L2SB algorithm and traditional data compression algorithm, Differential Pulse-Code Modulation (DPCM) and Huffman coding is chosen. The scheme of the L2SB algorithm resembles DPCM. DPCM calculates the difference of current data point and previous data point, then Huffman coding encodes this data.

3.2 Explore data preprocessing method

The EEG signals used in this project are sourced from two distinct researches: the study on non-linear deterministic patterns of brain electrical signals conducted at the University of Bonn [17] and the research on seizure detection carried out at the Massachusetts Institute of Technology(MIT)[18].

The dataset from the University of Bonn comprises EEG signals categorized into different groups, each containing one hundred data segments with a duration of 23.6 seconds. These segments were extracted from continuous multichannel EEG recordings after undergoing visual inspection and artifact removal, including EOG artifacts. For the purposes of this project, three groups from this source were selected: signals from five healthy individuals with eyes closed (dataset O), signals from five epilepsy patients during seizure-free periods (dataset F), and seizure signals (dataset S). Dataset F, O and S each contain 100 files, and each file contains 4097 data points.

This dataset consists of positive and negative integers in the range [-1885, 2047], which means all original data points can be represented by a 12-bit binary number. Since the L2SB algorithm distinguishes the difference between each nibble, in order to eliminate the influence of positive and negative signs, converting these data into non-negative data first before they are transferred into binary form might lead to better CR. An offset is needed to be added to the data. The minimum of the offset is the absolute value of the minimum of the data, and the maximum is 2048. Within this range, a sample of 30 is chosen to investigate the influence of adding different offset on the CR. As long

as the data is all non-negative, moving it up or down along the y-axis shouldn't have a significant impact on the CR.

The EEG data from MIT were collected from 24 subjects. The signals are mostly segmented into numerous uncategorized one-hour recordings. For this project, two one-hour segments (mixed signal 1 and mixed signal 2) from a single subject were chosen. The term "mixed" means the signal contains both non-seizure and seizure events. This dataset contains 48 files, and each file contains two segments of 3600 data points.

This dataset consists of 3-digit floats in the range [-3.28, 2.51]. To represent them in 12-bit binary form, the data need to be transformed into integers. There are two ways of transformation. The first method calculates the division of 2047 and maximum value and the division of -2048 and minimum value, then chooses one as the scaling factor. The data is multiplied by this scaling factor, rounded to integers and added by an offset. The second method adds the data by an offset and obtains non-negative data first, then scales it to the range [0, 4095]. These two methods both lead to non-negative range, although the range obtained by the first method is smaller than that of the second method. With the same original data points, a larger range means the difference between consecutive pairs becomes larger, and might lead to lower CR.

4 Results and Analysis

4.1 L2SB Algorithm implementation

This project uses Python in Google Colab to implement the L2SB algorithm, as shown in Appendix A. The code reads input data and stores them into a data frame. Then several functions were defined to implement the L2SB algorithm, as shown in Table 2.

Table 2 Function Name and Explanation

Function Name	Explanation
compare(str1, str2, split)	Compare input string str1 and str2 using a list called split. Split is a list consisting of several integers that sum up to 12. Return the header.
l2sb(input, split)	Implement the L2SB algorithm to compress string input by using the split list. Return a list of compressed results.
total_length(strings)	Return the total length of all strings in the input list.
calRatio_all(input, split)	Use the L2SB algorithm to compress input data, return the overall compression ratio.
Find_combinations(target_sum, nums_parts, current_combination, current_sum)	Return all possible combinations of num_parts integers that sum up to target_sum.

In this project, the CR is calculated as:

$$CR = original\ size / compressed\ size$$

4.2 Data preprocessing method analysis

For the dataset from the University of Bonn, L2SB algorithm with a separation of (4,4,4) nibbles was applied to the original data and to the data with different offsets added.

The results of adding different offsets are shown in Figure 2, Figure 3, Figure 4. From the results, several findings can be concluded:

1. As long as the offset is large enough to guarantee all data points are non-negative, increasing it will not have a significant impact on the CR, the fluctuation is 2.3%, 3.1% and 0.12% respectively for dataset F, O and S.

2. The effect of adding an offset on non-seizure dataset (dataset F and O) is clearly periodic, but there doesn't exist one offset that maximizes the CR in every dataset.
3. Using the absolute value of the minimum value as the offset appears to be a good choice, since it only brings a 0.03%, 0.4% and 0.07% decrease compared to the highest CR on the three datasets.

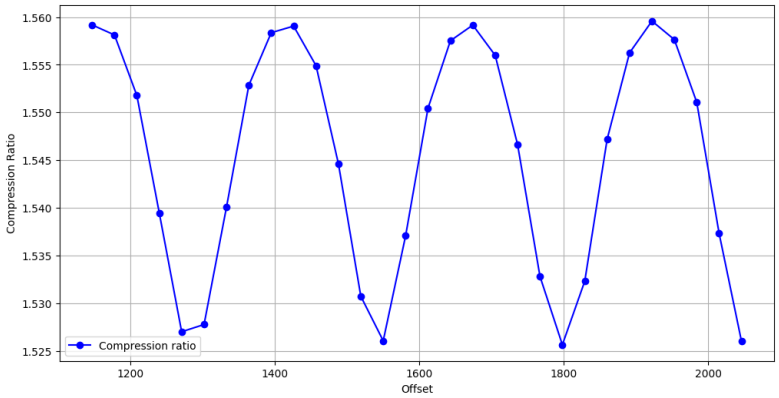

Figure 2 Compression Ratio under Different Offsets of Dataset F

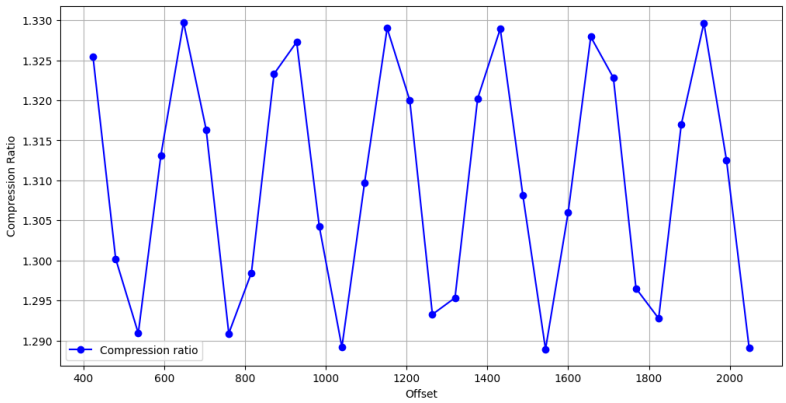

Figure 3 Compression Ratio under Different Offsets of Dataset O

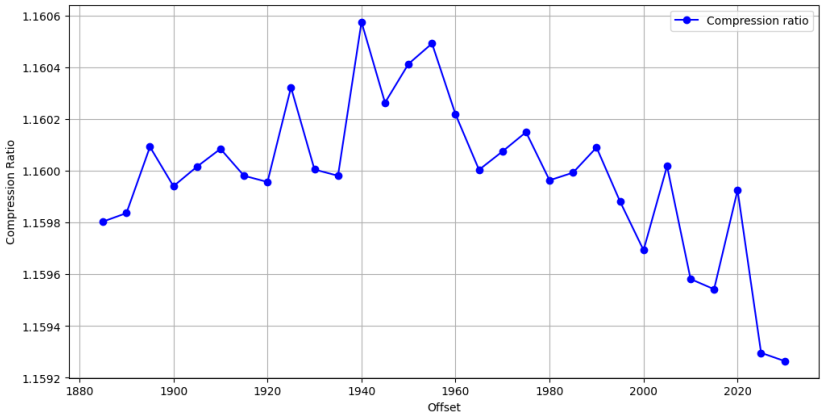

Figure 4 Compression Ratio under Different Offsets of Dataset S

The CR obtained from adding the absolute value of minimum as offset and adding no offset was compared in Table 3. The results showed that by guaranteeing all data points non-negative, the CR was better than that of using the original data.

Table 3 No Offset vs Min as Offset

Dataset	No offset	Min as offset	Improvement
F	1.527	1.559	2.09%
O	1.289	1.325	2.79%
S	1.159	1.16	0.09%

For the dataset for the MIT, two ways of scaling were studied. The first one calculates the division of 2047 and maximum value and the division of -2048 and minimum value, which is 624 and 815 respectively. For simplicity, a scaling factor of 600 was chosen. The second method adds the data by its absolute value of its minimum and obtains non-negative data. The data is then scaled into range [0,4095]. The CR results are shown in Table 4. It shows using 600 as scaling factor leads to a higher CR.

Table 4 Compression Ratio of Two Methods

Data source	Method 1	Method 2
Mixed signal 1	1.584	1.549
Mixed signal 2	1.573	1.53

In the following analysis, the first method is chosen for scaling the MIT dataset and adding the absolute value of the minimum value is chosen for scaling the Bonn dataset.

4.3 Parameter combination analysis

After preprocessing the dataset as mentioned above, L2SB algorithm with different nibble numbers was applied to the five datasets. For a particular nibble number, there exists many combinations. Figure 5 shows the highest CR for each nibble number.

When the header length remains the same (as shown in Table 1), increasing the number of nibbles can increase the CR. This trend is obvious in range two to three and four to seven, and subtle in range eight to twelve.

Among the five datasets, four groups have the highest CR when they are divided into three nibbles. The only exception is dataset F. In

dataset F, the highest CR was obtained when it was divided into seven nibbles. However, the CR of this group when divided into three nibbles is 1.59, and the highest is 1.61, which decreases only by 1.2%. Thus, to achieve the highest CR in most datasets, a nibble number of three is optimal, and the second-best choice is seven.

For two mixed signals, the highest CR is similar in all nibble numbers, while in the dataset F, O, and S, the values of the highest CR varies a lot in all nibble numbers, which agrees with the previous findings[4].

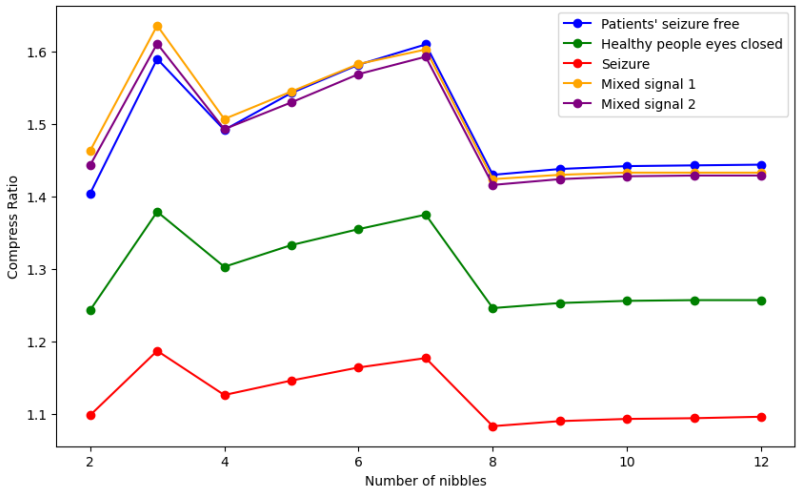

Figure 5 Highest Compression Ratio vs Number of Nibbles

To investigate all data points, a raincloud diagram was plotted for each dataset, as shown in Figure 6 to Figure 10. Across all datasets, the highest average CR is achieved when the nibble number is seven. Moreover, the data is distributed more densely and concentrated in the high CR range for a nibble number of seven, proving its efficiency.

Thus, choosing three as the nibble number can lead to the highest CR in most datasets, but only a small number of combinations can outperform those when the nibble number is seven, and only by a small margin. For example, in dataset O, a separation of (4,3,5) led to a CR of 1.379, while a separation of (4,1,1,1,1,1,3) led to a CR of 1.375, which improves by only 0.29%.

When the highest average CR is the objective, choosing a nibble number of seven can achieve this goal across all datasets. Furthermore, considering the distribution of all possible choices within one nibble number, seven proves to be an excellent choice since the data points are heavily concentrated in the high CR area. Choosing a nibble number of seven can guarantee the algorithm consistently delivers a relatively high CR for a group of cases.

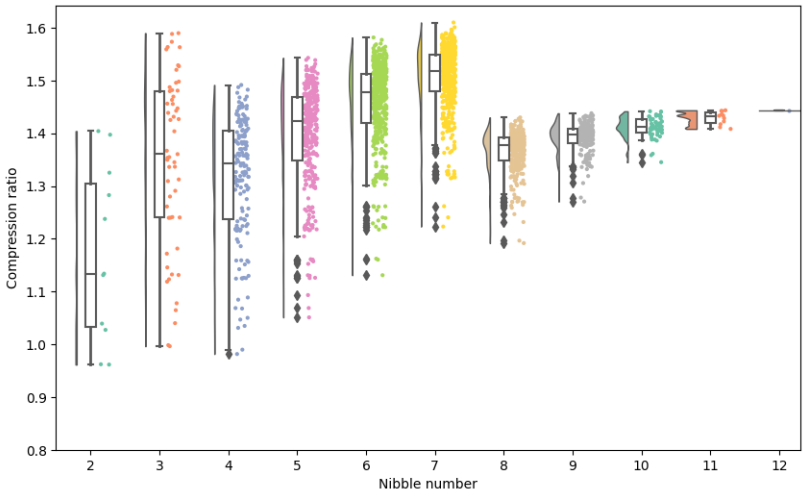

Figure 6 Compression Ratio vs Nibble Number of Dataset F

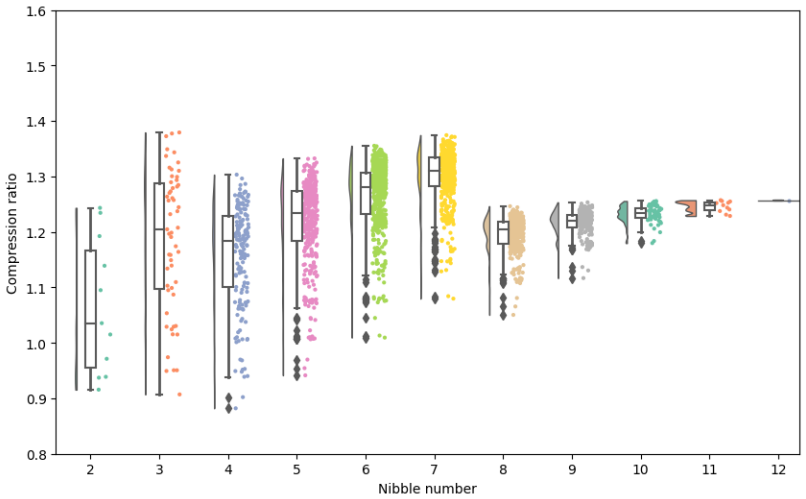

Figure 7 Compression Ratio vs Nibble Number of Dataset O

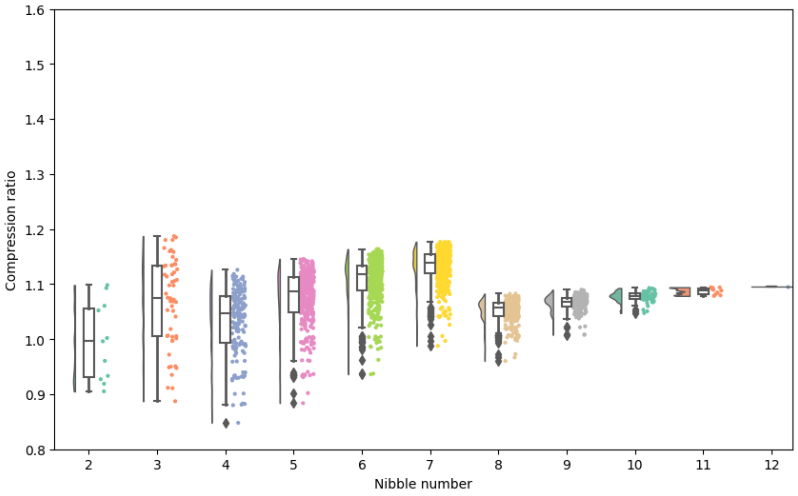

Figure 8 Compression Ratio vs Nibble Number of Dataset S

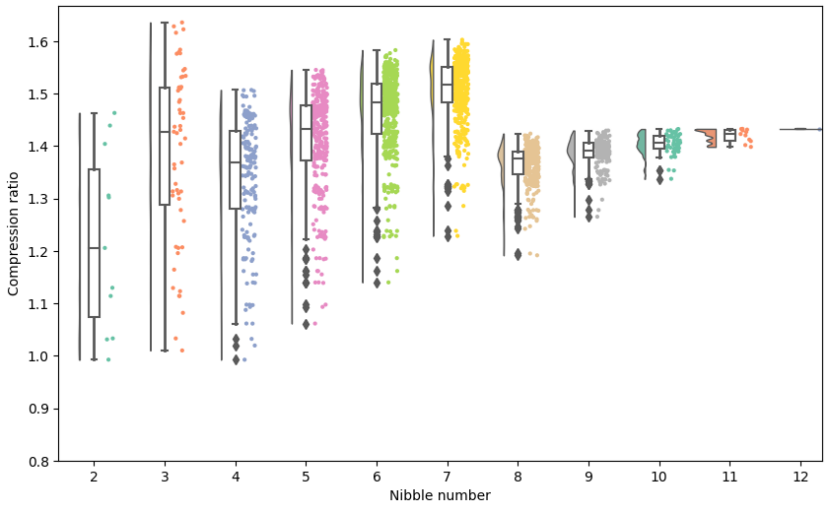

Figure 9 Compression Ratio vs Nibble Number of Mixed Signal 1

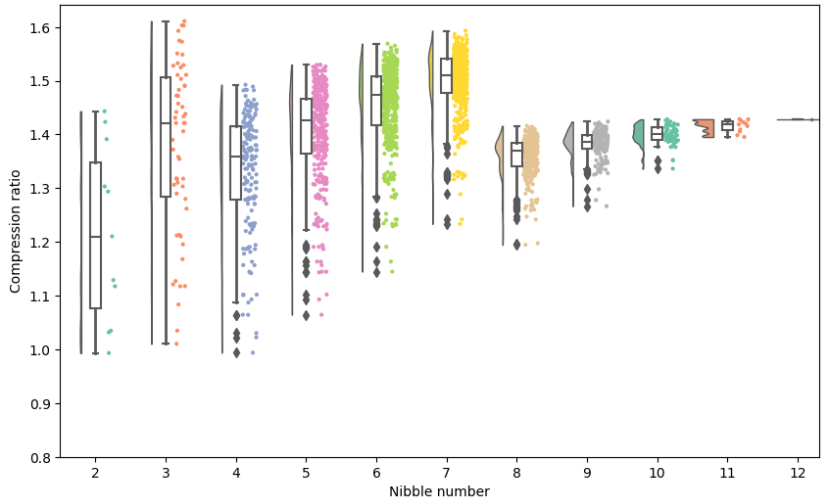

Figure 10 Compression Ratio vs Nibble Number of Mixed Signal 2

Since dividing the data into three nibbles can result in the highest CR in most datasets, further investigation was made. When the nibble number is three, no single combination consistently results in the highest compression ratio across all data sets, as shown in Table 5. However, certain patterns do emerge.

For mixed signal datasets, the combination (5,3,4) consistently yields the highest CR, making it the preferred choice for these types of data. This combination achieves the top CR for both Mixed signal 1 (CR1: 1.636) and Mixed signal 2 (CR1: 1.611), indicating its robust performance in scenarios with mixed signals.

For datasets with non-mixed data, there is no single combination that achieves the highest CR across all datasets. However, the combination (4,3,5) consistently ranks in the top three for these

datasets, demonstrating its relative effectiveness. For example, in dataset O, (4,3,5) achieves a CR1 of 1.379, and in dataset S, it achieves a CR3 of 1.180. This pattern suggests that (4,3,5) is a reliable choice for non-mixed data, offering consistently high CRs.

In summary, if a fixed separation is required, the combination (5, 3, 4) should be prioritized for mixed data due to its superior performance, while the combination (4, 3, 5) is a strong candidate for non-mixed data, providing consistently high CRs across various datasets.

Table 5 CR of Different Datasets

Dataset	Split1	CR1	Split2	CR2	Split3	CR3
F	6,2,4	1.590	5,3,4	1.588	4,3,5	1.573
O	4,3,5	1.379	4,2,6	1.378	5,2,5	1.373
S	3,3,6	1.187	4,2,6	1.185	4,3,5	1.180
Mixed signal 1	5,3,4	1.636	5,2,5	1.628	6,2,4	1.623
Mixed signal 2	5,3,4	1.611	5,2,5	1.603	4,3,5	1.603

4.4 L2SB algorithm vs traditional algorithm

Comparing the results of L2SB algorithm and DPCM and Huffman coding, the traditional algorithm consistently outperforms L2SB across all datasets, as shown in Table 6 and Figure 11. This aligns with the previous findings[16].

For non-mixed data, the CR achieved by the L2SB algorithm is approximately 75% to 85% of that achieved by the traditional method. However, for mixed data, this ratio drops to around 60%. Despite the lower CR of the L2SB algorithm compared to the traditional approach, L2SB has the advantage of not requiring a Huffman codebook to map the corresponding relationships.

Considering the energy consumption involved in storing and transmitting the Huffman codebook, it remains uncertain which algorithm would be more efficient in practical applications. This factor could potentially offset the compression advantages of the traditional algorithm, making it challenging to determine the overall best-performing method in real-world scenarios.

Table 6 Compression Ratio Results

Dataset	DPCM + Huffman	L2SB	L2SB to DPCM+Huffman Ratio (%)
F	2.130	1.590	75
O	1.778	1.379	78
S	1.398	1.187	85
Mixed signal 1	2.819	1.636	58
Mixed signal 2	2.697	1.611	60

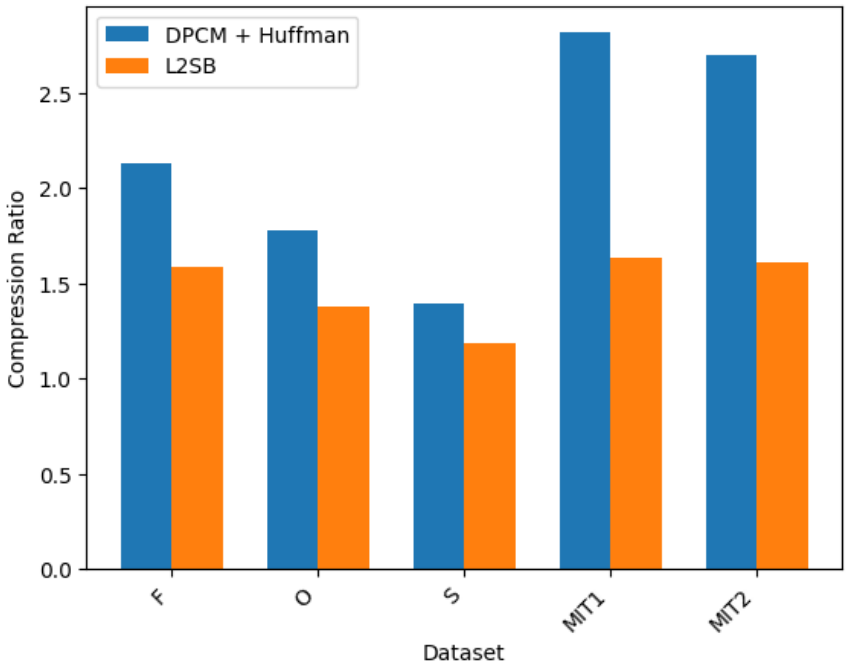

Figure 11 Compression Ratio Comparison

5 Conclusion

The L2SB algorithm separates 12-bit data into several nibbles and then compresses it. This project first explores different data preprocessing methods for integer and float type data. For integer type data within the range [-2048, 2047], simply using its absolute value of the minimum as offset leads to relatively good results. When all data points are non-negative, increasing the offset has a minimal effect on the results. For float type data, choosing a scaling factor according to its minimum and maximum values and then adding an offset to make all data points non-negative leads to better results. Overall, making all data points non-negative while minimizing their range can lead to good results; conversely, a larger range has a negative effect on the outcome. This conclusion aligns with expectations.

The strategy for choosing the nibble number depends on the algorithm's emphasis. If the goal is to obtain the highest CR, then partitioning into three nibbles can achieve this in most datasets. However, no single combination can achieve the highest CR in every dataset. When compressing mixed EEG data, the combination (5, 3, 4) achieves the highest CR, whereas for unmixed EEG data, the combination (4, 3, 5) achieves relatively good, but not top, results. If the goal is to obtain the highest average CR across all datasets, partitioning into seven nibbles can guarantee this across all datasets. Choosing a nibble number of seven can guarantee the algorithm consistently delivers a relatively high CR for a group of cases. This conclusion is unexpected since seven nibbles seem redundant and a low CR was expected compared to three or four nibbles.

When compared to traditional algorithms like DPCM combined with Huffman coding, the L2SB algorithm consistently underperforms across all datasets, indicating that the L2SB algorithm still requires further refinement. However, it's important to consider the trade-off between energy consumption and overall efficiency. While traditional algorithms achieve higher compression ratios, they also require storing and transmitting a Huffman codebook, which could increase energy usage. This trade-off introduces uncertainty regarding which algorithm is truly more efficient in practical applications. Therefore, future work should continue to explore and refine this balance to optimize the algorithm's applicability across diverse real-world scenarios.

There are several promising directions for future research.

i. The L2SB algorithm shows significant variability in CR across the non-mixed datasets F, O, and S. This variation may be attributed to differences in data features, and identifying the specific features

responsible for these discrepancies is a valuable avenue for further investigation.

ii. Although this project primarily focused on EEG data, other types of datasets, such as ECG data, may exhibit distinct characteristics that necessitate alternative strategies for data preprocessing and nibble selection. Exploring how these strategies should be adapted for various types of data could greatly enhance the L2SB algorithm's versatility and effectiveness.

iii. The nibble header used in this project was fixed for a specific nibble number. However, nibble headers can be of variable lengths, which presents another area for future exploration. For instance, in this project, a nibble number of five was represented by the headers 000, 001, 010, 011, and 100. Alternative approaches could involve replacing 100 with 1, or using 00, 01, 10, 110, and 111 as headers. There are numerous variations possible, and examining how assigning the shortest nibble header to the most frequently used nibble might optimize performance could be an intriguing topic for future research.

Appendix A

This appendix shows the code when dividing 12-bit data into 7 nibbles. Other divisions have the similar code.

Copy of 7 nibbles

August 5, 2024

1 Bonn data

[ ]:
```
import os
import pandas as pd
import numpy as np
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
os.chdir('/content/drive/My Drive/0-Project')
folders = ['F', 'O', 'S']
# O: healthy people with eyes closed; F: patients seizure free; S: seizure
dataframes = {}

for folder_path in folders:
  file_list = os.listdir(folder_path)
  data_dict = {}

  for file_name in file_list:
    column_name = file_name[1:4]
    file_path = os.path.join(folder_path, file_name)
    with open(file_path, 'r') as file:
      file_data = [line.strip() for line in file.readlines()]
      data_dict[column_name] = file_data

  df = pd.DataFrame(data_dict)
  df = df[sorted(df.columns)]

  # Store the DataFrame in the dictionary with the folder name as the key
  dataframes[folder_path] = df

# Accessing the DataFrames for F, O, and S
df_F = dataframes['F']
df_O = dataframes['O']
df_S = dataframes['S']
df_F = df_F.apply(pd.to_numeric)
df_O = df_O.apply(pd.to_numeric)
```

```
df_S = df_S.apply(pd.to_numeric)
```

2 Representation

1. All nibbles are the same, 000
2. Nibble 1 to 6 are the same, nibble 7 is different, 001
3. Nibble 1 to 5 are the same, nibble 6 to 7 are different, 010
4. Nibble 1 to 4 are the same, nibble 5 to 7 are different, 011
5. Nibble 1 to 3 are the same, nibble 4 to 7 are different, 100
6. Nibble 1 to 2 are the same, nibble 3 to 7 are different, 101
7. Nibble 1 is the same, nibble 2 to 7 are different, 110
8. All nibble are different, 111

[ ]:
```
def compare(str1, str2, split):
  '''
  Params:
    str1  : input string 1
    str2  : input string 2
    split : a list of 7 integers that sum up to 12

  Compare str1 and str2, both are 12-bit binary representation, separate them
  into 7 nibbles using split
  All nibbles are the same, 000
  Nibble 1 to 6 are the same, nibble 7 is different, 001
  Nibble 1 to 5 are the same, nibble 6 to 7 are different, 010
  Nibble 1 to 4 are the same, nibble 5 to 7 are different, 011
  Nibble 1 to 3 are the same, nibble 4 to 7 are different, 100
  Nibble 1 to 2 are the same, nibble 3 to 7 are different, 101
  Nibble 1 is the same, nibble 2 to 7 are different, 110
  All nibble are different, 111
  '''
  # Ensure both strings are 12-bit binary representations
  if len(str1) != 12 or len(str2) != 12:
    raise ValueError("Both strings must be 12-bit binary representations.")

  # Split the strings into 7 nibbles
  nibbles1 = [str1[:split[0]], str1[split[0]:(split[0]+split[1])],
              str1[(split[0]+split[1]):(split[0]+split[1]+split[2])],
              str1[(split[0]+split[1]+split[2]):
↪(split[0]+split[1]+split[2]+split[3])],
              str1[(split[0]+split[1]+split[2]+split[3]):
↪(split[0]+split[1]+split[2]+split[3]+split[4])],
              str1[(split[0]+split[1]+split[2]+split[3]+split[4]):
↪(split[0]+split[1]+split[2]+split[3]+split[4]+split[5])],
              str1[(split[0]+split[1]+split[2]+split[3]+split[4]+split[5]):]]
  nibbles2 = [str2[:split[0]], str2[split[0]:(split[0]+split[1])],
```

```
            str2[(split[0]+split[1]):(split[0]+split[1]+split[2])],
            str2[(split[0]+split[1]+split[2]):
↪(split[0]+split[1]+split[2]+split[3])],
            str2[(split[0]+split[1]+split[2]+split[3]):
↪(split[0]+split[1]+split[2]+split[3]+split[4])],
            str2[(split[0]+split[1]+split[2]+split[3]+split[4]):
↪(split[0]+split[1]+split[2]+split[3]+split[4]+split[5])],
            str2[(split[0]+split[1]+split[2]+split[3]+split[4]+split[5]):]]

 # Compare the nibbles
 if nibbles1 == nibbles2:
   return '000'
 elif nibbles1[0] == nibbles2[0] and nibbles1[1] == nibbles2[1] and␣
↪nibbles1[2] == nibbles2[2] and nibbles1[3] == nibbles2[3] and nibbles1[4] ==␣
↪nibbles2[4] and nibbles1[5] == nibbles2[5]:
   return '001'
 elif nibbles1[0] == nibbles2[0] and nibbles1[1] == nibbles2[1] and␣
↪nibbles1[2] == nibbles2[2] and nibbles1[3] == nibbles2[3] and nibbles1[4] ==␣
↪nibbles2[4]:
   return '010'
 elif nibbles1[0] == nibbles2[0] and nibbles1[1] == nibbles2[1] and␣
↪nibbles1[2] == nibbles2[2] and nibbles1[3] == nibbles2[3]:
   return '011'
 elif nibbles1[0] == nibbles2[0] and nibbles1[1] == nibbles2[1] and␣
↪nibbles1[2] == nibbles2[2]:
   return '100'
 elif nibbles1[0] == nibbles2[0] and nibbles1[1] == nibbles2[1]:
   return '101'
 elif nibbles1[0] == nibbles2[0]:
   return '110'
 else:
   return '111'
```

[ ]:
```
def l2sb(input, split):
  '''
  Params:
    input : a list of 12-bit binary string
    split :  a list of 7 integers that sum up to 12
  Use L2SB algorithm and return a list of compressed result.
  '''
  res = []
  res.append(input[0])
  for i in range(1, len(input)):
    code = compare(input[i], input[i-1], split)
    if code == '111':
      res.append('111' + input[i])
```

```
    elif code == '110':
      res.append('110' + input[i][split[0]:])
    elif code == '101':
      res.append('101' + input[i][(split[0]+split[1]):])
    elif code == '100':
      res.append('100' + input[i][(split[0]+split[1]+split[2]):])
    elif code == '011':
      res.append('011' + input[i][(split[0]+split[1]+split[2]+split[3]):])
    elif code == '010':
      res.append('010' +␣
↪input[i][(split[0]+split[1]+split[2]+split[3]+split[4]):])
    elif code == '001':
      res.append('001' +␣
↪input[i][(split[0]+split[1]+split[2]+split[3]+split[4]+split[5]):])
    else:
      res.append('000')
  return res
```

[ ]:
```
def total_length(strings):
  '''
  Params:
    strings : a list of string
  Return the total length of all strings in input string.
  '''
  return sum(len(s) for s in strings)
```

[ ]:
```
def calRatio_all(input, split):
  '''
  Params:
    input : a data frame, each column represents a txt file
    split : an integer, we split the 12 bits into 0-(split-1) and split-12

  Use new l2sb algorithm to compress the input data. Record the length of␣
↪compressed
  and original data of each column, then sum the compressed and original length
  of all data to get the overall compress ratio.
  '''
  original_length = 0
  compressed_length = 0
  for i in range(input.shape[1]):
    col = input.iloc[:,i]
    binarycol = [np.binary_repr(val, width=12) for val in col]
    col_compressed = l2sb(binarycol, split)
    original_length += total_length(binarycol)
    compressed_length += total_length(col_compressed)
  return original_length / compressed_length
```

#Test case

```
[ ]: split = [2,2,2,2,2,1,1]
     input = ['001100110011', #
              '001010101000', # 1 same 110
              '001001010101', # 1,2 same 101
              '001001101010', # 1,2,3 same 100
              '001001100101', # 1,2,3,4 same 011
              '001001100111', # 1-5 same, 010
              '001001100110', # 1-6 same, 001
              '001001100110', # all same, 000
              '011100111001'  # all different 111
             ]
     res = l2sb(input, split)
     print(res)
```

```
['001100110011', '1101010101000', '10101010101', '100101010', '0110101',
'01011', '0010', '000', '1110111001111001']
```

#Transfer data

```
[ ]: df_F_new = df_F.copy()
     df_O_new = df_O.copy()
     df_S_new = df_S.copy()

     df_F_new -= df_F_new.min().min()
     df_O_new -= df_O_new.min().min()
     df_S_new -= df_S_new.min().min()
```

```
[ ]: def find_combinations(target_sum, num_parts, current_combination=[],␣
      ↪current_sum=0):
       if num_parts == 1:
         # The last number must be exactly what is needed to reach the target sum
         if 1 <= target_sum - current_sum <= target_sum:
           yield current_combination + [target_sum - current_sum]
         return

       for i in range(1, target_sum - current_sum - (num_parts - 1) + 1):
         yield from find_combinations(target_sum, num_parts - 1, current_combination␣
      ↪+ [i], current_sum + i)

     # Get all combinations of 5 positive integers that sum to 12
     split_list = list(find_combinations(12, 7))
```

```
[ ]: f_ratio = {}
     o_ratio = {}
     s_ratio = {}
     for split in split_list:
```

```
  ratio_f = calRatio_all(df_F_new, split)
  ratio_o = calRatio_all(df_O_new, split)
  ratio_s = calRatio_all(df_S_new, split)
  f_ratio[tuple(split)] = ratio_f
  o_ratio[tuple(split)] = ratio_o
  s_ratio[tuple(split)] = ratio_s
```

[ ]:
```
import openpyxl
from openpyxl import Workbook

# Create a new workbook and select the active worksheet
wb = Workbook()
ws = wb.active

# Write the headers
ws.append(['Key', 'Value'])

# Write the key-value pairs
for key, value in o_ratio.items():
  ws.append([str(key), value])


# Save the workbook to a file
wb.save('O-output-7.xlsx')
```

[ ]:
```
for split in split_list:
  ratio_s = calRatio_all(df_S_new, split)
  s_ratio[tuple(split)] = ratio_s
```

[ ]:
```
top_5_f_ratio = sorted(f_ratio.items(), key=lambda item: item[1],␣
 ↪reverse=True)[:5]
print(top_5_f_ratio)
```

```
[((4, 2, 1, 1, 1, 1, 2), 1.6101131506983903), ((4, 2, 1, 1, 1, 2, 1),
1.601017064619685), ((4, 1, 2, 1, 1, 1, 2), 1.5966547024735092), ((4, 1, 1, 1,
1, 1, 3), 1.594525863369073), ((4, 1, 1, 1, 1, 2, 2), 1.5898963773297152)]
```

[ ]:
```
top_5_o_ratio = sorted(o_ratio.items(), key=lambda item: item[1],␣
 ↪reverse=True)[:5]
print(top_5_o_ratio)
```

```
[((4, 1, 1, 1, 1, 1, 3), 1.3745742898913542), ((4, 1, 1, 1, 1, 2, 2),
1.3721274813713125), ((2, 2, 1, 1, 1, 1, 4), 1.3713428834182821), ((2, 2, 1, 1,
1, 2, 3), 1.3704438053648678), ((2, 2, 2, 1, 1, 1, 3), 1.3669641434661732)]
```

[ ]:
```
top_5_s_ratio = sorted(s_ratio.items(), key=lambda item: item[1],␣
 ↪reverse=True)[:5]
```

```
print(top_5_s_ratio)
```

```
[((2, 1, 1, 1, 1, 2, 4), 1.1770804029424615), ((2, 2, 1, 1, 1, 2, 3),
1.1764691806185532), ((3, 1, 1, 1, 1, 2, 3), 1.1763740335363746), ((2, 2, 1, 1,
1, 1, 4), 1.1761522710977876), ((3, 1, 1, 1, 1, 1, 4), 1.176057175266822)]
```

#Import MIT data

[ ]:
```
os.chdir('/content/drive/My Drive/0-Project/MIT')
# 48 csv files
csv_files = [f for f in os.listdir() if f.endswith('.csv')]

df_mix = pd.DataFrame()

for file in csv_files:
  file_path = os.path.join(os.getcwd(), file)

  df = pd.read_csv(file_path, skiprows=1)

  df_last_two_cols = df.iloc[:, -2:]

  file_prefix = file[:3]
  df_last_two_cols.columns = [f"{file_prefix}-1", f"{file_prefix}-2"]

  df_mix = pd.concat([df_mix, df_last_two_cols], axis=1)
```

[ ]:
```
# scale the data
df_mix_scaled = df_mix.mul(600)
# round the scaled result into integers
df_mix_scaled = df_mix_scaled.round().astype(int)
scaled_min = df_mix_scaled.min().min()
# add offset to all data, make them non-negative
df_mix_scaled = df_mix_scaled.add(abs(scaled_min))

# choose 1,3,5.. columns in df_mixed_scaled as mixed_signal_1
mixed_signal_1 = df_mix_scaled.iloc[:,1::2]
# choose 0,2,4.. columns in df_mixed_scaled as mixed_signal_2
mixed_signal_2 = df_mix_scaled.iloc[:,::2]
```

[ ]:
```
mit_ratio_list1 = {}
mit_ratio_list2 = {}
for split in split_list:
  ratio1 = calRatio_all(mixed_signal_1, split)
  ratio2 = calRatio_all(mixed_signal_2, split)
  mit_ratio_list1[tuple(split)] = ratio1
  mit_ratio_list2[tuple(split)] = ratio2
```

[ ]:
```
top_5_mix_ratio_1 = sorted(mit_ratio_list1.items(), key=lambda item: item[1],␣
↪reverse=True)[:5]
top_5_mix_ratio_2 = sorted(mit_ratio_list2.items(), key=lambda item: item[1],␣
↪reverse=True)[:5]
print(top_5_mix_ratio_1)
print(top_5_mix_ratio_2)
```

```
[((3, 2, 1, 1, 1, 1, 3), 1.602887919206286), ((4, 1, 1, 1, 1, 1, 3),
1.5985642524769421), ((4, 2, 1, 1, 1, 1, 2), 1.5953751355060986), ((2, 3, 1, 1,
1, 1, 3), 1.5952634461875543), ((3, 1, 2, 1, 1, 1, 3), 1.5947040076782044)]
[((3, 2, 1, 1, 1, 1, 3), 1.5931357484418984), ((3, 1, 2, 1, 1, 1, 3),
1.5871460762222005), ((2, 2, 2, 1, 1, 1, 3), 1.584917601252892), ((3, 2, 2, 1,
1, 1, 2), 1.5810457501389592), ((4, 1, 1, 1, 1, 1, 3), 1.5810288734483067)]
```

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
!jupyter nbconvert --to pdf "/content/drive/My Drive/0-Project/Copy of 7␣
↪nibbles.ipynb"
```

Bibliography

[1] E. K. S. Louis *et al.*, "Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults," *Children, and Infants,* pp. 1-95, 2016.

[2] R. Soikkeli, J. Partanen, H. Soininen, A. Pääkkönen, and P. Riekkinen Sr, "Slowing of EEG in Parkinson's disease," *Electroencephalography and clinical neurophysiology,* vol. 79, no. 3, pp. 159-165 %@ 0013-4694, 1991.

[3] C. Ge, "The significance of periodic lateralized epileptiform discharges in EEG: an electrographic, clinical and pathological study," *Electroencephalograph Clin Neurophysiol,* vol. 17, pp. 177-193, 1964.

[4] C. Dai and C. Bailey, "A lossless data reduction technique for wireless EEG recorders and its use in selective data filtering for seizure monitoring," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015: IEEE, pp. 6186-6189.

[5] G. Antoniol and P. Tonella, "EEG data compression techniques," *IEEE transactions on bio-medical engineering,* vol. 44, 2, pp. 105-114, Feb 1997.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[7] G. Held and T. R. Marshall, *Data Compression*. New York: Wiley, 1991.

[8] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1992.

[9] J. S. Lawrence Marple, *Digital Spectral Analysis with Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[10] J. Markel and A. Grey, *Linear Prediction of Speech*. New York: Springer-Verlag, 1976.

[11] P. S. Hamilton and W. J. Tompkins, "Theoretical and experimental rate distortion performance in compression of ambulatory ECGs," *IEEE transactions on biomedical engineering,* vol. 38, no. 3, pp. 260-266, 1991.

[12] R. Battiti, A. Sartori, G. Tecchiolli, P. Tonella, and A. Zorat, "Neural compression: an integrated application to EEG signals," in *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications, Stockholm*, 1995, pp. 210-219.

[13] H. B. M. Riedmiller and H. Braun, "Rprop: a fast and robust backpropagation learning strategy," in *Proc. of the ACNN*, 1993.

[14] N. Ahmed and K. R. Rao, *Orthogonal transforms for digital signal processing*. Springer Science & Business Media, 2012.

[15] A. Cohen, F. Flomen, and N. Drori, "EEG sleep staging using vectorial autoregressive models," *Advances in Processing and Pattern Analysis of Biological Signals,* pp. 45-55, 1996.

[16] C. Crispin-Bailey, C. Dai, and J. Austin, "A 65-nm CMOS lossless bio-signal compression circuit with 250 FemtoJoule performance per bit," *IEEE Transactions on Biomedical Circuits and Systems,* vol. 13, no. 5, pp. 1087-1100, 2019.

[17] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state," *Physical Review E,* vol. 64, no. 6, p. 061907, 2001.

[18] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," Massachusetts Institute of Technology, 2009.