A Report on

Effects of Transcranial Magnetic Stimulation on working memory of healthy young adults

By

Omkar Mohan Jadhav — **B.Tech. Information Technology**

Yash Shivaji Bandal — **B.Tech. Information Technology**

Prepared in the fulfilment of

Research Project

(Neuroscience/Clinical Research)

Under the Guidance of

Dr. Dipmala Salunke

Associate Professor, Department of Information Technology

JSPM's Rajarshi Shahu College of Engineering

CERTIFICATE

This is to certify that the project entitled **'Effects of Transcranial Magnetic Stimulation on Working Memory of Healthy Young Adults'** submitted by Omkar Jadhav and Yash Bandal in fulfillment of the requirements of Research Project at JSPM's RSCOE, is an authentic work carried out by them under my supervision and guidance during January 2025.

Date:

Place : JSPM's RSCOE

Dr. Dipmala Salunke

Associate Professor, Department of Information Technology

JSPM's Rajarshi Shahu College of Engineering

Tathawade Campus

ABSTRACT

Electroencephalography (EEG) is a powerful tool for recording the electrical activity of the brain through electrodes placed on the scalp. This technique is widely used in clinical contexts to diagnose various neurological conditions, including epilepsy, sleep disorders, and encephalopathies.

In this study, we investigated the impact of intermittent theta-burst stimulation (iTBS) applied to the left dorsolateral prefrontal cortex (DLPFC) on cognitive performance and EEG correlates of working memory in healthy young adults. Fourteen right-handed female participants, aged 19 to 23, were randomly assigned to either an Active iTBS group (N=7) or a sham iTBS group (N=7). Participants performed 2-back and 3-back working memory tasks before and after iTBS, with accuracy and reaction time measured to assess cognitive performance. Simultaneous EEG recordings were made at three stages: before iTBS, 10 minutes post-iTBS, and 30 minutes post-iTBS. Key EEG electrodes included frontal (Fz, F3, F7), central (C3, FC1, FC5), parietal (Pz, P3, P7), and occipital (O1, Oz, O2) regions, with particular focus on the frontal and central electrodes to capture brain activity related to working memory.

The study aimed to compare cognitive performance and EEG patterns associated with working memory before and after iTBS, as well as to examine intergroup differences between the active and sham groups. Additionally, machine learning methods were applied to EEG data to analyze neural network activity and intergroup differences. Our findings provide insights into how non-invasive brain stimulation affects working memory performance and the underlying neural mechanisms.

ACKNOWLEDGEMENT

Firstly, we would thank the Department of Information Technology, JSPM's RSCOE giving us this opportunity of implementing and gaining practical knowledge as a part of the Neuroengineering , Clinical Research .

We would like to express our sincere thanks to Asst. Prof Dr.Dipmala Salunke, Department Of Information Technology, JSPM's RSCOE for offering us a project in the area of our interest and guiding us

Table Of Contents

EEG-Based Cognitive Assessment

1 EEG Introduction

1.1 EEG generation from the brain

When neurons are activated, they produce synaptic currents which then induce a magnetic field measurable by EMG and a secondary electrical field over the scalp measurable by EEG. The human head consists of different layers including the scalp, skull, brain and many other thin layers in between. The skull attenuates the signals approximately one hundred times more than the soft tissue. On the other hand, most of the noise is generated either within the brain (internal noise) or over the scalp (system noise or external noise).Therefore, only large populations of active neurons can generate enough potential to be recordable using the scalp electrodes. These measurements are then amplified greatly before further processing.

1.2 Applications of EEG signal processing

- Monitoring alertness, coma, and brain death;
- Locating areas of damage following head injury, stroke, and tumor;
- Testing afferent pathways (by evoked potentials);
- Monitoring cognitive engagement (alpha rhythm);
- Producing biofeedback situations;
- Controlling anesthesia depth (servo anesthesia);
- Investigating epilepsy and locating seizure origin;
- Testing epilepsy drug effects;
- Assisting in experimental cortical excision of epileptic focus;
- Monitoring the brain development;
- Testing drugs for convulsive effects;
- Investigating sleep disorders and physiology;

1.3 Characteristics of EEG signals

EEG signals reflect often brain rhythms that reflect the current state of the brain for example sleep or wakefulness. The main characteristics that are used to make inferences about the brain rhythms are the frequency and amplitude of the signal. These characteristics are however not independent of other factors and often change with age and also show variations from person to person.
The main brain waves distinguished by frequency are listed below.

1.3.1 Delta Waves

Delta waves lie within the range of 0.5–4 Hz. These waves are primarily associated with deep sleep and may be present in the waking state. It is very easy to confuse artefact signals caused

by the large muscles of the neck and jaw with the genuine delta response. This is because the muscles are near the surface of the skin and produce large signals, whereas the signal that is of interest originates from deep within the brain and is severely attenuated in passing through the skull. Signal processing methods need to be applied to distinguish genuine delta responses from artefacts.

1.3.2 Theta Waves

Theta waves lie within the range of 4–7.5 Hz. They are generally associated with access to subconscious material and creative inspiration or deep meditation. The theta wave plays an important role in infancy and childhood. Larger contingents of theta wave activity in the waking adult are abnormal and are caused by various pathological problems. The changes in the rhythm of theta waves have been used in maturational and emotional studies.

1.3.3 Alpha Waves

Lie in the range of 8-13 Hz and are mostly recorded over the posterior part of the brain called the occipital region of the brain. Most commonly they are rounded or sinusoidal in shape but in rare cases observed to have sharp negative peaks with rounded positive peaks. Alpha waves have been thought to indicate both a relaxed awareness without any attention or concentration. The alpha wave is the most prominent rhythm in the whole realm of brain activity and possibly covers a greater range than has been previously documented. Peaks have been observed in both the beta and the other ranges while in an alpha setting, showing alpha characteristics. The normal amplitude of the alpha wave is around 50uV. The origin and the significance of the alpha wave is an active research area.

1.3.4 Beta Waves

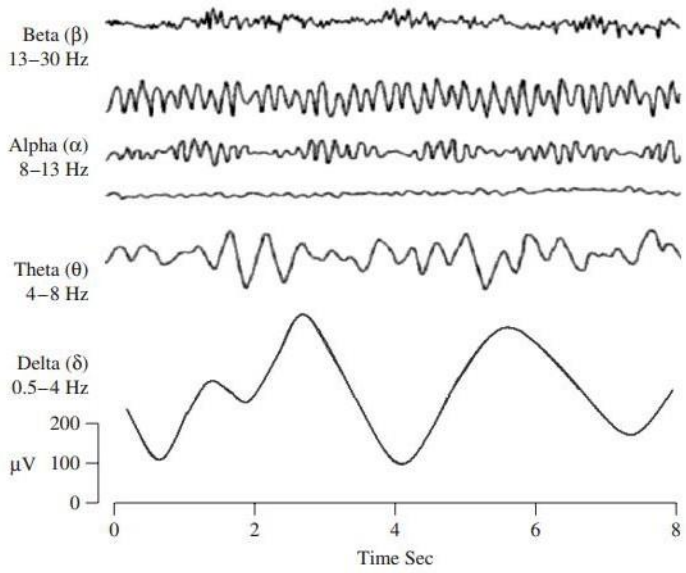
A beta wave is the electrical activity of the brain varying within the range of 14–26 Hz. A beta wave is the usual waking rhythm of the brain associated with active thinking, active attention, focus on the outside world, or solving concrete problems, and is found in normal adults. A high-level beta wave may be acquired when a human is in a panic state. Rhythmical beta activity is encountered chiefly over the frontal and central regions. Importantly, a central beta rhythm can be blocked by motor activity or tactile stimulation. The amplitude of beta rhythm is normally under 30μV.

1.3.5 Gamma Waves

The frequencies above 30 Hz correspond to the gamma range. Although the amplitudes of these rhythms are very low and their occurrence is rare, detection of these rhythms can be used for confirmation of certain brain diseases. The regions of high EEG frequencies and highest levels of cerebral blood flow (as well as oxygen and glucose uptake which are relevant to fMRI measurements) are located in the frontocentral area. The gamma wave band has also been proved to be a good indication of event-related synchronization (ERS) of the brain and can be used to demonstrate the locus for right and left index finger movement, right toes, and the rather broad and bilateral area for tongue movement.

The image shows the various types of waves described above and their frequency bands. Note that here the gamma waves are included in the beta range. This is a somewhat prevalent practice and the gamma waves are sometimes called fast beta waves.

These rhythms are cyclic in nature and correspond to the steady state responses of the brain, in addition to this transients such as an event-related potential (ERP) and containing positive occipital sharp transient (POST) signals (also called rho (ρ) waves) may be observed in EEG signals. Artefacts caused by the eye interference such as fluttering of eyelids may be similar to an alpha rhythm in the posterior part of the brain and need to be filtered out. Other extraneous signals may be caused by any bone defects or brain malfunction.



1.4 EEG RECORDING SYSTEMS

1.4.1 Requirements and Current Standards

Electroencephalography (EEG) is a pivotal tool in understanding brain functions and examining cognitive processes like memory. In your study, EEG plays a central role in investigating the effects of intermittent theta-burst stimulation (iTBS) on working memory performance. The EEG system used in this study, along with the electrodes and standards followed, ensure high-quality and reliable data acquisition to support the investigation of neural correlates of cognitive functions.

The EEG recording electrodes and their proper function are crucial for acquiring high quality data. Different types of electrodes are often used in the EEG recording systems, such as:

- disposable (gel-less, and pre-gelled types);
- reusable disc electrodes (gold, silver, stainless steel, or tin);
- headbands and electrode caps;
- saline-based electrodes;
- Needle electrodes.

In this study, EEG data was recorded using a **128-electrode R-Net MR cap**, which offers high-density electrode coverage for accurate neural activity capture. The electrode setup follows the **international 10-20 system** with additional electrodes placed to ensure optimal signal detection across relevant brain regions. Specifically, the system includes electrodes such as **Fp1, Fp2, Fz, Cz, Pz**, and others, ensuring comprehensive coverage of both frontal and parietal areas, which are key for examining the dorsolateral prefrontal cortex (DLPFC) activity involved in working memory tasks.

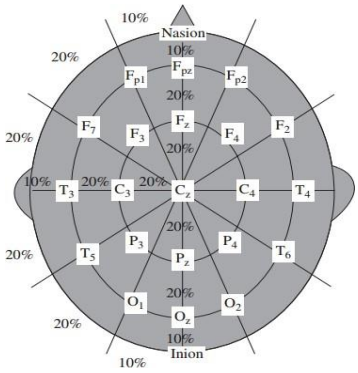
The system meets international standards, including **IEEE 1033-2018**, **ISO 13485**, and **IEC 60601-1**, ensuring data quality, safety, and compliance with clinical research protocols. The setup allows for high-resolution data collection, critical for analyzing event-related potentials (ERPs) and power spectral densities (PSDs) before and after the application of intermittent theta-burst stimulation (iTBS) on participants.

1.4.2 Modes of Recording

For the EEG recordings in this study, the **differential mode** was used, where each electrode pair records the difference in voltage between two electrodes. This method allows for capturing the neural activity at specific brain regions while minimizing common noise. In our setup, the electrodes are positioned according to the **international 10-20 system** and are supplemented with additional electrodes for better signal accuracy, particularly in the **frontal and parietal regions** critical for working memory tasks.

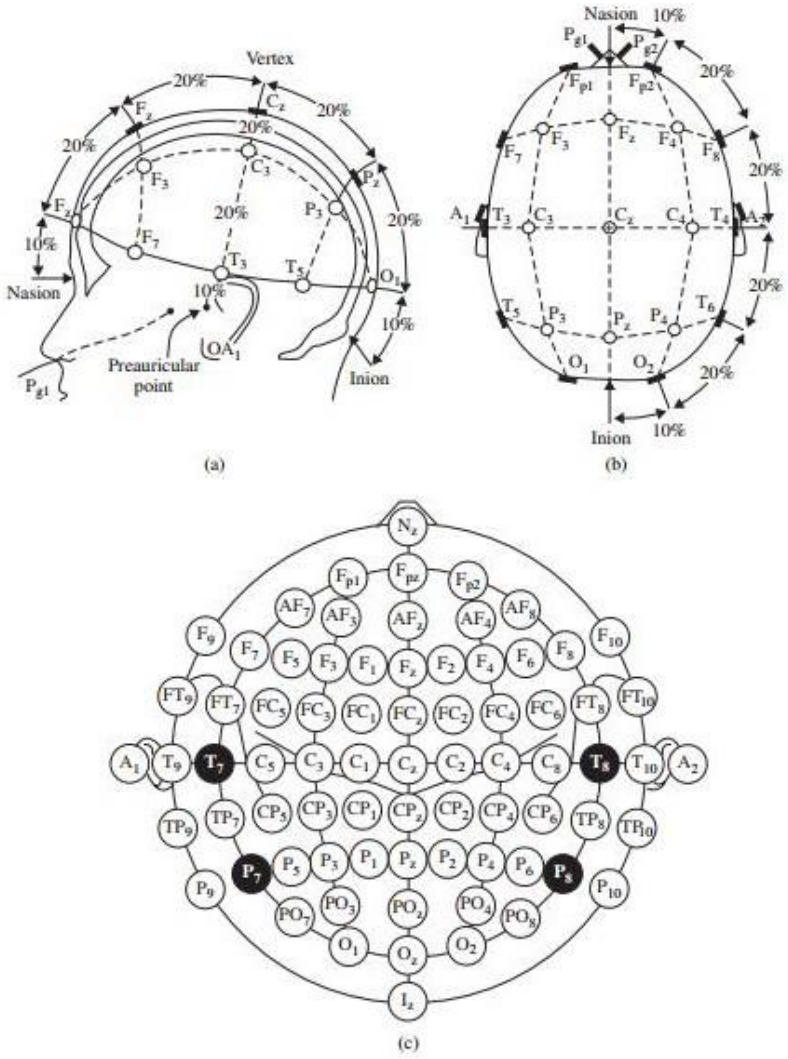
1.4.3 Electrode Placement Standards

The diagram below represents the placement of 21 electrodes in a standard recommended system called the 10-20 system.



Several other recording systems exist for example, in the Maudsley electrode positioning system, the conventional 10–20 system has been modified to capture better the signals from epileptic foci in epileptic seizure recordings. The only difference between this system and the 10–20 conventional system is that the outer electrodes are slightly lowered to enable better capturing of the required signals. The advantage of this system over the conventional one is that it provides a more extensive coverage of the lower part of the cerebral convexity which is important for epilepsy studies. Many systems have been proposed by researchers over the years mostly to cater to various specialized needs.

The diagram in the right shows a more detailed diagram of the 10-20 system for the placement of 75 electrodes around the skull:

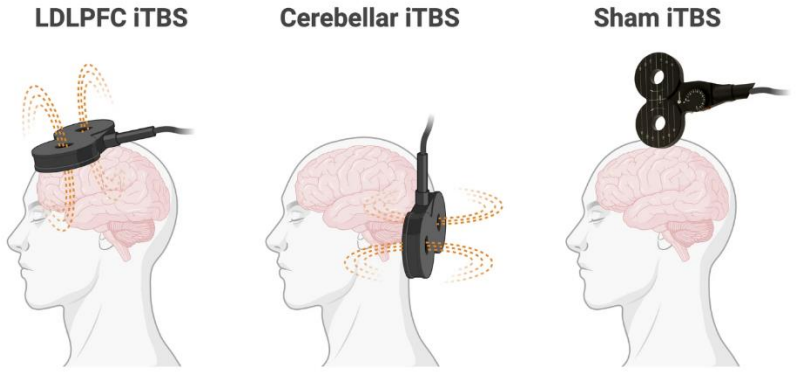


2. LITERATURE REVIEW

2.1 Previous Studies on iTBS and Working Memory

Intermittent theta-burst stimulation (iTBS) has been studied for its ability to enhance cognitive functions like working memory. Research suggests that iTBS, especially when applied to the dorsolateral prefrontal cortex (DLPFC), can improve accuracy and reaction time in working memory tasks like the N-back. Several studies have demonstrated that iTBS increases brain excitability and alters neural activity, particularly in the theta and alpha frequency bands, which are critical for cognitive control. While some studies report significant improvements in working memory performance, others show minimal or no effects, highlighting the need for further exploration.

Despite positive findings, there are discrepancies in results, and variability in factors such as participant characteristics, stimulation protocols, and task designs contribute to inconsistent outcomes.



iTBS administrations. *LDLPFC* left dorsolateral prefrontal cortex, *iTBS* intermittent theta burst stimulation. Sham iTBS targeted the vertex (Cz of the 10–20 EEG system) with a stimulation intensity of 15% of the maximum stimulator output.

2.2 Research Gaps and Relevance of the Study

Despite the growing body of research on iTBS, several gaps remain. Most studies only investigate short-term effects, neglecting long-term changes in cognitive function. Furthermore, the individual variability in response to iTBS is poorly understood. There is also a lack of studies that integrate both behavioral and neurophysiological data to understand how iTBS influences working memory.

This study aims to address these gaps by analyzing both behavioral performance and EEG activity before and after iTBS. By comparing 2-back and 3-back tasks, it will explore how task complexity influences iTBS effects. Additionally, it will examine differences between active and sham iTBS groups to clarify the true impact of stimulation. The findings will contribute to a better understanding of iTBS and its potential applications in cognitive enhancement.

3. STATEMENT OF PURPOSE

3.1 PROBLEM STATEMENT

Working memory is a crucial cognitive function that allows individuals to temporarily store and manipulate information. Intermittent theta-burst stimulation (iTBS), a non-invasive brain stimulation technique, has shown potential for enhancing cognitive functions, particularly when applied to the dorsolateral prefrontal cortex (DLPFC). However, the effects of iTBS on working memory performance remain inconsistent across studies, with variability in cognitive task outcomes and neural responses. Additionally, there is limited understanding of how task complexity influences iTBS effects and how these effects differ between active and sham stimulation groups. A comprehensive analysis combining behavioral and EEG data is necessary to better understand the potential benefits of iTBS in improving working memory performance.

3.2 OBJECTIVES

The primary objective of this study is to evaluate the effects of iTBS on working memory performance in healthy young adults using behavioral and EEG analysis. The study aims to achieve the following specific objectives:

3.2.1 Comparative analysis of cognitive test performance (accuracy and reaction time) for 2-back and 3-back tasks before and after iTBS.

3.1.1 Comparative analysis of EEG during performance of cognitive tests before and after iTBS

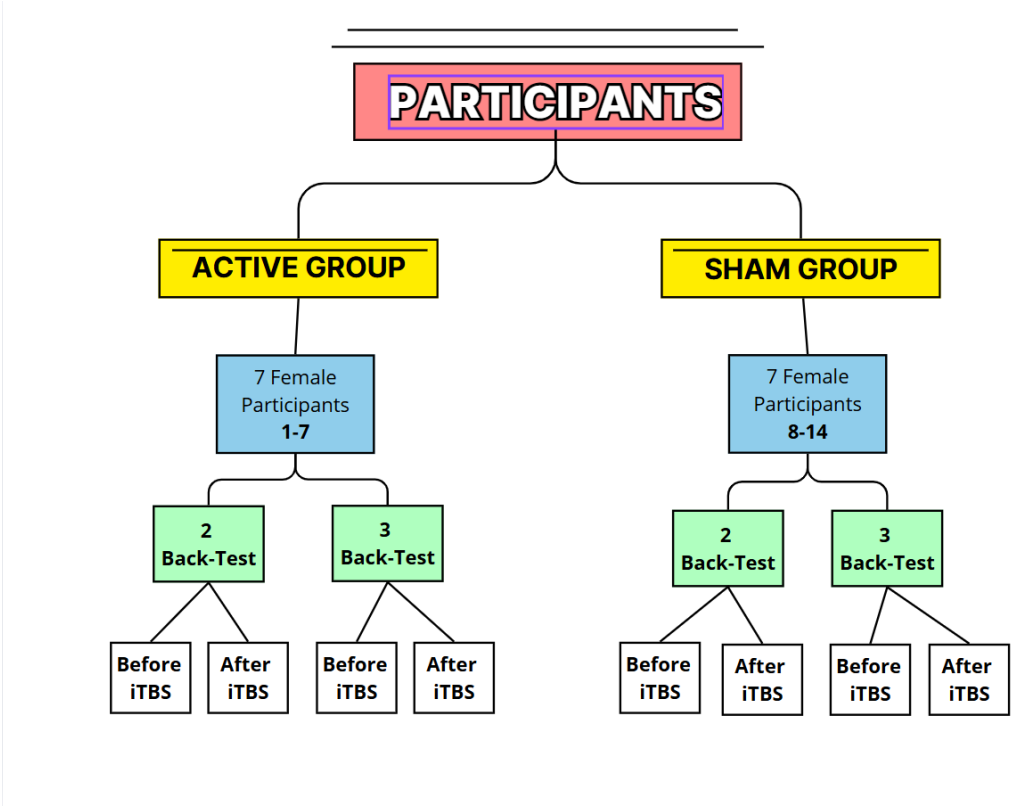
3.1.2 Comparative analysis of EEG during performance of cognitive tests of different complexity (2-back and 3-back) before and after iTBS

3.1.3 Analysis of intergroup differences between active and sham iTBS

4 DATA DESCRIPTION

4.1 EXPERIMENTAL CONDITIONS

The experimental design of this study investigates the effects of **intermittent theta-burst stimulation (iTBS)** on working memory performance in healthy young adults. The study included 14 healthy female participants (mean age: 20.07 ± 2.83 years), who were randomly assigned to two groups: the **Active iTBS group (EG, N=7)** and the **Sham iTBS group (CG, N=7)**. Each participant underwent working memory testing using the 2-back and 3-back tasks, with EEG recordings taken before (pre) and after (post) iTBS. The post-iTBS recordings were taken at two intervals: 10 minutes and 30 minutes following stimulation. The Active group received real iTBS to the left dorsolateral prefrontal cortex, while the Sham group received placebo stimulation. This design allowed for a comparison of cognitive performance (accuracy and reaction time) and EEG patterns before and after iTBS between the two groups.

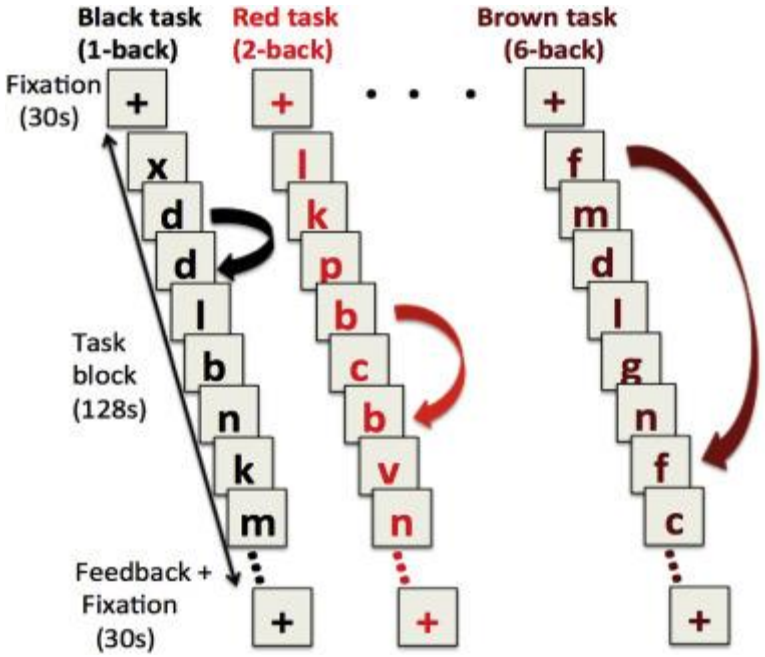


4.1.1 iTBS Protocol

The experimental group (Active) underwent a session of **intermittent theta-burst stimulation (iTBS)** applied to the **left dorsolateral prefrontal cortex (DLPFC)**, which has been shown to be involved in working memory processes. The **Sham group** received a pseudo-stimulation (placebo) protocol to control for non-specific effects of stimulation. Both groups completed the cognitive tasks both **before** and **after** the iTBS (or placebo) intervention. The effects of the stimulation were assessed by comparing the cognitive performance in terms of accuracy and reaction times on the **2-back** and **3-back** tasks.

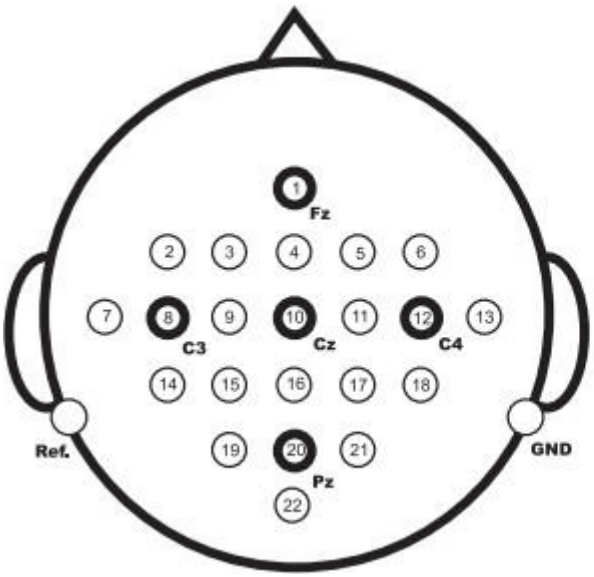
4.1.2 N-Back Test

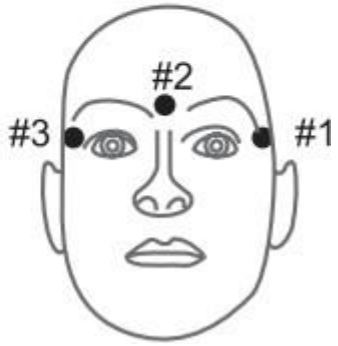
The N-back test was used to assess participants' working memory performance in this study, with both 2-back and 3-back tasks administered before and after intermittent theta-burst stimulation (iTBS) to the left dorsolateral prefrontal cortex (DLPFC). In the 2-back task, participants were instructed to match the current letter with the one presented two positions earlier, while in the 3-back task, they matched the current letter with the one presented three positions earlier. Each task consisted of visual stimuli (Russian alphabet letters) presented on a screen, with a keypress response required for correct matches. EEG data were recorded concurrently with task performance, with markers (S1 for stimulus presentation, S2 for keypress response, and S3 for correct responses) used to segment the EEG data. The tasks were performed in two blocks: one before iTBS and two after (10 and 30 minutes post-stimulation), allowing for an analysis of cognitive performance and brain activity across different time points and between the Active and Sham groups.



4.1 Data Collection and Properties

Data was collected using a 126-electrode R-Net MR cap (Brain Products GmbH, Germany) positioned according to the 10-10 system. The cap's electrodes were soaked in electrolyte (KCl) to ensure good contact with the scalp. The reference electrode was at **CPz**, and the ground electrode was at **FPz**. EEG was recorded during the 2-back and 3-back cognitive tasks, with a sampling frequency of 5 kHz and a common-mode rejection ratio of ≥ 110 dB. Additional electrodes were used for electrooculographic (EOG) recordings to remove eye movement artifacts. EEG data was collected at three intervals: before, 10 minutes, and 30 minutes after iTBS.The electrode placement scheme is shown below for reference.





5 EEG Signal Processing

5.1 Data Import And Synchronization – Merging Partial Records

During the EEG data acquisition process, several test sessions were interrupted, resulting in multiple files for the same test instance. To ensure data continuity and consistency, the following files were merged before further analysis:

Participant 8 – 3-back before iTBS

Files merged: 14032023_1_3_1, 14032023_1_3_11, 14032023_1_3_111

These files belong to the 3-back test conducted before iTBS and were recorded in separate segments due to interruptions.

Participant 8 – 3-back 10 minutes after iTBS

Files merged: 14032023_1_3_10, 14032023_1_3_101

These files represent the 3-back test conducted 10 minutes after iTBS and need to be concatenated for further processing.

Participant 9 – 3-back test (Sham Group, Session 1)

Files merged: 14032023_2_3_10, 14032023_2_3_101, 14032023_2_3_1011

These files correspond to the first 3-back test session of Participant 9 under sham stimulation and were recorded in separate files due to interruptions.

Participant 9 – 3-back test (Sham Group, Session 2)

Files merged: 14032023_2_3_10, 14032023_2_3_101, 14032023_2_3_1011

These files belong to the second 3-back test session of Participant 9 under sham stimulation.

Participant 8 – 2-back 30 minutes after iTBS

Files merged: 14032023_1_2_30, 14032023_1_2_301, 14032023_1_2_3011, 14032023_1_2_30111

These files represent the 2-back test conducted 30 minutes after iTBS.

5.2 Signal Cleaning And Artifact Removal

The EEG signals are often contaminated by various types of noise and artifacts, which can distort the data and compromise the validity of the analysis.

5.2.1 Common EEG Artifacts Observed

EEG data is frequently contaminated by a variety of artifacts, which can be categorized into:

- **Eye Movement Artifacts (EOG):**
  These arise from eye blinks and movements, typically visible as sharp deflections in the EEG signal. The frontal electrodes, especially near the Fp1 and Fp2 locations, are highly sensitive to these artifacts.
- **Muscle Artifacts (EMG):**
  These result from muscle contractions, particularly from facial muscles, and can affect the signal in the frontal and central regions.
- **Powerline Noise:**
  This is a common artifact caused by electrical interference from power lines, typically at 50 Hz (or 60 Hz, depending on the region).
- **Electrode Pop Artifacts:**
  These occur when there is poor contact or loose electrodes, leading to sudden jumps or spikes in the EEG signal.

5.2.2 Filtering Techniques

To clean the EEG data, several filtering techniques are applied to reduce the influence of noise and artifacts. These include:

- **Bandpass Filtering (1-40 Hz):** This filter removes frequencies outside the typical range for EEG, typically between 1 Hz and 40 Hz. It helps to retain the relevant brain activity while filtering out low-frequency drift and high-frequency noise.
- **Notch Filtering (50 Hz):** A notch filter is applied to remove powerline interference, which typically appears as a 50 Hz noise in the sign

ICA (Independent Component Analysis) was used as a computational technique to separate mixed signals into their independent components. In the context of EEG data, ICA was particularly effective for identifying and removing artifacts, such as those caused

Here Filtering takes place in following 7 steps

1. Bandpass Filtering (1-40 Hz)
2. Set EEG Montage (10-20 system)
3. Remove Powerline Noise (50 Hz Notch Filter)
4. Run ICA for Artifact Removal
5. Use Fp1 (or Fp2) as EOG Proxy Channel for Artifact Removal
6. Extract Events from Annotations
7. Create Epochs Aligned to Each Event Type

5.3 Segmenting EEG Data

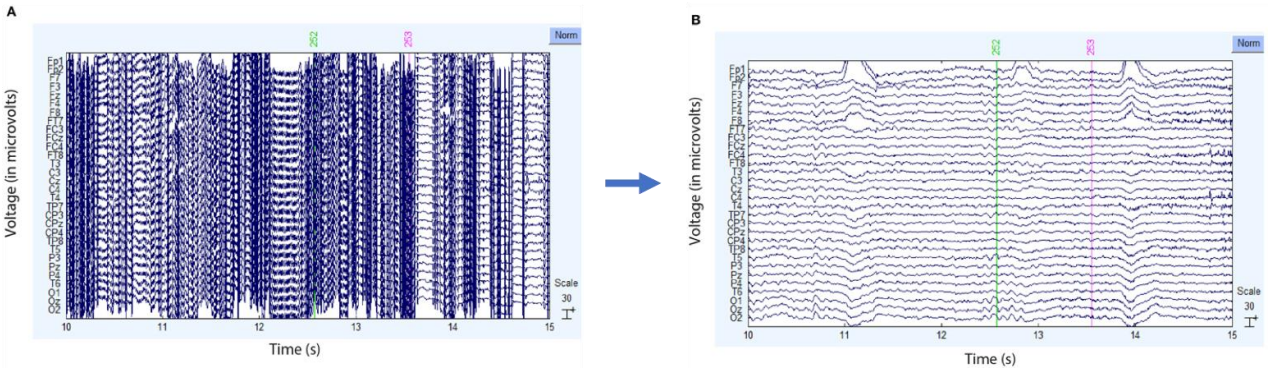
5.3.1 Marker Segmentation (S1 , S2 , S3)

In this step, the EEG data is segmented based on specific markers (S1, S2, and S3) to align the data with key events during the experiment. These markers represent distinct time points in the task:

- **S1**: Corresponds to the appearance of the stimulus (e.g., a letter on the screen). This marker marks the onset of the stimulus presentation.
- **S2**: Represents the key press by the participant, indicating their response to the stimulus.
- **S3**: Marks the correct response, providing feedback on whether the participant's answer was correct.

The segmentation is carried out by creating epochs around these markers. Each epoch is time-locked to the event of interest, with a predefined time window before and after the marker. Typically, this includes a period of -0.2 seconds (200 ms) before the event and +0.8 seconds (800 ms) after the event to capture the relevant brain activity surrounding each event.

Using these markers for segmentation ensures that EEG data is aligned to specific task events, allowing for the analysis of brain activity in response to stimuli, key presses, and feedback. This method is critical for isolating the neural responses that are most relevant for the analysis of working memory performance and cognitive processes during the task.



v

Filtering Signals

6 DATA ANALYSIS

6.1 BEHAVIORAL DATA ANALYSIS

In this section, we analyze the cognitive test performance of participants in both the **Active** and **Sham** groups across the 2-back and 3-back tasks, focusing on key behavioral metrics: accuracy, reaction time (RT), error rates, and omission rates. Data from each participant was collected before iTBS stimulation, and at two post-stimulation time points: 10 minutes and 30 minutes. Below, we provide a detailed description of how these performance metrics are calculated and applied to the data.

The following performance metrics are computed for each participant and used to assess cognitive performance.

1. **Accuracy (%)** Accuracy measures the proportion of correct responses (S3) out of all responses made (S2), expressed as a percentage.

$$Accuracy = \frac{Number\ of\ Correct\ Responses\ (S3)}{Total\ Number\ of\ Responses\ Made\ (S2)} * 100$$

   This metric shows how well participants perform the task in terms of making correct responses.

2. **Error Rate (%)** The error rate is calculated as the proportion of incorrect responses (S2 - S3) out of all responses made (S2), expressed as a percentage.

$$Error\ Rate = \frac{Responses\ Made\ (S2) - Correct\ Responses\ (S3)}{Responses\ Made\ (S2)} * 100$$

   This metric reflects how often participants make incorrect responses relative to the total number of responses.

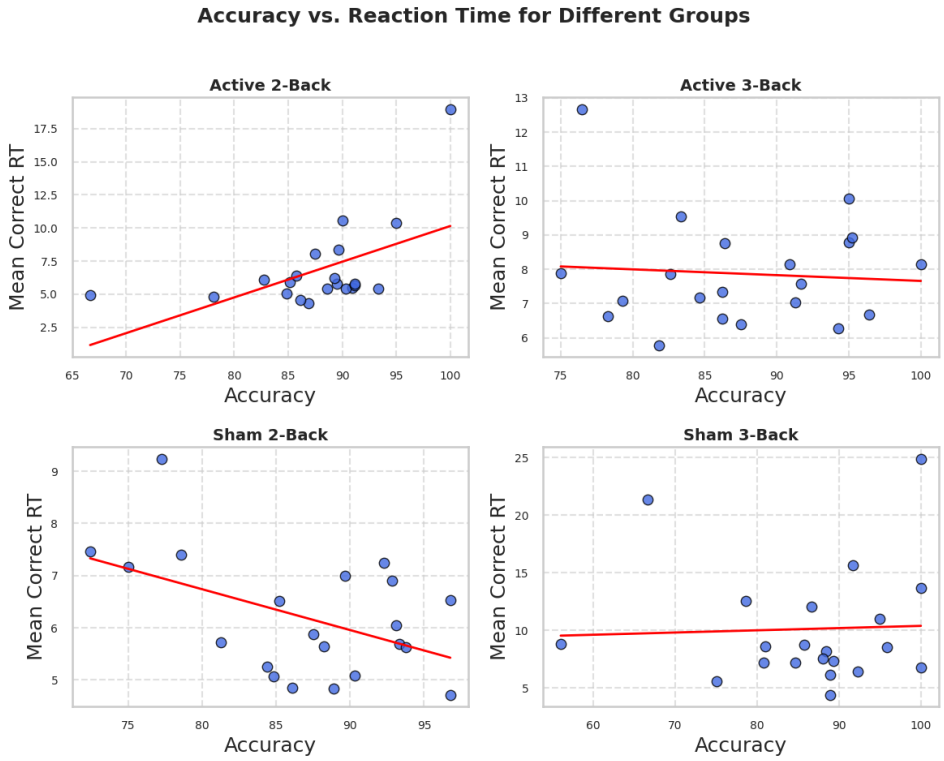
3. **Omission Rate (%)** The omission rate measures the proportion of trials (S1) in which no response was made (S2 = 0) out of all presented trials (S1), expressed as a percentage.

$$Ommision\ Rate = \frac{Number\ of\ Trials\ (S1) - Number\ of\ Responses\ Made\ (S2)}{Total\ Number\ of\ Trials\ (S1)} * 100$$

   This metric highlights how often participants fail to respond when required to do so.

4. **Reaction Time (s):** The reaction times represent the time in **seconds** it took for a participant to respond after a stimulus was presented during the cognitive test.

$$Mean_{RT} = \frac{\sum[\![(Reaction\ Times\ by\ Each\ participant]\!])}{Total\ Reaction\ Times}$$



6.2 EEG Feature Extraction and Analysis

EEG feature extraction is a crucial step in analyzing the neural responses during cognitive tasks such as the 2-back and 3-back tasks in the current study. The goal of feature extraction is to reduce the complexity of the raw EEG data while preserving essential information about brain activity, which can then be used for further analysis and interpretation.

6.2.1 Time-Frequency Analysis

Time-Domain Features

The time-domain analysis focuses on signal characteristics over time, such as the amplitude and frequency of oscillations. For this study, the primary time-domain features considered include:

I. **Mean Amplitude**: The average amplitude of the EEG signal over a defined epoch
II. **Peak-to-Peak Amplitude**: The difference between the maximum and minimum values of the EEG signal within an epoch.

Frequency-Domain Features

In addition to time-domain features, frequency-domain features are often utilized to understand different brain wave activities associated with various cognitive states. For this study, the following frequency bands were of particular interest:

- Delta Band (1-4 Hz):
- Theta Band (4-8 Hz):
- Alpha Band (8-13 Hz):
- Beta Band (13-30 Hz):
- Gamma Band (30-100 Hz

The Fast Fourier Transform (FFT) method was used to decompose the EEG signal into its constituent frequency components. Power spectral densities (PSDs) for each frequency band were then computed for each participant, providing insights into the neural oscillations occurring during the 2-back and 3-back tasks before and after iTBS.

In This Time-Frequency analysis Given the dynamic nature of cognitive tasks, time-frequency analysis provides an effective approach to studying changes in EEG activity over both time and frequency. The **Short-Time Fourier Transform (STFT)** or **Wavelet Transform** were applied to examine how brain activity evolved during task performance, both before and after iTBS. These methods allow for a more precise identification of transient shifts in brain oscillations, particularly those related to cognitive load and working memory processes.

Key time-frequency features analyzed included:

- **Event-related Synchronization (ERS)**: Increased power in specific frequency bands, typically occurring in response to task demands.
- **Event-related Desynchronization (ERD)**: A decrease in power within specific frequency bands, which often corresponds to active cognitive processing or task engagement.

EEG Connectivity Analysis

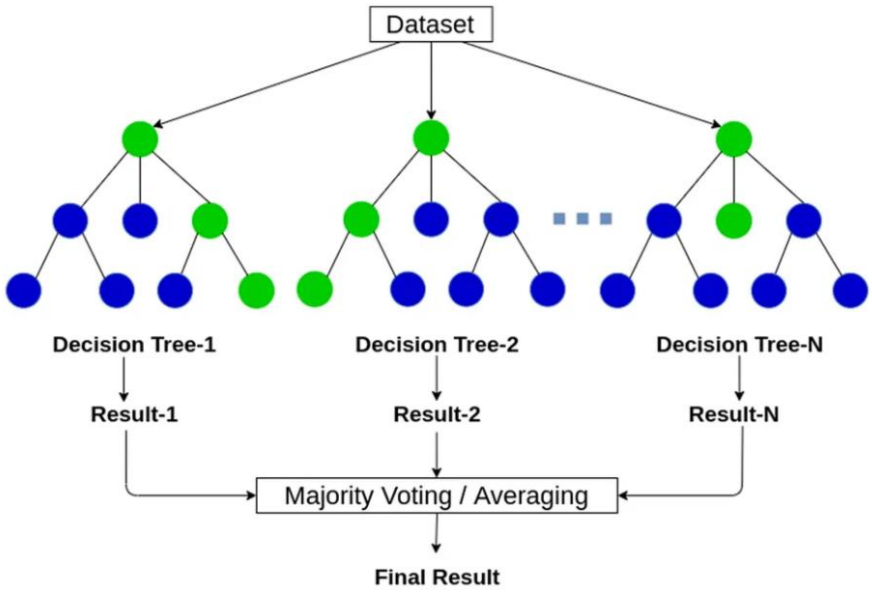
In addition to individual frequency bands, **functional connectivity** between different brain regions was assessed using coherence or phase-locking value (PLV) measures. This analysis helps understand the network dynamics between the left dorsolateral prefrontal cortex (DLPFC) and other brain regions implicated in working memory, such as the parietal and occipital regions. Changes in connectivity after iTBS could provide insights into the modulatory effects of TMS on brain network dynamics involved in working memory tasks.

6.3 Classification Using Random Forest Classifier

In the context of EEG data analysis, Random Forest was employed to classify the features extracted from EEG signals recorded during cognitive tasks (2-back and 3-back) in both the active and sham iTBS groups before and after stimulation.

6.3.1 Random Forest Classifier

The Random Forest classifier builds multiple decision trees, where each tree is trained on a randomly selected subset of data. It creates an ensemble by using different combinations of input data and features, ensuring that each tree is diverse. This diversity reduces the risk of overfitting, a common problem in single decision tree



How does Random Forest algorithm work?

The random forest algorithm operates in the following steps:

Step-1 : Bootstrap Sampling (Bagging):

From the original training dataset, multiple bootstrap samples are generated.

A bootstrap sample is a random sample drawn with replacement, meaning that the same data point may appear more than once in a sample, while others may not appear at all.

Each decision tree in the random forest is trained on a different bootstrap sample.

Step-2 : Random Feature Selection:

At each node of the decision tree, instead of considering all features (as in standard decision trees), the algorithm randomly selects a subset of features. This adds additional randomness and helps decorrelate the trees, making the final model more generalizable.

This random feature selection is a key difference between random forests and bagging with decision trees.

Step-3 : Grow Decision Trees:

Each tree in the forest is grown to the maximum extent without pruning. The goal is to let each tree overfit its particular bootstrap sample.

Step-4 : Aggregation:

For classification, the predictions of all trees are combined using majority voting. The class that gets the most votes becomes the final prediction.

For regression, the predictions of all trees are averaged to produce the final output.

Key Hyperparameters of Random Forests

Tuning the hyperparameters of a random forest model can have a significant impact on its performance. The main hyperparameters include:

1. Number of Trees (n_estimators):

   This parameter specifies how many trees the random forest should include. More trees generally improve accuracy, but they also increase computational cost.

2. Number of Features to Consider (max_features):

   Controls how many features the algorithm should consider when making splits at each node. Options include:
   - "**auto":** Use the square root of the number of features (default for classification).
   - **"sqrt":** Same as "auto".
   - **"log2":** Use the logarithm of the number of features.

   An integer: Specify an exact number of features to consider.

3. Tree Depth (max_depth):

   Limits the maximum depth of the individual trees. Shallow trees help reduce overfitting, but if trees are too shallow, the model may underfit.

4. Minimum Samples per Split (min_samples_split):

   The minimum number of samples required to split an internal node. Higher values prevent trees from being too specific to small groups of data, reducing overfitting.

5. Minimum Samples per Leaf (min_samples_leaf):

   The minimum number of samples required to be at a leaf node. It prevents overly small leaves, which can lead to overfitting.

6. Bootstrap Sampling (bootstrap):

   A boolean parameter that controls whether bootstrap samples are used when building trees. If set to False, the entire dataset is used for training each tree

6.3.2 Mathematical Explanation

The Random Forest classifier relies on decision trees as base learners. The classification process involves the following key mathematical concepts:

Decision Tree Split Criterion:

- **Gini Impurity:** Measures the impurity of a dataset. For a dataset DDD with classes CCC, it is calculated as:

$$I_{Gini}(D) = 1 - \sum_{i=1}^{C} P_i^2$$

Where P_i is the proportion of class i in D. The aim is to minimize Gini impurity for better splits.

- **Entropy:** Measures the disorder in a dataset. For dataset DDD, it is calculated as:

$$I_{Entropy}(D) = -\sum_{i=1}^{C} P_i \log_2 P_i$$

The goal is to minimize entropy to achieve better splits.

- **Random Feature Selection:**

For each tree, only a random subset of features is considered at each split. This ensures diversity among trees and prevents overfitting.

- **Ensemble Voting:**

After training, each tree makes a prediction. The final prediction is determined by majority voting:

$$yRF = mode(y_1, y_2, y_3 \dots \dots y_n)$$

Where $y_1, y_2, y_3 \dots \dots y_n$ are the predictions from the T trees, and the mode selects the most frequent class.

- **Out-of-Bag (OOB) Error:**

OOB error provides an unbiased estimate of the model's performance, calculated as:

$$OOB\ Error = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq yRF_i)$$

Where y_i is the true class and yRF is the predicted class. This error is used for validation without needing a separate test set.

6.4 Statistical Analysis Of Pre And Post iTBS Results

Statistical analysis was conducted to evaluate the effects of intermittent theta-burst stimulation (iTBS) on working memory performance by comparing cognitive test results and EEG-derived metrics before and after stimulation. The primary aim was to determine whether iTBS had a significant impact on cognitive performance and EEG features associated with working memory. A within-group and between-group analysis was performed to assess differences over time and between the active and sham stimulation conditions. The statistical analysis involved parametric and non-parametric tests based on data distribution and variance homogeneity.

6.4.1 Group Comparisons (Active vs Sham)

Group comparisons were performed to analyze the differences between the active iTBS group and the sham iTBS group across various metrics, including accuracy, reaction time, and EEG feature parameters such as power spectral density and event-related potentials. A repeated-measures ANOVA was conducted to assess the interaction effects of time (pre and post-iTBS) and group (active vs sham). Post-hoc analyses were carried out using paired t-tests with Bonferroni correction to adjust for multiple comparisons. Effect sizes were calculated to measure the magnitude of differences between groups. The results were visualized using box plots and line graphs to illustrate trends in performance and neural activity changes over time. The findings aimed to reveal whether active stimulation led to statistically significant improvements compared to the sham condition.

- **Normality Check:** The Shapiro-Wilk test was used to assess the normality of data distribution.
- **Paired t-tests:** Conducted within each group to compare pre and post-iTBS results.
- **Independent t-tests:** Used to compare the Active and Sham groups at each time point.
- **Effect Size Calculation:** Cohen's d was calculated to measure the effect size of iTBS intervention.

6.4.2 Correlation Analysis with Behavioral Metrics

Correlation analyses were conducted to investigate the relationship between EEG-derived features and behavioral performance metrics such as accuracy and reaction time. The EEG features analyzed included time-domain (e.g., mean amplitude), frequency-domain (e.g., power spectral density in theta and alpha bands), and time-frequency representations The correlation results were examined to determine if specific EEG patterns could serve as potential biomarkers of cognitive improvement following iTBS. Significant correlations were further validated through regression analyses to assess the predictive value of EEG features for behavioral outcomes. These findings provide insights into the neural mechanisms underlying working memory enhancement induced by iTBS.

7 Implementation Code

Included in this section the codes implemented as part of this project.

7.3 Code For Organizing Data into a Readable Format | Load Data

```
%% Patricipant data Both Active and Sham Group
Back2before = '/path/to/.vhdr '
Back2/3_10min = '/path/to/.vhdr '
Back2/3_30min = '/path/to/.vhdr '
Participant_Back2/3before = mne.io.read_raw_brainvision(Back2before, preload=True)
participant_Back2/3_10min = mne.io.read_raw_brainvision(Back2_10min, preload=True)
participant_Back2/3_30min = mne.io.read_raw_brainvision(Back2_30min, preload=True)


%% Event Data
raw = mne.io.read_raw_brainvision(Back2before, preload=True)

# Extract and inspect event markers
events, event_id = mne.events_from_annotations(raw)
```

7.4 Code For Preprocessing Participant Data

```
# segmented epoch
def preprocess_eeg(raw):
    # 1. Bandpass Filtering (1-40 Hz)
    raw.filter(1., 40., fir_design='firwin')

    # 2. Set EEG montage (assuming 10-20 system)
    montage = mne.channels.make_standard_montage('standard_1020')
    raw.set_montage(montage, on_missing='ignore')

    # 3. Remove powerline noise (50 Hz notch filter)
    raw.notch_filter(freqs=50)

    # 4. Run ICA for artifact removal
    ica = mne.preprocessing.ICA(n_components=20, random_state=97, max_iter='auto')
    ica.fit(raw)

    # 5. Use Fp1 (or Fp2) as EOG proxy channel for artifact removal
    eog_indices, scores = ica.find_bads_eog(raw, ch_name='Fp1')  # Using Fp1 as the
EOG channel
    ica.exclude = eog_indices  # Exclude the detected EOG components
    raw_cleaned = ica.apply(raw)  # Apply ICA to clean the data

    # 6. Extract events from annotations
    events, event_id = mne.events_from_annotations(raw_cleaned)

    # Map S1, S2, S3 markers to specific event IDs for epoching
    event_id = {
```

```
        'Stimulus/S  1': 1,   # Letter appearance s1
        'Stimulus/S  2': 2,   # Key press         s2
        'Stimulus/S  3': 3    # Correct response  s3
    }

    # 7. Create epochs aligned to each event type
    # tmin=-0.2s (200ms before event), tmax=0.8s (800ms after event)
    epochs = mne.Epochs(
        raw_cleaned, events, event_id,
        tmin=-0.2, tmax=0.8,
        baseline=(None, 0), preload=True
    )

    return epochs
```

7.5 Code For Performing Analysis

7.5.1 Finding Reaction Times

```
def calculate_reaction_times(group_data):

    reaction_times_data = {}

    for participant, data in group_data.items():
        reaction_times_data[participant] = {}
        for condition, epochs in data.items():
            # Extract events and event_id from the epochs object
            events = epochs.events
            event_id = epochs.event_id

            # Check if required event IDs are available
            if 'Stimulus/S  1' in event_id and 'Stimulus/S  2' in event_id:
                stimulus_events=events[events[:, 2]==event_id['Stimulus/S  1']]
                response_events=events[events[:, 2]== event_id['Stimulus/S  2']]
                # Calculate reaction times (in seconds)
                reaction_times = []
                for stimulus in stimulus_events:
                    # Find the first response after the stimulus
                    responses_after_stimulus =
response_events[response_events[:, 0] > stimulus[0]]
                    if len(responses_after_stimulus) > 0:
                        response = responses_after_stimulus[0]
                        rt = (response[0] - stimulus[0]) / epochs.info['sfreq']
# Convert samples to seconds
                        reaction_times.append(rt)

                reaction_times_data[participant][condition] = reaction_times
            else:
                # If required events are missing, add an empty list or a message
                reaction_times_data[participant][condition] = f"Missing event
IDs 'Stimulus/S  1' or 'Stimulus/S  2'"

    return reaction_times_data
```

7.5.2 Finding Performance Metrics (Accuracy/ Rates)

```
import pandas as pd

def compute_performance_metrics_to_df(extracted_data):
    # Initialize an empty list to store the data for the DataFrame
    performance_data = []

    for participant, time_points in extracted_data.items():
        for time_point, data in time_points.items():
            total_trials = data['S1_count']
            responses_made = data['S2_count']
            correct_responses = data['S3_count']

            # Recalculate metrics based on correct definitions
            accuracy = (correct_responses / responses_made) * 100 if
responses_made > 0 else 0
            error_rate = ((responses_made - correct_responses) / responses_made)
* 100 if responses_made > 0 else 0
            # omission_rate = ((total_trials - responses_made) / total_trials) *
100 if total_trials > 0 else 0

            # Collect the metrics and participant info for the DataFrame
            performance_data.append({
                'Participant': participant,
                'Time Point': time_point,
                'S1_count': total_trials,  # Total stimuli presented
                'S2_count': responses_made,  # Responses made
                'S3_count': correct_responses,  # Correct responses
                'Accuracy': accuracy,
                'Error Rate': error_rate,
                'Mean Correct RT': data['mean_RT'],
            })

    # Convert the list of dictionaries into a DataFrame
    performance_df = pd.DataFrame(performance_data)

    return performance_df
```

7.6 Code For Classificatio And Analysis

```
# Configure logging and warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=UserWarning)
mne.set_log_level('WARNING')
logging.getLogger('parallel').setLevel(logging.WARNING)

# Get optimal number of CPU cores
N_CORES = multiprocessing.cpu_count()
```

```
# Performance optimization parameters
PERFORMANCE_CONFIG = {
    'batch_size': 100,      # Batch size for processing
    'n_jobs': N_CORES,      # Number of CPU cores to use
    'freq_step': 8,         # Frequency step size
    'decim': 8,             # Decimation factor
    'n_estimators': 50,     # Number of trees in Random Forest
    'test_size': 0.2,       # Test split ratio
    'random_state': 42      # Random seed
}

def create_model_directory():
    """Create timestamped directory for model storage"""
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    model_dir = f'eeg_models_{timestamp}'
    os.makedirs(model_dir, exist_ok=True)
    return model_dir


def save_model(model_dir, condition, clf, scaler, metadata=None):
    """Save model, scaler, and metadata with timestamp"""
    try:
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')

        # Save paths
        model_path = os.path.join(model_dir,
f'{condition}_model_{timestamp}.joblib')
        scaler_path = os.path.join(model_dir,
f'{condition}_scaler_{timestamp}.joblib')
        metadata_path = os.path.join(model_dir,
f'{condition}_metadata_{timestamp}.json')

        # Save model and scaler
        joblib.dump(clf, model_path)
        joblib.dump(scaler, scaler_path)

        # Prepare metadata
        if metadata is None:
            metadata = {}

        metadata.update({
            'saved_at': timestamp,
            'model_path': model_path,
            'scaler_path': scaler_path,
            'model_type': type(clf).__name__,
            'scaler_type': type(scaler).__name__,
            'performance_config': PERFORMANCE_CONFIG
        })

        # Save metadata
        with open(metadata_path, 'w') as f:
            json.dump(metadata, f, indent=4)

        print(f"\nModel saved successfully in {model_dir}")
```

```
        return {'model_path': model_path, 'scaler_path': scaler_path,
'metadata_path': metadata_path}

    except Exception as e:
        print(f"\nError saving model: {str(e)}")
        raise

def load_model(model_dir, condition, timestamp=None):
    """Load saved model, scaler, and metadata"""
    try:
        model_files = [f for f in os.listdir(model_dir) if
f.startswith(f'{condition}_model_')]

        if not model_files:
            raise FileNotFoundError(f"No saved models found for condition:
{condition}")

        if timestamp is None:
            model_file = sorted(model_files)[-1]
            timestamp = model_file.split('_')[-1].replace('.joblib', '')

        model_path = os.path.join(model_dir,
f'{condition}_model_{timestamp}.joblib')
        scaler_path = os.path.join(model_dir,
f'{condition}_scaler_{timestamp}.joblib')
        metadata_path = os.path.join(model_dir,
f'{condition}_metadata_{timestamp}.json')

        clf = joblib.load(model_path)
        scaler = joblib.load(scaler_path)

        with open(metadata_path, 'r') as f:
            metadata = json.load(f)

        return {'classifier': clf, 'scaler': scaler, 'metadata': metadata}

    except Exception as e:
        print(f"\nError loading model: {str(e)}")
        raise

def extract_eeg_features_optimized(epochs, config=PERFORMANCE_CONFIG):
    """Extract EEG features with optimized parameters"""
    freqs = np.arange(1, 41, config['freq_step'])
    n_cycles = freqs / 2.

    freq_ranges = {
        'delta': (1, 4),
        'theta': (4, 8),
        'alpha': (8, 13),
        'beta': (13, 30)
    }

    all_features = []
    feature_dims = None
```

```
    total_batches = (len(epochs) + config['batch_size'] - 1) //
config['batch_size']

    with tqdm(total=total_batches, desc="Extracting Features", ncols=80) as
pbar:
        for start_idx in range(0, len(epochs), config['batch_size']):
            end_idx = min(start_idx + config['batch_size'], len(epochs))
            batch = epochs[start_idx:end_idx]

            # Ensure consistent epoching
            if hasattr(batch, 'crop'):
                batch = batch.crop(tmin=0, tmax=batch.times[-1])

            with warnings.catch_warnings():
                warnings.simplefilter("ignore")
                power = mne.time_frequency.tfr_morlet(
                    batch,
                    freqs=freqs,
                    n_cycles=n_cycles,
                    return_itc=False,
                    average=False,
                    decim=config['decim'],
                    n_jobs=config['n_jobs'],
                    verbose=False
                )

            batch_features = []
            for band, (fmin, fmax) in freq_ranges.items():
                freq_idx = np.where((freqs >= fmin) & (freqs <= fmax))[0]
                band_power = np.mean(power.data[:, :, freq_idx, :], axis=(2, 3))
                batch_features.append(band_power)

            try:
                batch_features = np.concatenate(batch_features, axis=1)

                if feature_dims is None:
                    feature_dims = batch_features.shape[1]
                elif batch_features.shape[1] != feature_dims:
                    if batch_features.shape[1] > feature_dims:
                        batch_features = batch_features[:, :feature_dims]
                    else:
                        pad_width = ((0, 0), (0, feature_dims -
batch_features.shape[1]))
                        batch_features = np.pad(batch_features, pad_width,
mode='constant')

                all_features.append(batch_features)
            except Exception as e:
                print(f"\nError processing batch: {str(e)}")
                continue

            del power
            gc.collect()
            pbar.update(1)
```

```
    if not all_features:
        raise ValueError("No features were successfully extracted")

    features = np.vstack(all_features)
    print(f"\nFeature extraction complete. Shape: {features.shape}")
    return features

def prepare_ml_data_optimized(active_data, sham_data,
config=PERFORMANCE_CONFIG):
    """Prepare machine learning data with optimization"""
    X = []
    y = []
    feature_dims = None

    total_sessions = len(active_data) * 3 + len(sham_data) * 3

    with tqdm(total=total_sessions, desc="Preparing Data", ncols=80) as pbar:
        # Process active group
        for participant in active_data:
            for session in ['before', '10min', '30min']:
                try:
                    features = extract_eeg_features_optimized(
                        active_data[participant][session],
                        config
                    )

                    if feature_dims is None:
                        feature_dims = features.shape[1]
                    elif features.shape[1] != feature_dims:
                        if features.shape[1] > feature_dims:
                            features = features[:, :feature_dims]
                        else:
                            pad_width = ((0, 0), (0, feature_dims -
features.shape[1]))
                            features = np.pad(features, pad_width,
mode='constant')

                    X.append(features)
                    y.extend([1] * len(features))

                    gc.collect()
                    pbar.update(1)
                except Exception as e:
                    print(f"\nError: participant {participant}, session
{session}: {str(e)}")

        # Process sham group
        for participant in sham_data:
            for session in ['before', '10min', '30min']:
                try:
                    features = extract_eeg_features_optimized(
                        sham_data[participant][session],
                        config
```

```
                    )

                    if feature_dims is None:
                        feature_dims = features.shape[1]
                    elif features.shape[1] != feature_dims:
                        if features.shape[1] > feature_dims:
                            features = features[:, :feature_dims]
                        else:
                            pad_width = ((0, 0), (0, feature_dims -
features.shape[1]))
                            features = np.pad(features, pad_width,
mode='constant')

                    X.append(features)
                    y.extend([0] * len(features))

                    gc.collect()
                    pbar.update(1)
                except Exception as e:
                    print(f"\nError: participant {participant}, session
{session}: {str(e)}")

    if not X:
        raise ValueError("No features were successfully extracted")

    X = np.vstack(X)
    y = np.array(y)
    print(f"\nFinal data shape: X: {X.shape}, y: {y.shape}")
    return X, y

def train_model_optimized(X, y, model_dir, condition,
config=PERFORMANCE_CONFIG):
    """Train model with optimization and progress tracking"""
    print(f"\nTraining {condition} model...")

    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=config['test_size'],
        random_state=config['random_state']
    )

    del X, y
    gc.collect()

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = np.zeros_like(X_train)

    with tqdm(total=len(X_train), desc="Scaling Features", ncols=80) as pbar:
        for i in range(0, len(X_train), config['batch_size']):
            batch = X_train[i:i + config['batch_size']]
            if i == 0:
                X_train_scaled[i:i + config['batch_size']] =
scaler.fit_transform(batch)
```

```
            else:
                X_train_scaled[i:i + config['batch_size']] =
scaler.transform(batch)
            pbar.update(len(batch))

    X_test_scaled = scaler.transform(X_test)

    # Train model
    clf = RandomForestClassifier(
        n_estimators=config['n_estimators'],
        random_state=config['random_state'],
        n_jobs=config['n_jobs'],
        verbose=0
    )

    with tqdm(total=100, desc="Training Model", ncols=80) as pbar:
        clf.fit(X_train_scaled, y_train)
        pbar.update(100)

    # Evaluate
    y_pred = clf.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    # Save model and results
    metadata = {
        'accuracy': accuracy,
        'classification_report': report,
        'data_shape': X_train.shape,
        'training_date': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    }

    save_model(model_dir, condition, clf, scaler, metadata)

    return clf, scaler, accuracy, report

def run_analysis_pipeline(A2B, A3B, S2B, S3B, config=PERFORMANCE_CONFIG):
    """Main analysis pipeline with all optimizations"""
    try:
        # Create model directory
        model_dir = create_model_directory()
        print(f"\nAnalysis started. Model directory: {model_dir}")
        print("\nConfiguration:")
        for key, value in config.items():
            print(f"- {key}: {value}")

        results = {}

        # Process 2-back condition
        print("\nProcessing 2-back condition...")
        X, y = prepare_ml_data_optimized(A2B, S2B, config)
        clf, scaler, accuracy, report = train_model_optimized(X, y, model_dir,
'2back', config)
        results['2back'] = {
```

```
            'accuracy': accuracy,
            'report': report,
            'classifier': clf,
            'scaler': scaler
        }

        del X, y
        gc.collect()

        # Process 3-back condition
        print("\nProcessing 3-back condition...")
        X, y = prepare_ml_data_optimized(A3B, S3B, config)
        clf, scaler, accuracy, report = train_model_optimized(X, y, model_dir,
'3back', config)
        results['3back'] = {
            'accuracy': accuracy,
            'report': report,
            'classifier': clf,
            'scaler': scaler
        }

        print("\nAnalysis complete!")
        print("\n2-Back Results:")
        print(f"Accuracy: {results['2back']['accuracy']}")
        print("\n2-Back Classification Report:")
        print(results['2back']['report'])
        print("\n3-Back Results:")
        print(f"Accuracy: {results['3back']['accuracy']}")
        print("\n3-Back Classification Report:")
        print(results['3back']['report'])

        return results, model_dir

    except Exception as e:
        print(f"\nError in analysis pipeline: {str(e)}")
        raise

# Example usage
if __name__ == "__main__":
    try:
        # Custom configuration (optional)
        custom_config = PERFORMANCE_CONFIG.copy()
        custom_config.update({
            'batch_size': 400,      # Increased batch size
            'freq_step': 12,        # Larger frequency steps
            'decim': 16,            # Higher decimation
            'n_estimators': 25      # Fewer trees
        })

        # Run analysis
        results, model_dir = run_analysis_pipeline(
            A2B, A3B, S2B, S3B,
            config=custom_config
        )
```

```
        # Load saved models (example)
        print("\nLoading saved models for verification...")

        # Load 2-back model
        model_2back = load_model(model_dir, '2back')
        print("\n2-back model loaded successfully")
        print(f"Model type: {type(model_2back['classifier']).__name__}")
        print(f"Metadata: {json.dumps(model_2back['metadata'], indent=2)}")

        # Load 3-back model
        model_3back = load_model(model_dir, '3back')
        print("\n3-back model loaded successfully")
        print(f"Model type: {type(model_3back['classifier']).__name__}")
        print(f"Metadata: {json.dumps(model_3back['metadata'], indent=2)}")

        print("\nAnalysis pipeline completed successfully!")

    except Exception as e:
        print(f"\nError in main execution: {str(e)}")
        logging.exception("Detailed error information:")
        raise

    finally:
        # Cleanup
        gc.collect()
        print("\nCleanup completed")
```

8 Results And Key Findings

This study examined the effects of iTBS on working memory performance in healthy young adults through behavioral and EEG analysis.

Key Findings:

1. **Behavioral Performance:**
   a. The active iTBS group demonstrated a **significant improvement in accuracy**, outperforming the sham (placebo) group, which exhibited a decline in accuracy.
   b. Performance enhancements were **most pronounced at 10 minutes post-stimulation**, indicating a transient but notable cognitive boost.
   c. The **3-back task**, being more cognitively demanding, showed smaller improvements compared to the 2-back task across both groups, suggesting task complexity influences the degree of iTBS-induced enhancement.
   d. The observed effects of iTBS **gradually diminished by the 30-minute mark**, although performance remained above baseline levels, indicating a temporary but sustained benefit.
2. **EEG Signal Analysis:**
   a. We successfully classified EEG signals into different frequency bands, including **delta, theta, alpha, beta, and gamma waves**, based on their characteristic frequencies.
   b. Changes in EEG activity post-iTBS indicated enhanced neural oscillations, particularly in the **alpha and theta bands**, which are associated with cognitive processes such as attention and working memory.
   c. Correlation analysis revealed a positive association between EEG power changes and behavioral improvements, further supporting the neuromodulatory effects of iTBS on cognitive function.

In conclusion, our study demonstrates that a single session of iTBS applied to the left dorsolateral prefrontal cortex (DLPFC) can transiently enhance cognitive performance and modulate EEG activity related to working memory tasks. The active group showed clear benefits over the sham group, emphasizing the potential of non-invasive brain stimulation techniques for cognitive enhancement in healthy individuals.

9 REFERENCES

1. National Library Of Medicine - Electroencephalography Signal Processing
2. Science-Direct- Analysis of Electroencephalography (EEG) Signals and Its Categorization–A Study
   Website:
   https://www.sciencedirect.com/science/article/pii/S1877705812022114
3. Random Forest Classifier
   Website:
   https://www.youtube.com/watch?v=v6VJ2RO66Ag&ab_channel=NormalizedNerd
   https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.RandomForestClassifier.html
4. Overview of MEG/EEG analysis with MNE-Python
   Website:
   https://mne.tools/stable/auto_tutorials/intro/10_overview.html
5. Filtering EEG Data
   Website:
   https://neuraldatascience.io/7-eeg/erp_filtering.html
6. Analysis and simulation of brain signal data by EEG signal processing technique using MATLAB- Guru murthy.