Universidad Nacional Autónoma de México
Escuela Nacional de Estudios Superiores Unidad Juriquilla

Licenciatura en Neurociencias

Modelos Computacionales

Proyecto de análisis de datos :
Conjunto de datos de EEG para el TDAH

Profesores:
David Oswaldo Pérez Martinez
Fernando Javier Alcantara López

Alumna:
Martha Paola Álvarez Pérez

Juriquilla, Querétaro, 29 de noviembre de 2025

I. Introducción

El Trastorno por Déficit de Atención e Hiperactividad (TDAH) se ha asociado con alteraciones en la conectividad funcional del cerebro, especialmente en redes frontales, cinguladas y parietales. El análisis de conectividad mediante EEG permite estudiar cómo interactúan las diferentes regiones cerebrales.

En este proyecto se busca analizar las diferencias en los patrones de conectividad entre un grupo de niños con TDAH y un grupo control, utilizando un dataset de EEG público disponible en Kaggle. Participaron 61 niños con TDAH y 60 niños controles, con edades de 7 a 12 años. La evaluación se centró en la atención visual: se mostró a los niños un conjunto de imágenes de personajes de dibujos animados y se les pidió contar los caracteres presentes en cada imagen.

Los datos obtenidos se analizaron aplicando principios de la teoría de redes para comprender la organización de la actividad cerebral en cada grupo.

II. Objetivos

Objetivo General

Comparar la conectividad funcional cerebral entre niños con TDAH y controles usando EEG y métricas de teoría de redes.

Objetivos Específicos

A. Construir matrices de conectividad y resaltar las conexiones más significativas.
B. Representar la conectividad mediante grafos 2D y 3D, identificando hubs y comunidades.
C. Detectar diferencias en la organización cerebral entre los grupos y patrones característicos del TDAH.

III. Metodología

Se utilizó el programa Visual Studio con el lenguaje de Python. Primero se creó una carpeta llamada "Proyecto 1", en la cual se cargaron los datos utilizados y se generaron dos archivos: uno llamado conectividad.ipynb y otro llamado proyecto.ipynb. En el archivo conectividad.ipynb se realizó la carga del dataset, en este caso adhdata.csv, se importaron las librerías necesarias para el procesamiento numérico, la creación de grafos, la visualización y otras funciones, y se definieron las

bandas para calcular la conectividad específica, como por ejemplo la banda alfa (imagen 1 y 2).

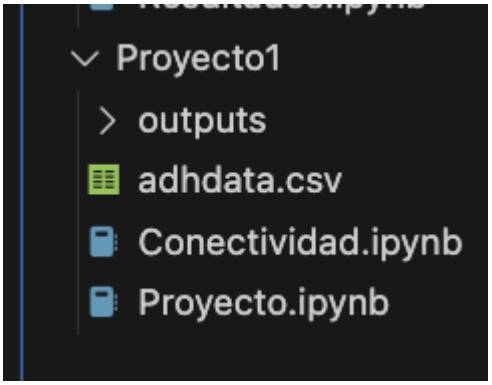


Imagen 1

```python
import os
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from scipy.signal import coherence, butter, filtfilt, hilbert
from itertools import combinations
from networkx.algorithms import community
import warnings
warnings.filterwarnings("ignore")


RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

# Bandas
BANDS = {
    "delta": (1, 4),
    "theta": (4, 8),
    "alpha": (8, 13),
    "beta": (13, 30),
    "gamma": (30, 45)
}
```

Imagen 2

Posteriormente, se emplearon funciones como ensure_dir(path), que crea una carpeta y garantiza que se almacenen de manera ordenada en cada ejecución, y load_csv_detect(data_path), que permite leer archivos y devolver una matriz de canales (imagen 3).

```python
# Crear carpeta
def ensure_dir(path):
    os.makedirs(path, exist_ok=True)


def load_csv_detect(data_path):
    df = pd.read_csv(data_path, index_col=None)

    # Caso 1: filas = canales
    if df.shape[0] < df.shape[1]:
        data = df.to_numpy(dtype=float)
        ch_names = list(df.index.astype(str))

        if df.dtypes[0] == object:
            try:
                names = df.iloc[:,0].astype(str).values
                data = df.iloc[:,1:].to_numpy(dtype=float)
                ch_names = list(names)
            except:
                ch_names = [f"Ch{i}" for i in range(data.shape[0])]
    else:
        # Caso 2: filas = muestras → transponer
        if all(np.issubdtype(dt, np.number) for dt in df.dtypes):
            data = df.to_numpy(dtype=float).T
            ch_names = list(df.columns.astype(str))
        else:
            data = df.select_dtypes(include=[np.number]).to_numpy(dtype=float).T
            ch_names = list(df.select_dtypes(include=[np.number]).columns.astype(str))

    return {"data": data, "ch_names": ch_names}
```

Imagen 3

Para el procesamiento de señales se utilizó la función bandpass(data, fs, low, high), que aplica un filtro Butterworth pasa-banda mediante filtfilt para eliminar el ruido sin generar distorsión de fase. Luego se calculó la conectividad utilizando compute_coherence_matrix(data_bp, fs, fmin, fmax), que calcula la coherencia espectral entre todos los pares de canales. Después se seleccionan solo las frecuencias de la banda de interés (fmin–fmax), se promedian los valores y se construye una matriz simétrica de tamaño canales x canales. Además, se utilizó la función compute_plv_matrix(data_bp) (imagen 4).

```python
def bandpass(data, fs, low, high, order=4):
    b, a = butter(order, [low/(fs/2), high/(fs/2)], btype='band')
    return filtfilt(b, a, data, axis=1)


def compute_coherence_matrix(data, fs, fmin, fmax, nperseg=None):
    n_channels = data.shape[0]
    nperseg = nperseg or int(fs*2)

    coh = np.zeros((n_channels, n_channels))

    for i in range(n_channels):
        for j in range(i, n_channels):
            f, Cxy = coherence(data[i], data[j], fs=fs, nperseg=nperseg)
            mask = (f >= fmin) & (f <= fmax)
            coh[i, j] = np.mean(Cxy[mask]) if mask.any() else np.mean(Cxy)
            coh[j, i] = coh[i, j]

    np.fill_diagonal(coh, 0)
    return coh


def compute_plv_matrix(data):
    analytic = hilbert(data, axis=1)
    phases = np.angle(analytic)
    n_channels = phases.shape[0]

    plv = np.zeros((n_channels, n_channels))

    for i in range(n_channels):
        for j in range(i+1, n_channels):
            phase_diff = phases[i] - phases[j]
            val = np.abs(np.mean(np.exp(1j*phase_diff)))
            plv[i,j] = val
            plv[j,i] = val

    np.fill_diagonal(plv, 0)
    return plv
```

Imagen 4

Para definir umbrales se empleó la función threshold_matrix(mat, percentile=75), que aplica un umbral basado en percentiles, calculando el valor correspondiente al percentil 75, manteniendo solo las conexiones mayores o iguales a este valor y generando matrices listas para el cálculo de métricas de teoría de redes. A partir de la matriz umbralizada se construye un grafo no dirigido mediante graph_metrics_from_adj y se calculan diversas métricas para su visualización. Finalmente, se utilizaron funciones como plot_heatmap, que genera mapas de calor, y plot_graph_2d, que genera un grafo en 2D (imagen 5 y 6).

```python
def threshold_matrix(mat, percentile=75):
    vals = mat.flatten()
    vals = vals[~np.isnan(vals)]
    th = np.percentile(vals, percentile)

    mat_bin = (mat >= th).astype(int)
    mat_thresh = mat.copy()
    mat_thresh[mat < th] = 0

    return mat_thresh, mat_bin, th


def graph_metrics_from_adj(adj):
    G = nx.from_numpy_array(adj)

    deg = dict(G.degree(weight='weight'))
    betw = nx.betweenness_centrality(G)
    clust = nx.clustering(G, weight='weight') if len(G) > 0 else {}

    try:
        comms = list(community.greedy_modularity_communities(G))
        modularity = nx.community.modularity(G, comms)
    except:
        comms, modularity = [], None

    return {
        "graph": G,
        "degree": deg,
        "betweenness": betw,
        "clustering": clust,
        "communities": comms,
        "modularity": modularity
    }
```

Imagen 5

```python
def threshold_matrix(mat, percentile=75):
    vals = mat.flatten()
    vals = vals[~np.isnan(vals)]
    th = np.percentile(vals, percentile)

    mat_bin = (mat >= th).astype(int)
    mat_thresh = mat.copy()
    mat_thresh[mat < th] = 0

    return mat_thresh, mat_bin, th


def graph_metrics_from_adj(adj):
    G = nx.from_numpy_array(adj)

    deg = dict(G.degree(weight='weight'))
    betw = nx.betweenness_centrality(G)
    clust = nx.clustering(G, weight='weight') if len(G) > 0 else {}

    try:
        comms = list(community.greedy_modularity_communities(G))
        modularity = nx.community.modularity(G, comms)
    except:
        comms, modularity = [], None

    return {
        "graph": G,
        "degree": deg,
        "betweenness": betw,
        "clustering": clust,
        "communities": comms,
        "modularity": modularity
    }
```

Imagen 6

El archivo proyecto.ipynb se utilizó como una segunda etapa del análisis, separada del conectividad.ipynb original, en la cual se trató de agrupar y comparar los resultados ya procesados. En el proyecto.ipynb se lograron cargar los resultados ya generados por el anterior archivo y que permitieran visualizar los datos almacenados en la carpeta outputs (Imagen 6)

```python
DATA_PATH = "adhdata.csv"
OUT_ROOT = "./outputs"
FS = 256.0
METHOD = "plv"
BAND = "alpha"
THRESH_PERCENTILE = 75
POP_FILTER_PCT = None
RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)

# canales
CHANNELS = ['Fp1','Fp2','F3','F4','C3','C4','P3','P4','O1','O2',
            'F7','F8','T7','T8','P7','P8','Fz','Cz','Pz']

BANDS = {
    "delta": (1, 4),
    "theta": (4, 8),
    "alpha": (8, 13),
    "beta": (13, 30),
    "gamma": (30, 45)
}
```

Imagen 6

El procesamiento individual de cada sujeto se realizó principalmente con la función process_ensemble, que toma los datos de un ID, aplica un filtro bandpass mediante bandpass para eliminar ruido sin distorsionar la fase, calcula la conectividad usando compute_plv_matrix o compute_coherence_matrix, y guarda las matrices de conectividad continua y umbralizada.(imagen 7)

```python
#procesamiento individual
def process_ensemble(df_group, group_id, outroot, fs=256.0, method="plv", band_key="alpha", threshold_percentile=75):
    """
    df_group: dataframe con filas = muestras para un mismo ID
    """
    ch_names = CHANNELS
    data = df_group[ch_names].to_numpy(dtype=float).T
    n_channels, n_samples = data.shape
    ensure_dir(outroot)
    fmin, fmax = BANDS[band_key]

    # 1) bandpass
    data_bp = bandpass(data, fs, fmin, fmax)

    # 2) conectividad
    if method == "coherence":
        conn = compute_coherence_matrix(data_bp, fs, fmin, fmax)
    elif method == "plv":
        conn = compute_plv_matrix(data_bp)
    else:
        raise ValueError("method must be 'coherence' or 'plv'")
```

Imagen 7

Las visualizaciones correspondientes se generaron con plot_heatmap para los mapas de calor y plot_graph_2d y plot_graph_3d para los grafos, utilizando tamaños de nodo proporcionales al grado de cada canal. Toda esta información se guarda en archivos CSV (imagen 8).

```python
#heatmap y matriz
plot_heatmap(conn, ch_names, f"Connectivity ({method}) {band_key} - {group_id}", os.path.join(outroot, "heatmap_continuous.png"))
pd.DataFrame(conn, index=ch_names, columns=ch_names).to_csv(os.path.join(outroot, "adjacency_continuous.csv"))

# 4) umbral por percentil
conn_thresh, conn_bin, th = threshold_matrix(conn, percentile=threshold_percentile)
plot_heatmap(conn_thresh, ch_names, f"Thresholded >= {th:.3f} ({threshold_percentile}p)", os.path.join(outroot, "heatmap_thresholded.png"))
plot_heatmap(conn_bin, ch_names, f"Binary adjacency ({threshold_percentile}p)", os.path.join(outroot, "heatmap_binary.png"))
pd.DataFrame(conn_thresh, index=ch_names, columns=ch_names).to_csv(os.path.join(outroot, "adjacency_thresholded.csv"))

# 5) grafos y métricas
metrics = graph_metrics_from_adj(conn_thresh)
G = metrics["graph"]
degree = metrics["degree"]
# node sizes proporcionales al grado
node_size_map = {i: (degree.get(i,0)+1)*80 for i in range(len(ch_names))}
plot_graph_2d(G, ch_names, pos=None, title=f"Graph 2D - {group_id}", outpath=os.path.join(outroot, "graph2d.png"), node_size_map=node_size_map)
plot_graph_3d(G, ch_names, title=f"Graph 3D - {group_id}", outpath=os.path.join(outroot, "graph3d.png"), node_size_map=node_size_map)
```

Imagen 8

La función run_pipeline_on_csv recorre todos los IDs del dataset, ejecutando process_ensemble por sujeto, almacenando las matrices binarias en una lista y generando un resumen de métricas en un CSV. La función population_filter permite obtener un grafo poblacional. (Imagen 9)

```
    return conn_thresh, conn_bin, metrics_out

def run_pipeline_on_csv(data_path, out_root, fs, method, band, thresh_pct, pop_filter_pct=None):
    df = pd.read_csv(data_path)
    missing = [c for c in CHANNELS + ['ID'] if c not in df.columns]
    if missing:
        raise ValueError(f"Faltan columnas esperadas en el CSV: {missing}")
    ensure_dir(out_root)

    summary = []
    binary_matrices = []
    ids = df['ID'].unique()
    print(f"Se encontraron {len(ids)} ID(s). Procesando cada ID como ensayo...")

    for gid in ids:
        df_g = df[df['ID'] == gid].reset_index(drop=True)
        outdir = os.path.join(out_root, str(gid))
        ensure_dir(outdir)
        print(f"Procesando ID = {gid} -> muestras: {len(df_g)}")
        conn_thresh, conn_bin, metrics = process_ensemble(df_g, gid, outdir, fs=fs, method=method, band_key=band, threshold_percentile=thresh_pct)
        summary.append(metrics)
        binary_matrices.append(conn_bin)

    # aplicar filtro poblacional
    if pop_filter_pct is not None:
        print("Aplicando filtro poblacional:", pop_filter_pct)
        pop_keep = population_filter(binary_matrices, pop_pct=pop_filter_pct)

        # matriz poblacional
        pd.DataFrame(pop_keep, index=CHANNELS, columns=CHANNELS).to_csv(os.path.join(out_root, "population_filtered_edges.csv"))
        Gpop = nx.from_numpy_array(pop_keep)
        nx.write_gml(Gpop, os.path.join(out_root, "population_graph.gml"))
```

Imagen 9

Por último, a partir de otros códigos similares se generaron las visualizaciones de hubs, mapas de calor, comunidades y grafos en 2D y 3D, lo que permite representar de manera gráfica la conectividad (imagen 10). Cabe aclarar que el código podría parecer un poco desordenado debido a la repetición de funciones; esto se debe a que la carga de los archivos requería mucho tiempo, y al intentar simplificarlo, el proceso se volvía aún más lento y propenso a trabarse.

```
#Mapas de calor

for group in ["ADHD", "Control"]:


    avg_mat = group_avg[group]


    thr_mat = group_thr[group]

    print("\n Heatmap continuo")
    plot_group_heatmap(avg_mat, group, "Continuous")

    print("\n Heatmap thresholded (percentil 75)")
    plot_group_heatmap(thr_mat, group, "Thresholded 75%

    print("\n Grafo 2D grupal")
    plot_group_graph_2d(thr_mat, group)

    print("\n Grafo 3D grupal")
    plot_group_graph_3d(thr_mat, group)

✓ 0.5s
```

Imagen 10

IV. Resultados

En el grupo control, se observó una conectividad más localizada y segregada, con módulos bien

definidos en las regiones frontal, parietal y occipital. Las conexiones fronto-parietales fueron moderadas pero consistentes, y el patrón mostró mayor simetría interhemisférica. En contraste, el grupo con TDAH presentó una conectividad más difusa, con mayor variabilidad entre regiones, conexiones menos organizadas, y una menor segregación modular reflejada en una coherencia más dispersa.(Imagen 11 y 12)

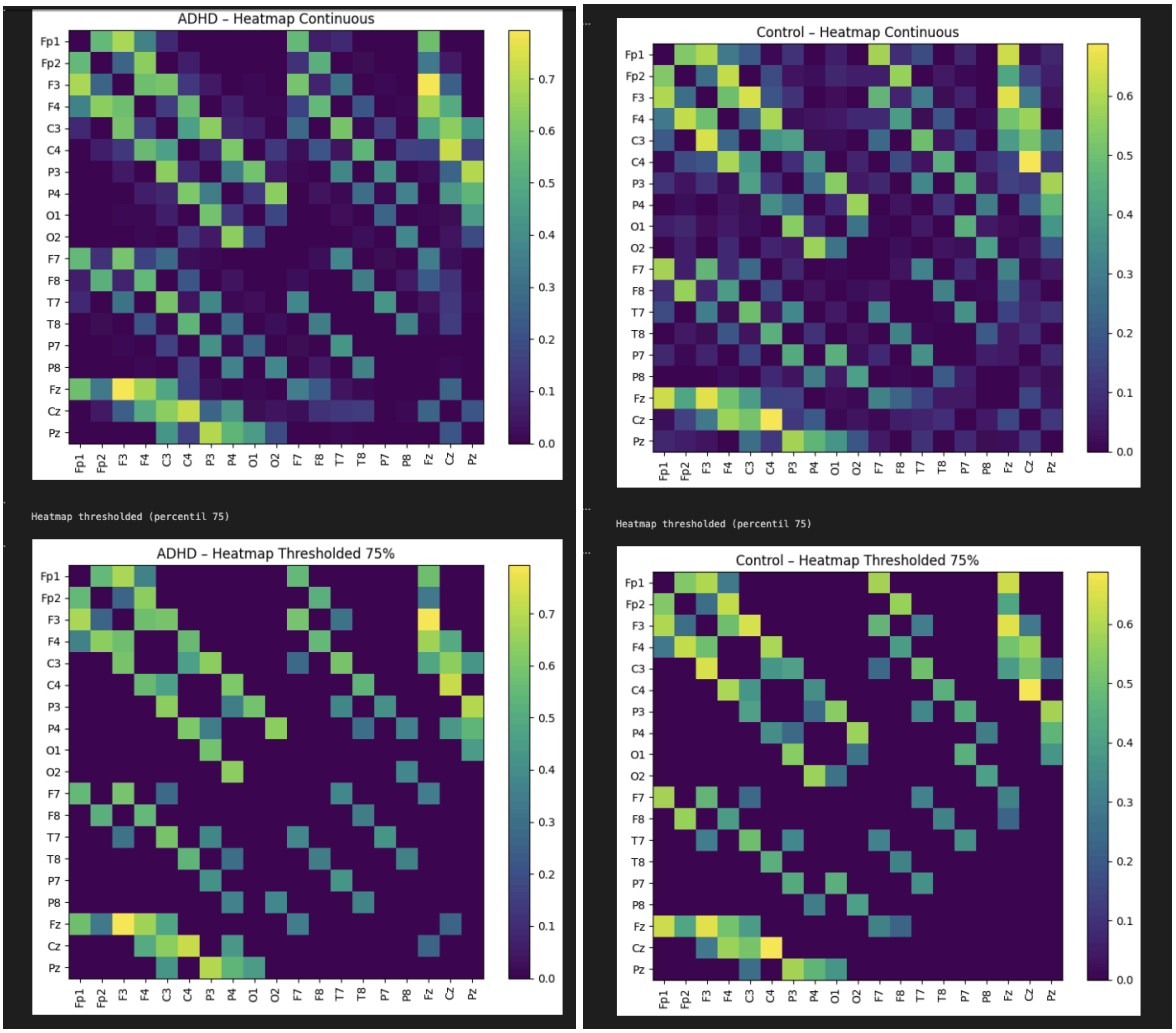
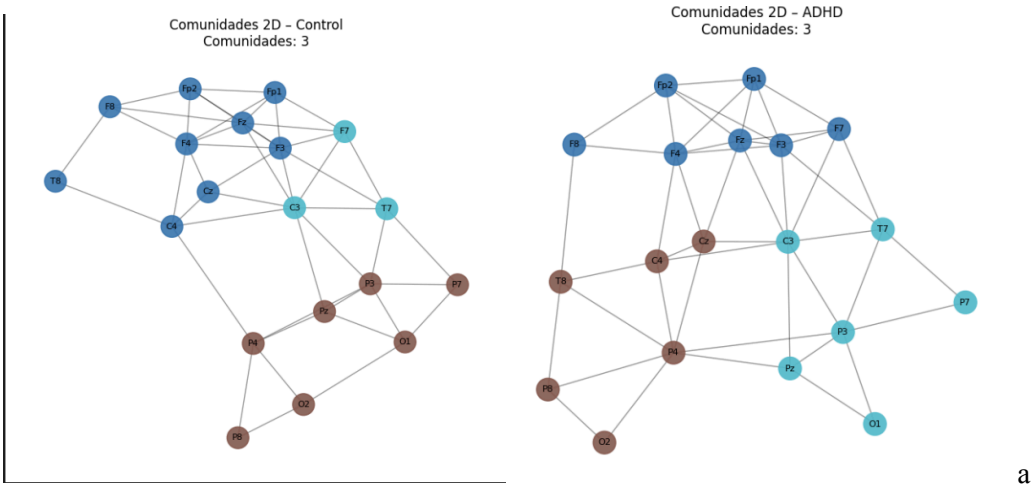


Imagen 11 y 12

El análisis de comunidades en los grafos de conectividad mostró diferencias claras entre los grupos Control y TDAH. En el grupo Control se observaron tres comunidades bien definidas, con una organización modular coherente con la neuroanatomía funcional típica. La comunidad frontal-centrada presentaba una estructura más cohesionada, reflejando una sincronización estable entre regiones prefrontales y centrales, mientras que la comunidad occipito-parietal se mantenía separada y organizada, característica de una actividad alfa posterior funcionalmente eficiente.

En contraste, el grupo TDAH también presentó tres comunidades, pero con límites menos definidos y mayor difusión entre regiones. Las redes frontales y centrales mostraron una menor segregación, con conexiones más mezcladas y una integración atípica de áreas temporales. De igual manera, la comunidad occipito-parietal perdió parte de su estructura característica, lo cual sugiere una organización alfa menos estable. Este patrón es consistente con la literatura del TDAH, donde se reporta una menor modularidad y mayor ruido funcional, reflejando dificultades en el control ejecutivo, la atención sostenida y el filtrado sensorial..(Imagen 13 y 14)



a

Imagen !3 y 14

En los grafos 3D se observa una diferencia clara entre los grupos. En el grupo TDAH, el hub principal es P4, una región parietal derecha. Esto indica que la red funcional depende de áreas más posteriores para integrar la información, un patrón menos típico y asociado a una organización menos eficiente del flujo cerebral. En contraste, en el grupo Control el hub es C3, una región central que normalmente actúa como punto de integración entre áreas frontales, parietales y occipitales. Este patrón es más estable y consistente con una red funcional organizada de manera óptima.(Imagen 15 y 16)

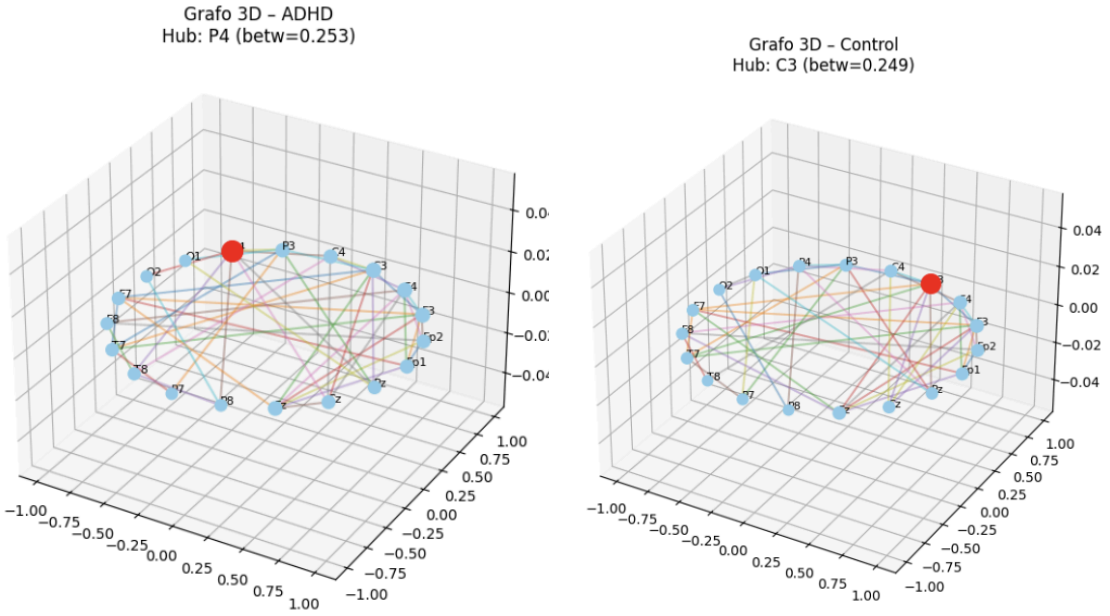


Imagen 15 y 16

V. Discusión

Los resultados muestran diferencias claras en la organización funcional cerebral entre controles y TDAH, centradas en regiones frontales, parietales y occipitales. En los controles, la conectividad localizada y modular refleja un funcionamiento eficiente: las áreas frontales facilitan control ejecutivo

y atención, las parietales integran información sensoriomotora, y la comunidad occipito-parietal soporta un procesamiento visual estable (Soman et al., 2023; Qian et al., 2018). El hub C3 actúa como nodo integrador entre estas regiones, favoreciendo un flujo cerebral organizado.
En el TDAH, la conectividad es más difusa, con hubs desplazados hacia la parietal derecha (P4) y menor segregación de comunidades. Las conexiones fronto-centrales desorganizadas y la pérdida de estructura occipito-parietal sugieren dificultades en funciones ejecutivas, atención sostenida e integración sensoriomotora y visual (Michael et al., 2025; Qian et al., 2018). Estos hallazgos coinciden con estudios previos que reportan menor modularidad, mayor ruido funcional y reorganización de hubs en TDAH. En conjunto, aunque ambos grupos muestran un número similar de comunidades, la arquitectura funcional es más especializada y coherente en controles, y más difusa y menos eficiente en TDAH, afectando la coordinación entre regiones clave para el procesamiento cognitivo y sensorial.

VI. Conclusión

El análisis de conectividad cerebral mediante EEG mostró diferencias claras entre niños con TDAH y controles. Mientras que el grupo control presentó redes funcionales más organizadas, modulares y simétricas, con hubs centrales que facilitan la integración entre regiones, el grupo TDAH mostró conectividad más difusa, menor segregación modular y hubs desplazados hacia áreas posteriores. Estos hallazgos reflejan una arquitectura funcional menos eficiente en TDAH, coherente con dificultades en atención, control ejecutivo e integración sensoriomotora, y respaldan la evidencia de alteraciones en la organización cerebral asociadas a este trastorno.

VII. Bibliografias

Danizo. (s. f.). EEG Dataset for ADHD [Database]. Kaggle. https://www.kaggle.com/datasets/danizo/eeg-dataset-for-adhd/data Kaggle+1

Michael, C., Mitchell, M. E., Cascone, A. D., Fogleman, N. D., Rosch, K. S., Cutts, S. A., Pekar, J. J., Sporns, O., Mostofsky, S. H., & Cohen, J. R. (2025). Reconfiguration of Functional Brain Network Organization and Dynamics With Changing Cognitive Demands in Children With Attention-Deficit/Hyperactivity Disorder. Biological psychiatry. Cognitive neuroscience and neuroimaging, 10(8), 846–855. https://doi.org/10.1016/j.bpsc.2024.11.006
Soman, S. M., Vijayakumar, N., Thomson, P., Ball, G., Hyde, C., & Silk, T. J. (2023). Functional and structural brain network development in children with attention deficit hyperactivity disorder. Human brain mapping, 44(8), 3394–3409. https://doi.org/10.1002/hbm.26288

Qian, X., Castellanos, F. X., Uddin, L. Q., Loo, B. R. Y., Liu, S., Koh, H. L., Poh, X. W. W., Fung, D., Guan, C., Lee, T. S., Lim, C. G., & Zhou, J. (2019). Large-scale brain functional network topology disruptions underlie symptom heterogeneity in children with attention-deficit/hyperactivity disorder. NeuroImage. Clinical, 21, 101600. https://doi.org/10.1016/j.nicl.2018.11.010