A Project Report

On

# Optimizing Memory Usage Of Graph Neural Networks

BY

**MUGDHA GUPTA**

**(2021B3A72724H)**

Under the supervision of

**PROF. SAMEERA MUHAMED SALAM**

**SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF**

**CS F376: DESIGN PROJECT**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)**

**HYDERABAD CAMPUS**

**(DECEMBER 2024)**

# ACKNOWLEDGMENTS

My sincere appreciation goes out to my mentor, Professor Sameera Muhamed Salam of the BITS Pilani, Hyderabad Campus's Department of Computer Science. Her constant assistance and perceptive oversight were invaluable to me during the "Optimising Memory Usage of Graph Neural Networks" research. Her support and guidance on relevant research topics and techniques increased the project's effectiveness and provided me with first-hand research experience.

I also want to express my gratitude to BITS Pilani Hyderabad Campus for making it possible for me to take part in this study as part of my academic requirements. My understanding of graph theory and neural networks has improved as a result of applying my academic knowledge in a practical context.

**Birla Institute of Technology and Science-Pilani,**

**Hyderabad Campus**

## Certificate

This is to certify that the project report entitled "Optimizing Memory Usage of Graph Neural Networks**"** submitted by Ms. MUGDHA GUPTA (ID No. 2021B3A72724H) in partial fulfillment of the requirements of the course CS F376, Design Project Course, embodies the work done by him under my supervision and guidance.

**Date: 28th November 2024** **(Prof. Sameera Muhamed Salam )**

BITS- Pilani, Hyderabad Campus

## ABSTRACT

This project aims to use a novel framework for classifying human state anxiety using Graph Neural Networks (GNNs) based on the EEG signals recorded. EEG channels are represented as nodes, with connections between channels as edges, allowing GNNs to process the data as a graph. Several features from the EEG signals are extracted. These features serve as node attributes in the graph, which are aggregated through graph convolutional layers to capture relationships between channels. Graph pooling layers summarize these features into a graph-level representation for classifying anxiety into four categories: Normal, Light, Moderate, and Severe. We wish to continue this work by optimizing memory usage while implementing graph neural networks to avoid frequent out-of-memory (OOM) problems and poor inference efficiency.

# CONTENTS

# Introduction

Graph Neural Networks (GNNs) are widely used in a variety of fields, such as biology, social networks, and knowledge graphs. Notwithstanding these achievements, the continuously growing quantity and complexity of graph data pose significant scalability challenges for GNNs. Two primary difficulties have been identified:

1. **Memory Requirement:** GNN models typically need to store entire adjacency and feature matrices, which makes managing large graphs challenging and memory-intensive.
2. **Ineffective Gradient Updates**: For large graphs, it is computationally unfeasible to recursively update each node during GNN training, leading to ineffective training.

Given a graph G (V, E ), |V | = n denotes the set of n nodes, and |E | = m denotes a set of m edges. Node u ∈ N (v) is the neighborhood of node v, where v ∈ V, and (u, v) ∈ E.
The basic GNN architecture can be summarized as:

$$\mathbf{h}^{(l+1)} = \sigma\left(A\mathbf{h}^{(l)}W^{(l)}\right)$$

Anxiety is a common mental health challenge that many people face, often marked by feelings of unease, fear, or worry. It usually shows up in stressful situations like speaking in public, taking exams, or undergoing medical procedures. While a little anxiety can be a natural response, helping us stay alert, ongoing anxiety can lead to serious problems in both mental and physical health. Anxiety and stress are used interchangeably the same. Stress comes from dealing with specific external challenges, but anxiety can linger even after those challenges are resolved. If left unchecked, long-term anxiety and stress can lead to issues like depression, sleep problems, and even heart disease.

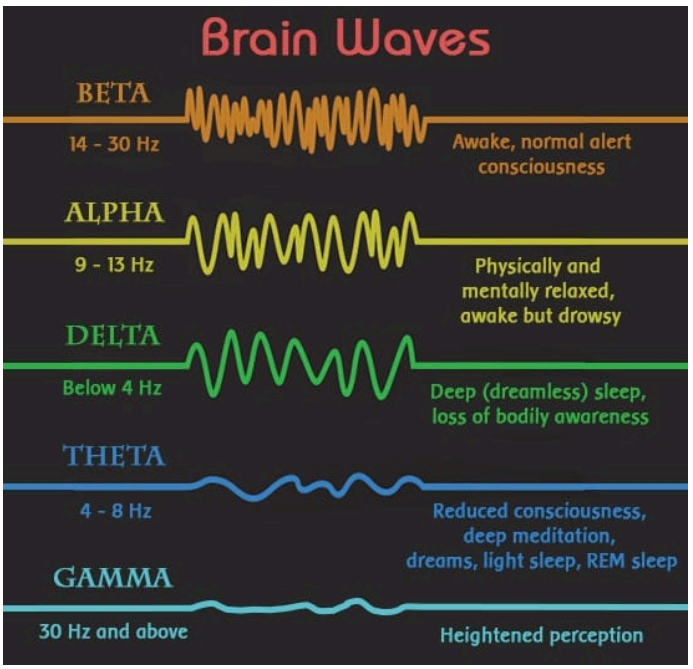
Anxiety disorders are common and impact a large portion of the global population. For example, less than 40% of individuals who suffer from anxiety disorders receive the necessary treatment, despite the fact that approximately 18% of them do. Hospitalization and long-term health issues, such as heart disease and stroke, are associated with anxiety. Additionally, it is closely related to depression and frequently exacerbates both illnesses. There are various types of anxiety disorders, including post-traumatic stress disorder, obsessive-compulsive disorder (OCD), social anxiety, panic disorder, and generalized anxiety disorder. Certain anxiety disorders are more common in women, and anxiety frequently causes social and academic difficulties in kids and teenagers.

The State-Trait Anxiety Inventory and Hamilton Anxiety Scale are two examples of questionnaires that are used in current ways of diagnosing anxiety to get people to describe their symptoms. Despite their usefulness, these don't give a clear picture of the bodily and mental processes at play. With the use of instruments like skin response sensors (GSR), cardiac monitoring (ECG), and brain activity scans (EEG), researchers have recently begun investigating novel approaches to quantify anxiety more accurately.

This project focuses on using EEG, which measures brain activity, to create a more accurate way of identifying anxiety. EEG (Electroencephalography) signals are electrical activities of the brain recorded using electrodes placed on the scalp. These signals represent the brain's spontaneous electrical activity over a period, typically measured in microvolts (μV). EEG signals are generated by the synchronous activity of neurons in the brain, particularly the cortical neurons. Electrodes placed on the scalp detect these electrical activities.

EEG signals are categorized into different frequency bands, each associated with different brain states:

- Delta (0.5-4 Hz): Deep sleep, unconscious states.
- Theta (4-8 Hz): Light sleep, relaxation, creativity.
- Alpha (8-13 Hz): Relaxed wakefulness, calmness.
- Beta (13-30 Hz): Active thinking, focus, anxiety.
- Gamma (30-100 Hz): High-level information processing, and cognitive functioning.




EEG signals are a potent tool in clinical and research contexts, helping to diagnose and treat a

variety of neurological and psychological problems while offering important insights into brain activity. They are essential in medical diagnostics and neuroscience due to their great temporal resolution and non-invasive nature.

By displaying EEG data as a graph, collecting pertinent information, and employing graph convolutional layers to discover the connections between EEG channels, the GNN framework categorizes anxiety.

We aim to gain a deeper understanding of the distinctions between anxious and non-anxious states by integrating graph neural networks with EEG data. This strategy might result in more accurate methods of diagnosing anxiety and, eventually, better treatment options.

## Optimizing Memory in GNNs

### Sampling:

The purpose of sampling is to select a subset of all samples that will accurately reflect the distribution of all samples. Therefore, the sampling algorithm for huge networks presents a technique to solve target problems using a fraction graph instead of the full graph.

There are three types of sampling algorithms for big graphs: layer-wise sampling, graph-wise sampling, and node-wise sampling. With node-wise sampling, like GraphSAGE, nodes are chosen according to the target node. Sub-graphs are sampled for model inference in graph-wise samplings, such as GraphSAINT, which increases scalability and efficiency.

Ensuring sampling quality and creating effective sampling algorithms are crucial challenges in building large-scale GNNs. Addressing these problems has been the main focus of recent research to facilitate the efficient building of large-scale GNNs.

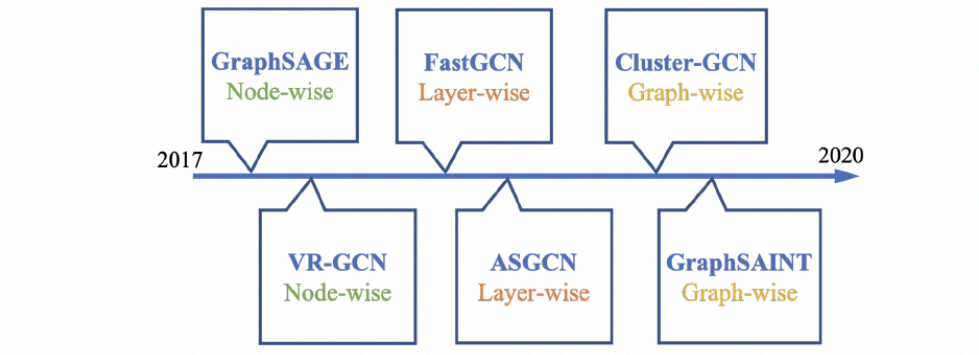



Fig. 6.2: Timeline of leading research work toward large-scale GNNs.

## WholeGraph

WholeGraph provides storage abstractions that enable multiple memories, such as pinned host and device memory and have great performance and large capacity. Additionally, WholeGraph efficiently distributes training operations among multiple GPUs, enhancing memory optimization and scalability for extensive graph datasets. It accomplishes this by being compatible with torch DistributedDataParallel mode and natively supports PyTorch.

WholeGraph optimizes memory for Graph Neural Networks (GNNs) by utilizing a versatile storage system called WholeMemory, which supports a variety of memory formats, including GPU and pinned host memory. These modes optimize memory access patterns, reduce memory overhead, and provide better scalability by dividing data among GPUs. WholeGraph also makes caching easier and easily integrates with deep learning frameworks for efficient large-scale GNN training.

# Methodology

## Building a Graph Convolution Network Model:

### Cora Dataset:

One popular benchmark dataset for tasks like node classification in Graph Neural Networks (GNNs) is the Cora dataset. Its nodes are scientific papers, and its edges are the citations that connect them. The characteristics of each node (paper) are represented by a feature vector, and the nodes are categorized according to their subject matter.

### Feature Vectors (Input Features) in the Cora Dataset:

- A bag-of-words feature vector describes each paper (node).
- Each member of the 1,433 binary features (input features) in the feature vector indicates whether or not a certain word from a predetermined dictionary appears in the document.
    - Every feature has a binary value (0 or 1) that indicates whether a given word appears in the paper or not.

As a result, every node in the graph has a feature vector with 1,433 dimensions, each of which represents the frequency of a particular word in the paper.

### Output Features (Classification Labels):

- Classifying each node (paper) into one of several categories is the aim of the Cora dataset's node classification design.
- Different academic topics or subject areas are represented by seven categories (classes):
    1. Case-Based
    2. Genetic Algorithms
    3. Neural Networks
    4. Probabilistic Methods
    5. Reinforcement Learning
    6. Rule Learning
    7. Theory

The aim is to predict the correct class label for each node (paper) based on its feature vector and the graph's structure. Each node is assigned one of these seven classes according to its topic area.

## Classification of unknown nodes

In a graph neural network (GNN) such as the one trained on the Cora dataset, classifying unknown nodes (i.e., nodes not encountered during training) entails utilizing the graph structure and node attributes of the unknown node and its neighbors.

In the context of GNNs, it functions as follows:

**1. Feature Propagation through Neighbors:**

Information from neighboring nodes is aggregated via graph neural networks, including Graph Convolutional Networks (GCNs). The GNN will utilize the node's known features and those of its neighbors to forecast the class when classifying an unknown node—that is, a node that was not labeled during training.

- Every node in a GNN aggregates features from its immediate neighbors (and possibly farther neighbors if the network is deeper) to update its representation (embedding).
- An unknown node's representation is impacted by the representations of its neighbors, which may have significant structural and feature similarities or may have previously been labeled during training.

**2. Node Feature Embedding:**

- The feature vector of the unknown node is present.
- This feature vector is combined with the feature vectors of the node's neighbors by the GNN to generate an embedding for the node. As the network's layers process this embedding, each layer aggregates further data from its neighbors.

**3. Aggregation of Neighbor Information:**

- The network learns a representation for every node during the GCN's forward pass by fusing its features with those of its neighbors. The GNN's layers go through this process iteratively.
- The GNN may predict the label even if the label of the unknown node is not provided during training by using the embeddings of its neighboring nodes, which may have labels.

**4. Classification of Unknown Nodes:**

The final layer usually produces logits or probabilities corresponding to each class once the GNN has learned the embeddings for all nodes (even unknown ones) during training. Based on its final embedding, the model forecasts the unknown node's class.

- The model can estimate its class because of the feature vector of the unknown node and the knowledge about its neighbors.
- The class with the highest probability is selected as the prediction for the unknown node after the GNN generates probabilities for each of the seven classes (in the instance of the Cora dataset).

**5. Generalization to Unknown Nodes:**

Among the structural and feature-based information that GNNs extract from the graph are the connections between nodes and the traits of neighboring nodes.
This approach is effective when a graph is continuously updated with new data points (nodes) and its labels must be predicted without retraining the entire model.

## Example using Cora:

When a new research paper is added to the citation network, the GNN can forecast its category based on the categories of the papers it cites (its neighbors in the graph) and the terms in its abstract (its feature vector).

## Converting  a dataset to a DataFrame

```
import pandas as pd  # Import pandas for handling DataFrames
import os  # Import os for handling file paths

# Assuming data_dir is a variable containing the directory path where the file is stored
citations_data = pd.read_csv(
    os.path.join(data_dir, "cora.cites"),  # Combine data directory and file name to get the full file path
    sep="\t",  # Specify the separator, which in this case is a tab character
    header=None,  # Since the file does not have a header, set this to None
    names=["target", "source"],  # Provide column names for the DataFrame
)

# Now 'citations_data' is a DataFrame with two columns: 'target' and 'source'.
print(citations_data.head())  # Print the first few rows of the DataFrame
```

## Label Encoding

Many machine learning methods require the conversion of categorical data (such as paper IDs and subject names) into numerical labels, which is done through label encoding.

- Data manipulation for citations: The data citations The numeric labels created before are used to replace the source and target paper IDs in the DataFrame, which stores citation relationships.

**Purpose of Label Encoding:**

- When transforming categorical data into a numerical format for usage by machine learning models, label encoding is frequently utilized. The code prepares these columns for use in algorithms that need numerical inputs by converting paper IDs and subjects into numeric values.

**Visualizing Data**

- Edges: Citations are represented by the edges. Each edge illustrates the citation relationship between two nodes.

- Node Colours: To make it simple to visually identify various research fields or themes within the dataset, each node (article) is colored according to its subject.
- The spring layout is a force-directed method that arranges nodes so that non-connected nodes are pushed apart and connected nodes are drawn closer together. This frequently produces a layout that shows groups of linked documents.

## Making graph data

1. Node features: This component shows how many nodes and features there are in an array.

2. Edges: The number of edges in both dimensions is represented by a sparse matrix of links between the nodes.

3. Edge weights: This is an array-based optional element. A quantification of the number of edges between nodes is represented by the values under this array.

### For Nodes

```
import tensorflow as tf feature_names = set(papers_data.columns) - {"paper_id", "subject"} feature_names
'''Convert the selected features into a NumPy array. The resulting array will have a shape of (number_of_papers, number_of_features)
where each row corresponds to the features of a paper (node).
Convert the NumPy array into a TensorFlow tensor with a specific data type.
Each row corresponds to a paper , and each column represents a feature of that paper (e.g., citation count, year, etc.).'''
```

### For Edge Weights

Every edge (citation) in the graph is given a uniform weight of 1 by the edge_weights tensor. Later on, graph models can adjust the information flow between nodes during training by utilising edge weights. These weights can be changed to reflect a relationship's strength.

## Graph info Tuple

```
graph_info = (node_features, edges, edge_weights)
'''node_features: A tensor with the features of each paper (node) in the graph.
edges: A NumPy array containing the source and target nodes of each edge (citation relationship).
edge_weights: A tensor containing the weights for each edge, in this case, all set to 1.
'''
```

To complete the challenges, a graph convolutional layer with prepare, aggregate, and update functions is constructed as a Keras layer.

## Graph neural node classifier

We must create a graph neural node classifier after creating the layer. The following process can be used by this classifier:

- To create the node representation, the node features must be preprocessed.
- Using graph layers.

- Final node representations are produced by post-processing the node representation.
- Generating the predictions based on the node representation using a softmax layer.

## Instantiating and fitting a GNN

Specify the learning rate, dropout rate, number of epochs, batch size, hidden units, and other hyperparameters that are used to configure the GNN model.

```
hidden_units = [32, 32]
learning_rate = 0.01
dropout_rate = 0.5
num_epochs = 300
batch_size = 256
gnn_model = GNNNodeClassifier(graph_info=graph_info, num_classes=num_classes, hidden_units=hidden_units,
                              dropout_rate=dropout_rate, name="gnn_model", )
gnn_model.summary()
```

```
Model: "gnn_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 preprocess (Sequential)     (2708, 32)                52804

 graph_conv1 (GraphConvLayer  multiple                 5888
 )

 graph_conv2 (GraphConvLayer  multiple                 5888
 )

 postprocess (Sequential)    (2708, 32)                2368

 logits (Dense)              multiple                  231

=================================================================
Total params: 67,179
Trainable params: 63,481
Non-trainable params: 3,698
_________________________________________________________________
```

## Fitting a GNN Model for Node Classification

The steps involved in building and fitting a GNN model for node categorization are as follows:

1. The node IDs and the associated subject labels define the training data (x_train and y_train).
2. For multi-class classification tasks, the run_experiment function builds the model with a sparse categorical cross-entropy loss function and an Adam optimizer.
3. The purpose of early stopping is to avoid overfitting.

4. With a 15% validation split, the model is trained using the model.fit() method. The procedure keeps going until either early stopping is initiated or the maximum number of epochs is achieved.
5. The training history is sent back for additional examination.

## Fitting the Model

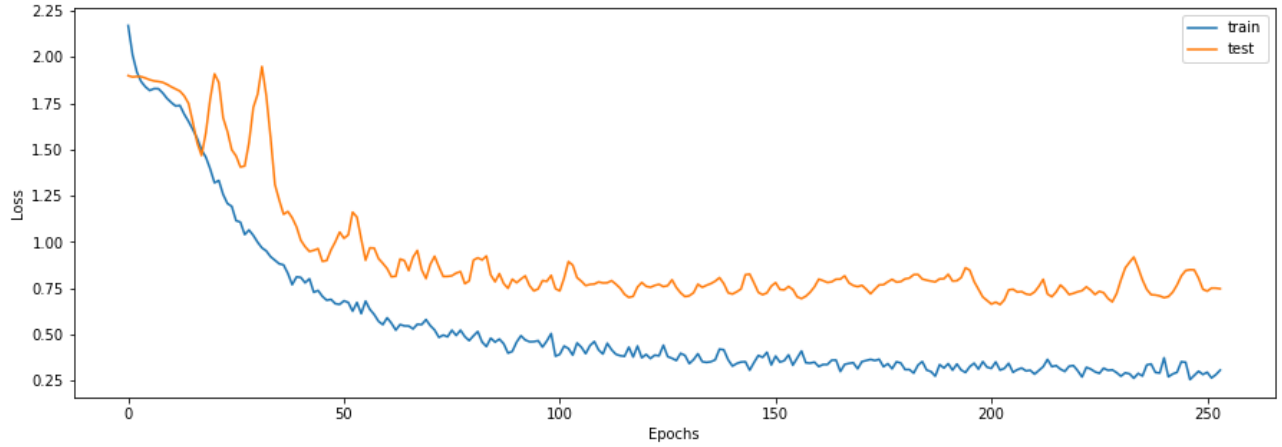
```
def run_experiment(model, x_train, y_train): # Compile the model.
model.compile( optimizer=keras.optimizers.Adam(learning_rate),
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")], ) # Create an early stopping callback.
               early_stopping = keras.callbacks.EarlyStopping( monitor="val_acc", patience=50, restore_best_weights=True )
               # Fit the model.
               history = model.fit( x=x_train, y=y_train, epochs=num_epochs,
                                   batch_size=batch_size, validation_split=0.15, callbacks=[early_stopping], )
               return history
```

## Output

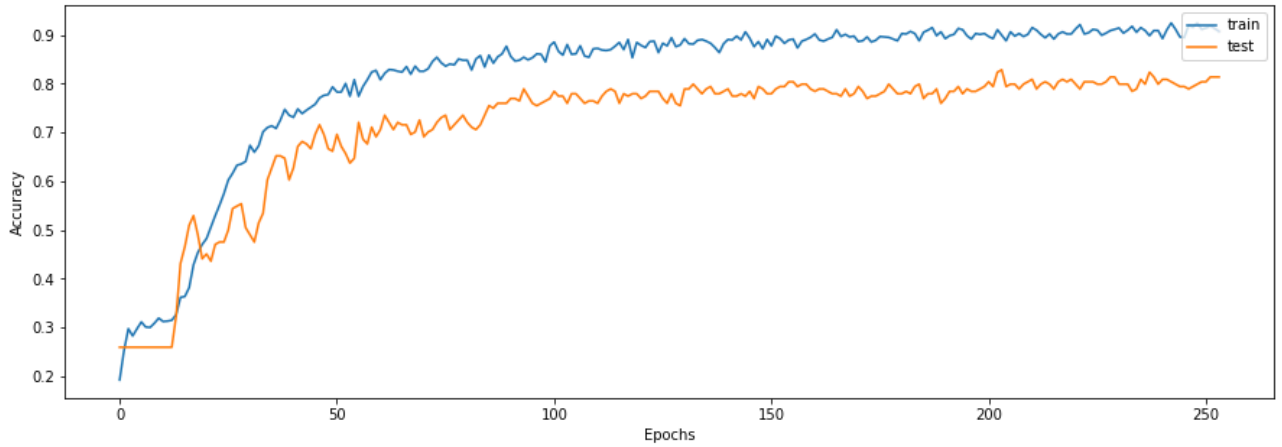
```
Epoch 1/300
5/5 [==============================] - 6s 356ms/step - loss: 2.1695 - acc: 0.1927 - val_loss: 1.8986 -
Epoch 2/300
5/5 [==============================] - 1s 200ms/step - loss: 2.0133 - acc: 0.2526 - val_loss: 1.8919 -
Epoch 3/300
5/5 [==============================] - 1s 191ms/step - loss: 1.9197 - acc: 0.2977 - val_loss: 1.8941 -
Epoch 4/300
5/5 [==============================] - 1s 183ms/step - loss: 1.8668 - acc: 0.2830 - val_loss: 1.8935 -
Epoch 5/300
5/5 [==============================] - 1s 183ms/step - loss: 1.8382 - acc: 0.2977 - val_loss: 1.8856 -
Epoch 6/300
5/5 [==============================] - 1s 183ms/step - loss: 1.8188 - acc: 0.3116 - val_loss: 1.8766 -
Epoch 7/300
5/5 [==============================] - 1s 178ms/step - loss: 1.8285 - acc: 0.3012 - val_loss: 1.8701 -
Epoch 8/300
5/5 [==============================] - 1s 195ms/step - loss: 1.8282 - acc: 0.3003 - val_loss: 1.8671 -
Epoch 9/300
5/5 [==============================] - 1s 193ms/step - loss: 1.8053 - acc: 0.3090 - val_loss: 1.8624 -
Epoch 10/300
5/5 [==============================] - 1s 189ms/step - loss: 1.7757 - acc: 0.3194 - val_loss: 1.8516 -
Epoch 11/300
```

## Visualizing the results

Epochs vs Loss




Epochs vs Accuracy




With an accuracy of almost 90% on the training set and 80% on the test data, we can observe that the model has done well.

# Implementation of GNN for Human Anxiety Classification using EEG signals

## Feature Extraction from EEG Signals

### 1. Pre-processing

To clean the EEG signals and get rid of any artifacts that may bias the results, pre-processing is essential. Electrical noise, muscular movements, and eye blinks are examples of common artifacts.

- **Bandpass Filtering**: This method eliminates noise and artifacts by filtering away frequencies that are beyond the target range, such as 0.5-45 Hz.
- **Independent Components Analysis (ICA)**: By dividing the EEG signal into discrete parts, ICA makes it possible to detect and eliminate artifacts such as muscular contractions and eye blinks.

### 2. Analysis of Frequency Domains

- To examine the power distribution across various frequency bands, frequency domain analysis entails converting the time-domain EEG signals into the frequency domain.
- **PSD, or power spectrum density**: Methods such as the Welch method are used to calculate PSD. To do this, the EEG signal is divided into overlapping windows, a window function (such as Hamming) is applied, the Fast Fourier Transform (FFT) is calculated for each segment, and the power of the FFT coefficients is averaged.

### 3. Feature Extraction

From the frequency domain analysis, we extract relevant features that can be used for classification.

#### 1. Mean Power (MP)

- **Definition**: The average EEG signal power in a particular frequency band, such as delta (2–4 Hz), theta (4–7 Hz), alpha (8–12 Hz), beta (13–30 Hz), or gamma (30–45 Hz), is known as mean power (MP). The strength or amplitude of the brainwave impulses in a particular band is referred to as power in this context.

#### 2. Rational Asymmetry (RASM)

- **Definition**: Rational Asymmetry (RASM) measures the ratio of the power spectra between asymmetric EEG channels on the left and right hemispheres of the brain. It is defined as the ratio between the power of corresponding electrode pairs on the two hemispheres.

$$RASM = \frac{Power_{RightHemisphere}}{Power_{LeftHemisphere}}$$

**3. Asymmetry Index (AI)**

- **Definition**: The power differential between symmetric EEG channels—that is, pairs of electrodes positioned symmetrically on the left and right hemispheres—is measured by the Asymmetry Index (AI).

$$AI = Power_{RightHemisphere} - Power_{LeftHemisphere}$$

## Graph Representation of EEG Data

The EEG signals are represented as a graph where nodes represent the EEG channels and edges represent the connections between those channels. A richer and more organized examination of EEG data is made possible by this graph-based model, which captures the spatial (physical) and functional (signal-based) interactions between various brain areas.

## 1. Nodes in the Graph

An electrode is placed on the scalp to record brain activity, and each node in the graph represents a single EEG channel. For example, if 14 EEG channels are used, the graph will have 14 nodes. Every node is linked to specific features, including the extracted EEG properties (Mean Power, Rational Asymmetry, Asymmetry Index) from various frequency bands (delta, theta, alpha, beta, and gamma).

There are 14 channels (such as AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4), each of these channels will be represented as a node. **Input Features (18 per Node)**

Each EEG channel (node) has **18 features**, derived from the EEG signals. These features are extracted from various frequency bands (delta, theta, alpha, beta, gamma) and represent key metrics such as **Mean Power (MP)**, **Rational Asymmetry (RASM)**, and **Asymmetry Index (AI)**.

Here's a simplified example of how the features might be structured for a single node:

```
# Example feature vector for a single EEG channel (node)
node_features = [
    mp_delta, mp_theta, mp_alpha, mp_beta, mp_gamma,  # Mean Power
    rasm_delta, rasm_theta, rasm_alpha, rasm_beta, rasm_gamma,  # Rational
Asymmetry
    ai_delta, ai_theta, ai_alpha, ai_beta, ai_gamma,  # Asymmetry Index
    std_theta, std_beta, entropy_alpha  # Additional features
]

# Example of a node feature vector
node_features = [
    0.5, 0.8, 0.6, 0.7, 0.4,  # Mean Power values
    1.2, 1.1, 1.3, 1.0, 1.4,  # Rational Asymmetry values
    0.2, 0.3, 0.4, 0.5, 0.6,  # Asymmetry Index values
    0.1, 0.2, 0.3  # Standard Deviation and Entropy values
]
```

## 2. Edges in the Graph

The graph's edges show the relationships between nodes, in this example, between EEG channels. The purpose of these linkages is to show how the brain activity captured by various electrodes relates to one another. The EEG data is typically represented as a graph using one of two primary types of connections (or edges):

**A. Edges Based on Physical Proximity:**

The physical proximity of the EEG electrodes on the scalp determines the borders of this kind of connection. EEG electrodes are placed in particular areas of the scalp according to a predetermined pattern. In the graph, edges connect electrodes that are physically adjacent to one another, such as neighbors.

For example, if electrodes F7 and F3 are close on the scalp, an edge will exist between the corresponding nodes in the graph.

**B. Edges Based on Functional Connectivity:**

The correlation between EEG signals collected at several electrodes, independent of their physical closeness, is known as functional connectivity. The way that different parts of the brain coordinate and communicate with one another during particular mental states, including anxiety, is known as functional connectivity.

The coherence or correlation between the EEG signals from various nodes (electrodes) is used to create edges. An edge forms between the relevant nodes in the graph when there is a high correlation between the signals from two electrodes, indicating a strong functional relationship.

**Approaches to Assess Functional Connectivity:**

**Correlation**: The linear relationship between two EEG channels' time-series data is measured using correlation. Two electrodes may be functionally coupled if their signals exhibit a high degree of correlation.

**Coherence**: Indicates how consistently two signals' phase differences occur over a certain frequency range. A stronger functional connection is indicated by higher coherence.

**An example of edges based on functional connectivity:**

Even though these nodes are not physically adjacent to one another, an edge will be formed between them if the signals from F3 (left frontal) and F4 (right frontal) are highly linked.

Likewise, if the occipital lobe electrode O1 and the temporal lobe electrode T7 exhibit strong coherence, an edge will link them, signifying their functional connection.

**Graph Example:**

Suppose there are 14 EEG channels, and we want to model this as a graph for GNN-based classification:

**Nodes**: Each of the 14 channels (AF3, F7, F3, etc.) is a node.

**Edges**:

Some edges are based on physical proximity, such as connecting AF3 ↔ F7 because they are close to each other on the scalp.

Other edges are based on functional connectivity, such as connecting F3 ↔ O1 if these electrodes have highly correlated signals, even though they are far apart on the scalp.

## Graph Neural Network (GNN) Model

### 1. Graph Construction

- Input: The EEG data is represented as a graph ( G = (V, E) ), where ( V ) is the set of nodes (EEG channels) and ( E ) is the set of edges (connections between channels).
- Node Features: Each node has a feature vector containing the extracted EEG features.

**2. GNN Layers**

- **Graph Convolutional Layers**: The node features are subjected to graph convolutions by the three GCN layers (conv1, conv2, and conv3). Every convolution updates each node's feature representation by combining data from nearby nodes (depending on the edges).
- **Global Pooling**: To provide a graph-level feature representation that summarises brain activity across all EEG channels, the global_mean_pool layer aggregates the node-level features.
- **Fully Connected Layer**: The graph-level representation is mapped to one of the four anxiety levels by the last fully connected (FC) layer.
- **Functions of Activation**: To add non-linearity, the Rectified Linear Unit (ReLU) is applied after every graph convolution layer. The log probabilities for each anxiety level are obtained by running the model's output through a log_softmax function.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, global_mean_pool

class AnxietyGNN(nn.Module):
    def __init__(self, num_node_features, num_classes):
        super(AnxietyGNN, self).__init__()
        self.conv1 = GCNConv(num_node_features, 64)
        self.conv2 = GCNConv(64, 128)
        self.conv3 = GCNConv(128, 64)
        self.fc = nn.Linear(64, num_classes)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))
        x = F.relu(self.conv3(x, edge_index))
        x = global_mean_pool(x, batch)  # Pooling over the graph
        x = self.fc(x)
        return F.log_softmax(x, dim=1)

# Example usage
# Assuming `data` is a PyTorch Geometric Data object with EEG graph data
model = AnxietyGNN(num_node_features=18, num_classes=4)  # 18 features per
node, 4 anxiety levels
output = model(data)
```

## Output of the GNN model

Based on clinical or self-reported indicators, the model's final output divides anxiety into four levels:

**Normal Anxiety**: The term "normal anxiety" refers to a baseline state in which a person has little or no anxiety symptoms.

**Light Anxiety:** This level denotes a light form of anxiety in which symptoms are noticeable but do not substantially disrupt day-to-day functioning. It could show up as mild anxiety or uneasiness.

**Moderate Anxiety**: Symptoms of anxiety are more noticeable and may interfere with day-to-day activities. People could have bodily symptoms like restlessness or a more persistent feeling of concern.

**Severe anxiety**: The most intense type of anxiety is called severe anxiety, and it manifests as overpowering symptoms that interfere with day-to-day functioning. People may suffer from intense anxiety, panic episodes, or difficulty focusing.

# CONCLUSION

To increase scalability and avoid out-of-memory (OOM) problems, the crucial problem of optimizing memory utilization in Graph Neural Networks (GNNs) is being tackled in this project. The need for effective GNN models that can manage large-scale networks without compromising speed is growing as graph data continues to increase exponentially in size and complexity.

Building a Graph Convolutional Network (GCN) with the Cora dataset—a well-known standard for GNN research—was the first stage of the project. We illustrated the fundamental operation of GCNs and the difficulties encountered while scaling to larger graphs using this dataset.

Each EEG channel is represented as a node in a graph by the AnxietyGNN model, which uses EEG data to categorize anxiety levels. Metrics like Mean Power, Rational Asymmetry, and Asymmetry Index are among the node properties that are taken from EEG signals in various frequency bands. To translate the electrical activity of the brain to one of four anxiety levels—Normal, Light, Moderate, and Severe—the GNN architecture analyses this graph-based data to discover correlations between EEG channels. With the use of graph-based learning and EEG data, this model has the potential to be a very effective tool for comprehending and diagnosing anxiety.

Future research will concentrate on using sampling-based strategies to optimize the GNN model used to identify anxiety from EEG signals. The deployment of GNN models is complicated by the low processing power and storage capacity of low-power edge IoT devices. As a result, GNNs must be optimized for deployment on edge computing devices with limited resources to further improve their scalability and effectiveness in real-world applications.

# REFERENCES


1. Zhao, J., Zhou, Z., Guan, Z., Zhao, W., Ning, W., Qiu, G., et al. (2019). Intentgc: A scalable graph convolution framework fusing heterogeneous information for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2347–2357).

2. Klepl D, Wu M, He F. Graph Neural Network-Based EEG Classification: A Survey. IEEE Trans Neural Syst Rehabil Eng. 2024;32:493-503. doi: 10.1109/TNSRE.2024.3355750. Epub 2024 Jan 25. PMID: 38236670.

3. Andac Demir, Toshiaki Koike-Akino, Ye Wang, Masaki Haruna, and Deniz Erdogmus. EEG-GNN: Graph neural networks for classification of electroencephalogram (EEG) signals. In 2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), pages 1061–1067. IEEE, 2021.

4. Muhammad F, Al-Ahmadi S. Human state anxiety classification framework using EEG signals in response to exposure therapy. PLoS One. 2022 Mar

5. Paul SG, Saha A, Hasan MZ, Noori SRH, Moustafa A (2024) A systematic review of graph neural network in healthcare-based applications: recent advances, trends, and future directions. IEEE Access 12:15145–15170

6. K. Zheng, S. Yu, B. Chen, CI-GNN: A Granger Causality-Inspired Graph Neural Network for Interpretable Brain Network-Based Psychiatric Diagnosis, *Neural Netw.* 172 (2024)