SleepGCN-Transformer: A Hybrid Graph-Convolutional and Transformer Network for Sleep Stage Classification

Tanmay Rathod[a,*], Rajesh Kumar Mohapatra[b,**], Santosh Kumar Satapathy[c], Nitin Singh Rajput[d]

[a]*Department of Information and Communication Technology, Pandit Deendayal Energy University, Gandhinagar, Gujarat, India*
[b]*Department of Information and Communication Technology, Pandit Deendayal Energy University, Gandhinagar, Gujarat, India*
[c]*Department of Information and Communication Technology, Pandit Deendayal Energy University, Gandhinagar, Gujarat, India*
[d]*Department of Information and Communication Technology, Pandit Deendayal Energy University, Gandhinagar, Gujarat, India*

Abstract

Objective: SleepGCN-Transformer is a hybrid deep learning model combining graph convolutional networks (GCN) and transformer networks to classify sleep stages accurately.

Methods: We use the SleepEDF dataset, focusing on four selected channels: EEG Fpz-Cz, EEG Pz-Oz, EMG submental, and EOG horizontal. Our proposed methodology integrates GCN to capture spatial dependencies across the multi-modal signals, while the Transformer encoder effectively learns temporal patterns.

Results: Training results demonstrate the model's high performance, achieving 93.12% accuracy on the training set and 93.04% accuracy on the validation set. The model leverages Focal Loss to address class imbalance, improving performance on underrepresented sleep stages.

Conclusion: Feature importance analysis highlights EMG and EEG Pz-Oz as key predictors for sleep stage classification. This approach lays the foundation for integrating Explainable AI in sleep diagnostics, enabling interpretable AI tools for clinical decision-making, and demonstrates potential for future explainable medical applications.

Keywords: Graph Convolutional Network, Transformer, Multi-head Attention, Explainable AI, EEG, Sleep Stage Classification

I. Introduction

Sleep stage classification is currently a very popular topic among researchers Supratak et al. (2017); Phan and et al. (2019). In this section, we discuss why it is needed and why sleep stage classification is necessary Tsinalis et al. (2016). First, we begin with the basics—what the need for this classification is and how it has traditionally been done manually Chambon et al. (2018) versus how it can now be automated using algorithms Perslev et al. (2021a); Liang et al. (2020). In this research study, we explore from scratch why sleep stage classification is needed, its benefits, and how modern machine learning techniques (deep learning) Dong et al. (2019b) have helped solve this problem. We also explain the steps we have taken to address this challenge. Sleep is a very necessary function for humans. It helps heal and recharge neurons Zhang et al. (2020), maintain mental health Supratak et al. (2017), and recover from muscle and injury loss Phan and et al. (2019). Sleep also recharges brain cells and, in short, is essential for a properly functioning body Tsinalis et al. (2016).

Sleep occurs in different stages throughout the night Chambon et al. (2018). Sometimes, we feel more refreshed after a short sleep, while other times, even after sleeping longer, we do not feel as fresh. So, what is considered good sleep? Accord-

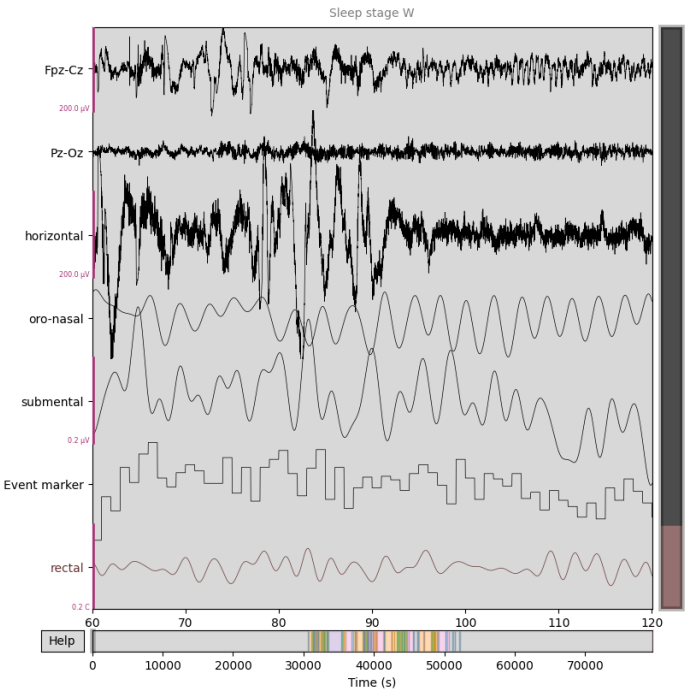


Figure 1: Sample Wake Stage Signals from the SleepEDF Dataset

*Corresponding author
**Corresponding author
Corresponding author
Email addresses: tanmayrathod777@gmail.com (Tanmay Rathod), abc@google.com (Santosh Kumar Satapathy)

ing to research, 7 to 9 hours of sleep is recommended Perslev et al. (2021a), although this can vary from person to person. During sleep cycles, our brain generates electrical signals that pass through neurons Liang et al. (2020). These signals create certain patterns, based on which we classify sleep stages **?**. Sleep stages are divided into five categories: Wake, N1, N2, N3, N4, and Rapid Eye Movement (REM) Dong et al. (2019b). These are further classified into two main types—Non-Rapid Eye Movement (NREM) and REM Zhang et al. (2020). As we begin to sleep, we transition from wakefulness to light sleep, then to deeper sleep (N2), followed by the deepest sleep stages (N3 and N4). Finally, we enter REM sleep, where our eyeballs move, and we transition between sleep cycles. During REM sleep, we experience dreams, muscle healing occurs, and the body enters a locked state to prevent movement and injury due to dream-related tension Supratak et al. (2017).

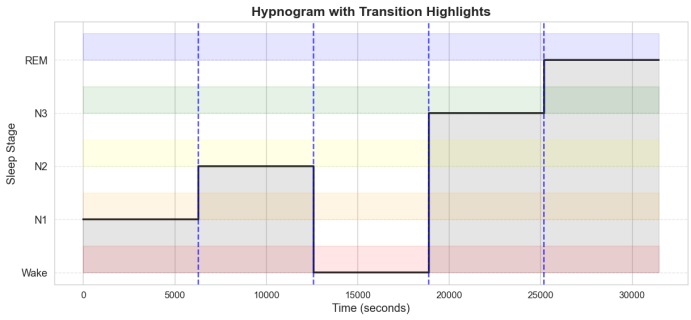


Figure 2: Hypnogram Sleep Stage Transition

To diagnose various sleep disorders, experts analyze sleep stages to identify irregularities in transitions Phan and et al. (2019). For this, doctors and researchers collect physiological signals such as electroencephalogram (EEG) (brain wave signals), electromyogram (EMG) (chin muscle movement signals), electrooculogram (EOG) (eye movement signals), electrocardiogram (ECG) (cardiac rhythm and heart health), temperature, rectal readings, and several other channels **?**. These signals are recorded while the person sleeps, and the data is stored in hypnograms or in European Data Format (EDF) files Chambon et al. (2018). During sleep studies, individuals sleep for around 8 hours in a controlled environment while these signals are monitored, recorded, and later reviewed by experts to manually mark sleep stage transitions and detect abnormalities Liang et al. (2020). This manual process is tedious and time-consuming Perslev et al. (2021a). But imagine if, instead of requiring experts to mark transitions manually, we could develop an algorithm that takes the EDF files as input—without pre-marked annotations—and automatically classifies the sleep stages Li et al. (2024); Satapathy and Loganathan (2022, 2021); Satapathy et al. (2021a,b). That would be amazing, right? And with around 93% robust accuracy, this approach becomes even more promising Dong et al. (2019b).

The goal of this paper is to develop such an algorithm. First, we discuss the datasets used in this research. There are many publicly available sleep datasets, such as the Sleep Heart Health Study (SHHS), Sleep-European Data Format (Sleep-EDF), and Instituto de Engenharia Electrónica e Telemática de Aveiro (ISRUC) sleep data Perslev et al. (2021a), provided by various organizations and governments for research purposes while ensuring patient confidentiality. In this study, we primarily use the Sleep-EDF dataset, and we will also share the configuration details for this work. This is the work we have done, and throughout this paper, we will describe each step in detail so that this research can be reproduced.

In this paper, we present a holistic method of sleep stage classification via a hybrid deep learning network. Section 2 is devoted to the related work and a brief review of existing literature in this area. Section 3 describes our proposed method along with an explanation of Graph Convolutional Networks (GCN) and Transformer-based structures employed in our network. Section 4 shows the experimental results, evaluation measures, and a strong defense of our method. We also present the Explainable AI (XAI) component in the same section, describing how it increases interpretability in sleep diagnosis. In Section 5, we perform an ablation study to examine the contributions of different components of our model. We conclude the paper with important findings and future work directions.

II. Related Work

Our work introduces a novel approach to sleep stage classification by leveraging graph-based modeling and deep learning techniques. The key contributions of this research are as follows:

Overall, our approach brings a novel perspective to sleep stage classification by integrating graph-based modeling, robust preprocessing, and interpretability, ultimately pushing the boundaries of automated sleep analysis.

2.1. Contribution

- **Graph-Based Representation of Sleep EEG Data:** Unlike traditional sequential methods, we construct a graph-based representation of multi-channel sleep signals. This enables the model to effectively capture spatial dependencies and interactions between signals such as EEG (electroencephalogram), EMG (electromyogram), and EOG (electrooculogram), which are crucial for accurate sleep stage classification.

- **Hybrid GCN-Transformer Architecture:** We propose a novel hybrid model that combines Graph Convolutional Networks (GCN) to model spatial relationships and Transformer networks to learn temporal dependencies. This dual approach enhances the model's ability to extract meaningful patterns across time and space from the sleep data.

- **Robustness and Pretrained Model Testing:** To ensure generalizability and reliability, we test the model both on training data and on unseen samples using pretrained weights. We implement custom test scripts to validate model predictions on EDF (European Data Format) files, ensuring compatibility and performance in real-world scenarios.

Study Title	Architecture / Methodology	Dataset(s)	Accuracy (%)
DeepSleepNet Supratak et al. (2017)	CNN + BiLSTM	Sleep-EDF, MASS	82.0–86.2
SleepEEGNet Mousavi et al. (2019)	CNN + Seq2Seq RNN	Sleep-EDF	84.26
RobustSleepNet Phan et al. (2022)	CNN + Transfer Learning	8 PSG datasets	~97 (F1)
Cross-Modal Transformers Mostafaei et al. (2024)	Transformer + CNN	Sleep-EDF	Not Specified
Attention-ST-GCN Ji et al. (2022)	ST-GCN + Attention Blocks	Sleep-EDF, ISRUC	+4.6–5.3
GAT-GRU Network Biswal and et al. (2017)	Multi-layer GAT + GRU	ISRUC, SHHS	82.5–87.3
Sleep Staging Clinical Study Phan and et al. (2018)	CNN + LSTM	Sleep-EDF, Clinical	82.9–83.9
DL vs ML Review Liu (2024)	Hybrid CNN, LSTM, etc.	Multiple	Up to 96.4
Sparse DBN + Ensemble Supratak et al. (2017)	DBN + Ensemble Classifiers	Not Specified	91.31
DL for Sleep Scoring Biswal et al. (2019)	CNN + LSTM	SHHS, SNUBH	76–82
SleepTransformer Phan et al. (2022)	Transformer + Positional Encoding	Sleep-EDF, ISRUC	85.5
ChronoNet Mousavi et al. (2019)	CNN + RNN Hybrid	Sleep-EDF	86.0
SleepDPC Ji et al. (2022)	Contrastive Learning + Transformer	Sleep-EDF Expanded	87.6
EEGSNet Li et al. (2024)	Siamese CNN + Contrastive Loss	ISRUC, MASS	88.1
MultiResNet Pei et al. (2022)	Multi-resolution CNN	Sleep-EDF, SHHS	89.4
EfficientSleepNet Phan et al. (2022)	Lightweight CNN + Attention	MASS	84.7
AttnSleep Eldele et al. (2021)	Attention-based BiLSTM	Sleep-EDF	86.9
DeepSleepGAN Pei et al. (2022)	GAN + CNN Classifier	Sleep-EDF	83.3
SleepGRU Mousavi et al. (2019)	GRU-based RNN	ISRUC	81.6
R-CNN Sleep Supratak et al. (2017)	Region-based CNN	Sleep-EDF	85.1

Table 1: Related Work in Sleep Stage Classification

- **Advanced Preprocessing and Feature Extraction:** A robust preprocessing pipeline is applied that includes adaptive filtering to remove noise, epoch segmentation for time-based structuring, and class label mapping. This improves the quality of the input signals and ensures consistent data formatting for training and evaluation.
- **Handling Class Imbalance with Focal Loss and Label Smoothing:** Our training strategy incorporates Focal Loss to focus on hard-to-classify examples and label smoothing to prevent overfitting. These techniques help in managing class imbalance across sleep stages, especially the underrepresented ones like N1 or REM.
- **Comprehensive Evaluation on the Sleep-EDF Dataset:** The model is rigorously tested on the Sleep-EDF dataset, a publicly available and widely accepted benchmark in sleep research. Our method demonstrates superior accuracy and robustness when compared to existing baseline models.
- **Explainable AI (XAI) and Channel Importance Analysis:** We integrate an Explainable AI framework to interpret model decisions. By analyzing channel-level contributions, we identify EEG Pz-Oz and EMG signals as the most influential features, offering insights for clinical applications and future enhancements.

III. Methodology

We are proposing the Sleep-GCN-Transformer in which graph convolution neural network we are using for to capture the spatial feature capturing and we are using transformer for to capture time series features We have used the Sleep EDF Dataset available here: https://www.physionet.org/content/sleep-edfx/1.0.0/. We selected data from around 30 patients, resulting in 60 files, including 30 EDF (European Data Format) files. These files contain multiple bio-signals and channels. Additionally, we have hypnogram files, which record the time and stages, with multiple labeled stages such as wake, N1, N2, N3, N4, and REM, along with their corresponding times in the format HH.MM.SS. The channels in the EDF data include multiple bio-signals, such as rectal temperature, EOG, EMG, EEG (FPZ-CZ, PZ-OZ), and others. These channels provide data at a specific frequency in Hz, which we use as features to train our model. Our model requires two main types of input: spatial features, which we extract from the signals for graph representation, and temporal dependency data, which captures the timing of signals for training.

In the shown image, you can see multiple signal channels with different frequencies, such as EEG, EOG, EMG, nasal, and temperature, along with their respective event markers. The time interval is 60 seconds, and the frequency ranges from 60 to 120 seconds.

As shown above, the frequency changes quite rapidly, making it difficult to capture the patterns and information. This rapid variation increases the chances of false classification. We are dealing with multiple challenges here, including noise, class imbalance (with more data from the wake class), and a large number of time steps. At each second, there are around 100 time steps, adding to the complexity of the task.

3.1. Dataset Information

The dataset used for this study is sourced from the Sleep-EDF dataset, a publicly available collection of sleep recordings that include both EEG (Electroencephalogram) signals and sleep stage labels. The data is stored in EDF (European Data Format) files, which consist of multiple channels capturing various physiological signals. Specifically, the dataset contains both PSG (Polysomnography) signals and corresponding hypnogram files, which contain the annotations of the sleep stages.

The dataset consists of multiple participants, and for each participant, two types of files are available:

- **PSG files:** These contain the time-series physiological signals (e.g., EEG, EMG, ECG, etc.).
- **Hypnogram files:** These contain the sleep stage labels corresponding to each time segment in the PSG data.

Algorithm 1: Main Pipeline for Sleep Stage Classification

Data: Raw EEG and annotation file paths
Result: Trained SleepGCN_Transformer model

fnames ← `[('SC4001E0-PSG.edf', 'SC4001EC-Hypnogram.edf')]`
$(X_{\text{all}}, Y_{\text{all}})$ ← `preprocess_data(fnames)`
X_{tensor} ← `torch.tensor(`X_{all}`, float32)`
Y_{tensor} ← `torch.tensor(`Y_{all}`, long)`
dataset ← `SleepEEGDataset(`$X_{\text{tensor}}, Y_{\text{tensor}}$`)`
train_size ← $0.8 \times$ `len(`*dataset*`)`
val_size ← `len(`*dataset*`)` − *train_size*
(*train_dataset*, *val_dataset*) ← `random_split`(dataset, [train_size, val_size])
model ← `SleepGCN_Transformer`(
    input_dim=4, hidden_dim=256, output_dim=5,
    num_layers=5, nhead=4, dropout=0.1)
trainer ← `SleepTrainer`(
    model, train_dataset, val_dataset,
    batch_size=32, lr=0.0003,
    num_epochs=20, weight_decay=1e-4)
`trainer.train()`

Algorithm 2: SleepEEGDataset Class Definition

Data: Input features X, labels Y
Result: A graph-based dataset with node features and edge structure

```
class SleepEEGDataset(Dataset):
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y
        A = np.array([
        ])
        edge_index = torch.tensor(np.array(np.nonzero(A)), dtype=torch.long)
        edge_weights = torch.tensor(A[A.nonzero()], dtype=torch.float32)
        self.edge_index = edge_index
        self.edge_weights = edge_weights
    def len(self):
        return len(self.X)
    def get(self, idx):
        label = torch.tensor([self.Y[idx]], dtype=torch.long)
        return Data(x=node_features, edge_index=self.edge_index,
                    edge_attr=self.edge_weights, y=label)
```

The dataset includes the following types of signals:

- **EEG (Electroencephalogram):** These signals are recorded from multiple channels like **Fpz-Cz**, **Pz-Oz**, and others, which capture brain activity.
- **EMG (Electromyogram):** Signals obtained from the submental area to capture muscle activity during sleep.
- **EOG (Electrooculogram):** These capture eye movements during sleep.
- **ECG (Electrocardiogram):** Heart activity signals that can be used for additional physiological context.
- **Temperature and Rectal Signals:** These monitor the body's temperature during sleep.

The following table summarizes the file details available for each participant in the Sleep-EDF dataset, including file names, sizes, and the number of channels recorded in each PSG file:

PSG Signals: The PSG files contain multiple channels of physiological signals. These include EEG (such as **Fpz-Cz**, **Pz-Oz**), EMG, EOG, ECG, and other related physiological recordings. These channels provide comprehensive insights into the subject's brain activity, muscle activity, eye movement, and heart signals during sleep.

Hypnogram Files: The hypnogram files contain the sleep stage labels, which are aligned with the corresponding time seg-

Algorithm 3: Preprocess Sleep EEG Data

Data: List of file pairs: (psg_file, ann_file)
Result: Preprocessed feature matrix X and labels Y

$X\_all \leftarrow []$
$Y\_all \leftarrow []$
foreach (psg_file, ann_file) *in fnames* **do**
    $raw \leftarrow$ mne.io.read_raw_edf($psg\_file, preload = True$)
    $annotations \leftarrow$ mne.read_annotations(ann_file)
    $raw.set\_annotations(annotations)$
    $raw.pick\_channels(channels)$
    $raw.filter(0.3, 30., fir\_design = ' firwin')$
    $data \leftarrow raw.get\_data()$
    $sfreq \leftarrow raw.info['sfreq']$
    $epochs\_list \leftarrow []$
    $labels\_list \leftarrow []$
    foreach *annot in annotations* **do**
        $stage \leftarrow annot['description']$
        if $stage \in stage\_mapping$ **then**
            $onset\_sample \leftarrow \text{int}(annot['onset'] \times sfreq)$
            $duration\_samples \leftarrow \text{int}(annot['duration'] \times sfreq)$
            $epoch\_length\_samples \leftarrow \text{int}(30 \times sfreq)$
            for $i \leftarrow 0$ **to** *duration_samples with step epoch_length_samples* **do**
                $start \leftarrow onset\_sample + i$
                $stop \leftarrow start + epoch\_length\_samples$
                if $stop > data.shape[1]$ **then**
                    break
                $epochs\_list.append(data[:, start : stop])$
                $labels\_list.append(stage\_mapping[stage])$
    $X \leftarrow$ np.array($epochs\_list$)
    $Y \leftarrow$ np.array($labels\_list$)
    $X\_all.append(X)$
    $Y\_all.append(Y)$
$X\_all \leftarrow \texttt{np.concatenate}(X\_all, axis = 0)$
$Y\_all \leftarrow \texttt{np.concatenate}(Y\_all, axis = 0)$
return X_all, Y_all

ments of the PSG signals. These labels represent different sleep stages, such as Wake, NREM (Non-Rapid Eye Movement), and REM (Rapid Eye Movement) stages. These annotations are crucial for supervised learning tasks, as they provide the ground truth for classification.

3.2. Preprocessing

At the beginning of our preprocessing method, we removed several channels that, while important for classification, were not as impactful as the selected channels. We chose to retain the EEG channels (Fpz-Cz, Pz-Oz), the EOG channel (horizontal), which captures eye movement, and the EMG submental channel (submental), which records chin muscle activity.

The preprocessing pipeline starts with reading the EDF file and annotating it according to the mapped classes shown above. We then select the relevant channels, apply filtering, and segment the data into 30-second samples using the 'firwin' filter design. Unnecessary frequencies are removed, and we retain only the 0.3 to 30 Hz frequency range, as frequencies beyond this range are considered noise for our study. While this is not a standard approach, it was found to be more suitable for our specific task.

Finally, we create epochs and store them in a structured format. The resulting input shape for our model is $(X, 4, 3000)$, where X represents the number of samples, 4 corresponds to the selected channels, and 3000 represents the time steps.

3.2.1. Filtering Method

We used the windowed time-domain design (firwin) method with the following parameters: Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation, Lower passband edge: 0.30 Hz, Lower transition bandwidth: 0.30 Hz, (-6 dB cutoff frequency: 0.15 Hz) Upper passband edge: 30.00 Hz Upper transition bandwidth: 7.50 Hz (-6 dB cutoff frequency:

Algorithm 4: Training Pipeline for Sleep EEG Model

Data: Training and Validation datasets, Model, Learning rate, Epochs, Batch size, Weight decay
Result: Trained model with performance metrics on training and validation data

```
Function FocalLoss(logits, targets):
    probs ← F.softmax(logits, dim=-1)
    targets_one_hot ← F.one_hot(targets, num_classes=logits.shape[1]).float()
    targets_one_hot ← targets_one_hot * (1 - label_smoothing) + label_smoothing / logits.shape[1]
    ce_loss ← -targets_one_hot * torch.log(probs + 1e-9)
    focal_loss ← alpha * (1 - probs) ** gamma * ce_loss
    return focal_loss.mean()
Function SleepTrainer(model, train_dataset, val_dataset, batch_size, lr, num_epochs, weight_decay):
    class_weights ← self.get_class_weights(train_dataset).to(device)
    loss_fn ← FocalLoss(alpha=class_weights, gamma=3, label_smoothing=0.1)
    optimizer ← optim.AdamW(model.parameters(), lr=lr, weight_decay=weight_decay)
    scheduler ← optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs)
    train_loader ← DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader ← DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
    for epoch ← 1 to num_epochs do
        model.train()
        total_train_loss, total_train_accuracy ← 0, 0
        foreach batch in train_loader do
            batch ← batch.to(device)
            optimizer.zero_grad()
            output ← model(batch)
            target ← batch.y
            loss ← loss_fn(output, target)
            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()
            total_train_accuracy += compute_accuracy(output, target)
        scheduler.step()
        Validation Phase:
        model.eval()
        total_val_loss, total_val_accuracy ← 0, 0
        foreach batch in val_loader do
            batch ← batch.to(device)
            output ← model(batch)
            target ← batch.y
            loss ← loss_fn(output, target)
            total_val_loss += loss.item()
            total_val_accuracy += compute_accuracy(output, target)
        avg_train_loss ← total_train_loss / len(train_loader)
        avg_train_accuracy ← total_train_accuracy / len(train_loader)
        avg_val_loss ← total_val_loss / len(val_loader)
        avg_val_accuracy ← total_val_accuracy / len(val_loader)
        print(f"Epoch epoch/num_epochs: Train Loss: avg_train_loss:.4f, Train Acc: avg_train_accuracy:.4f | Val Loss:
          avg_val_loss:.4f, Val Acc: avg_val_accuracy:.4f")
return Trained Model
```

33.75 Hz)

3.3. Graph creation

From these 4 channels we make the graph with the connecting nodes structures where edges representing the relationship with the channels we are trying to capture the spatial feature

Algorithm 5: SleepGCN_Transformer Model Declaration

Data: Input features of shape (batch size, input_dim), graph structure with edges (edge_index), batch information (batch)
Result: Predicted outputs for each input, of shape (batch size, output_dim)

```
Function SleepGCN_Transformer(input_dim, hidden_dim, output_dim, num_layers, nhead, dropout):
    gcn_layers ← nn.ModuleList()
    batch_norms ← nn.ModuleList()
    gcn_layers.append(GCNConv(input_dim, hidden_dim))
    batch_norms.append(BatchNorm(hidden_dim))
    for i ← 1 to num_layers - 1 do
        gcn_layers.append(GCNConv(hidden_dim, hidden_dim))
        batch_norms.append(BatchNorm(hidden_dim))
    encoder_layer ← TransformerEncoderLayer(d_model=hidden_dim, nhead=nhead, dropout=dropout,
     batch_first=True)
    transformer ← TransformerEncoder(encoder_layer, num_layers=2)
    fc ← nn.Linear(hidden_dim, output_dim)
    activation ← ReLU()
    dropout ← Dropout(dropout)
Function forward(data):
    X, edge_index, batch ← data.x, data.edge_index, data.batch
    for i ← 0 to len(gcn_layers) - 1 do
        X ← activation(gcn_layers[i](X, edge_index))
        X ← batch_norms[i](X)
        X ← dropout(X)
    X ← global_mean_pool(X, batch)
    X ← X.unsqueeze(1)
    X ← transformer(X)
    X ← X.squeeze(1)
    return fc(X)
```

from the nodes and edges where these node is representing the features we have given weightage from the some papers and experience over these work where you can see we have given maximum weightage over the eeg channels and less weightage on between the emg and EOG we can give the self node graph weightage also but we haven't done that here EOG have lower correlations.

The last dataset consists of 11,076 samples, organized as `Data(x=[3000, 4], edge_index=[2, 12], edge_attr=[12], y=[1])`, in which `x` is the time-series data with 3,000 time steps and 4 channels, `edge_index` specifies the connectivity of the graph, `edge_attr` is the designated edge weights, and `y` is the corresponding sleep stage label. The dataset is formed through the `SleepEEGDataset` class. This formalized process accurately captures spatial dependencies among EEG, EMG, and EOG signals, improving sleep stage classification.

3.4. Sleep GCN Architecture

3.4.1. Graph Convolution Neural Network

In our model, we use a Graph Convolutional Neural Network (GCN) to capture spatial domain features. The input to the model is a graph tensor where each node has 3,000 time-step features, and there are a total of 11,076 samples.

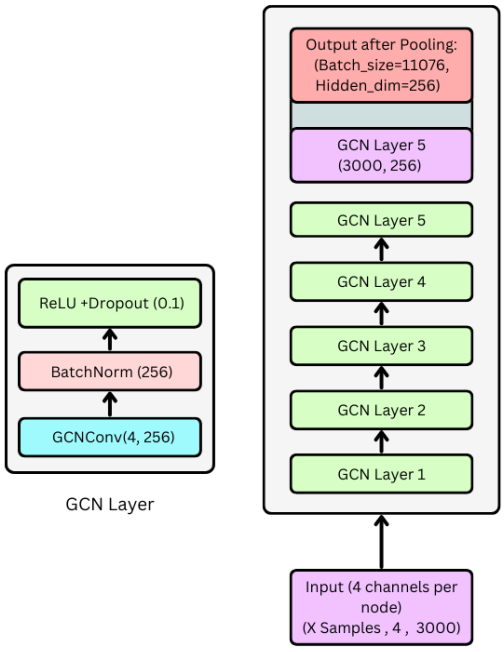
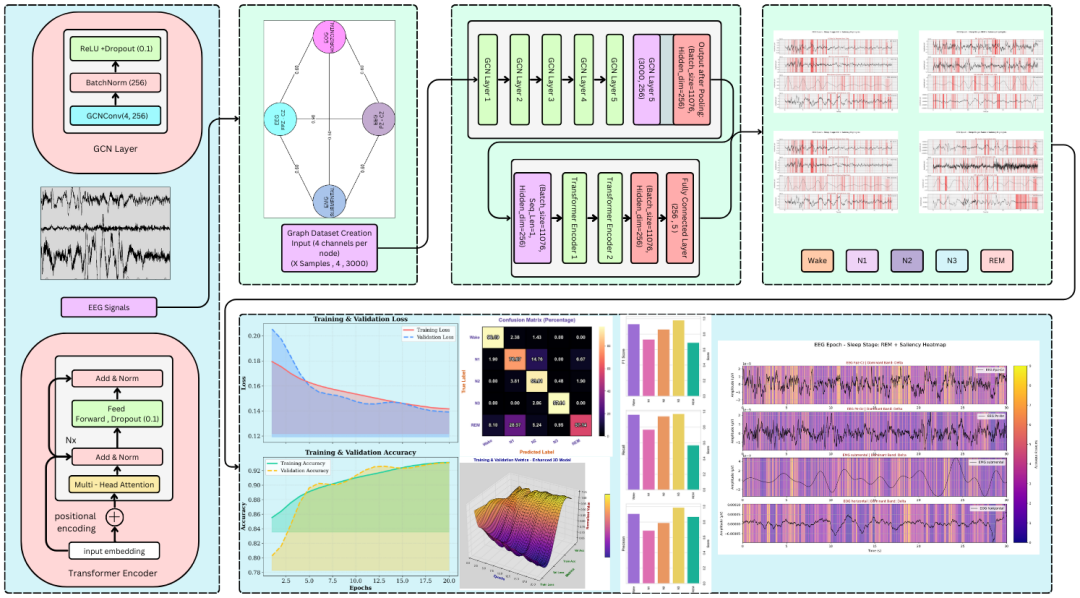


Figure 7: Proposed GCN Architecture

This tensor serves as the input to the Graph Convolution



Overall Architechture of Proposed GCN-Transformer for sleep stages Classification

Figure 3: Overall Diagram of the Sleep GCN-Transformer Architecture and Results Pipelinee

Layer (GCL), with the same number of output samples assigned corresponding labels. The GCN processes an input of four nodes, corresponding to the four channels provided, with the first GCN layer transforming it into a shape of (4, 256). Afterward, we apply batch normalization with 256 units, followed by a ReLU activation function and a dropout rate of 0.1. The GCN architecture consists of five identical layers, with the final (fifth) layer feeding into a pooling layer. The pooling operation follows the equation given below, where we use global mean pooling over nodes based on batch indices. The batch size is (11,076, 256), resulting in a graph-level embedding.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{1}$$

Where $H^{(l)} \in \mathbb{R}^{N\times D}$ is the feature matrix at layer l, where N is the number of nodes and D is the feature dimension. $\tilde{A} = A + I$ is the adjacency matrix with added self-loops (identity matrix I), and $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$. $W^{(l)}$ is the trainable weight matrix for layer l, and σ is a non-linear activation function, typically ReLU.

The second equation represents the transformation of the original input feature matrix X after one GCN layer:

$$X' = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW\right) \tag{2}$$

Where X is the original input feature matrix, and X' is the transformed feature representation after one GCN layer.

The third equation describes the pooling operation:

$$x = \frac{1}{|V|}\sum_{v\in V} h_v \tag{3}$$

Where V is the set of nodes in the graph, h_v is the feature vector of node v, and x is the pooled graph-level representation.

The fourth equation represents the ReLU activation function applied to x:

$$\text{ReLU}(x) = \max(0, x) \tag{4}$$

The fifth equation is the softmax function applied to x_i:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5}$$

Finally, the sixth equation represents the normalization of x' using the mean and standard deviation:

$$x'' = x' - \frac{u}{\sigma} \tag{6}$$

Where u is the mean of x', σ is the standard deviation of x', and x'' is the normalized feature.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{7}$$

Where $H^{(l)} \in \mathbb{R}^{N\times D}$ is the feature matrix at layer l, where N is the number of nodes and D is the feature dimension. $\tilde{A} = A + I$ is the adjacency matrix with added self-loops (identity matrix I), and $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$. $W^{(l)}$ is

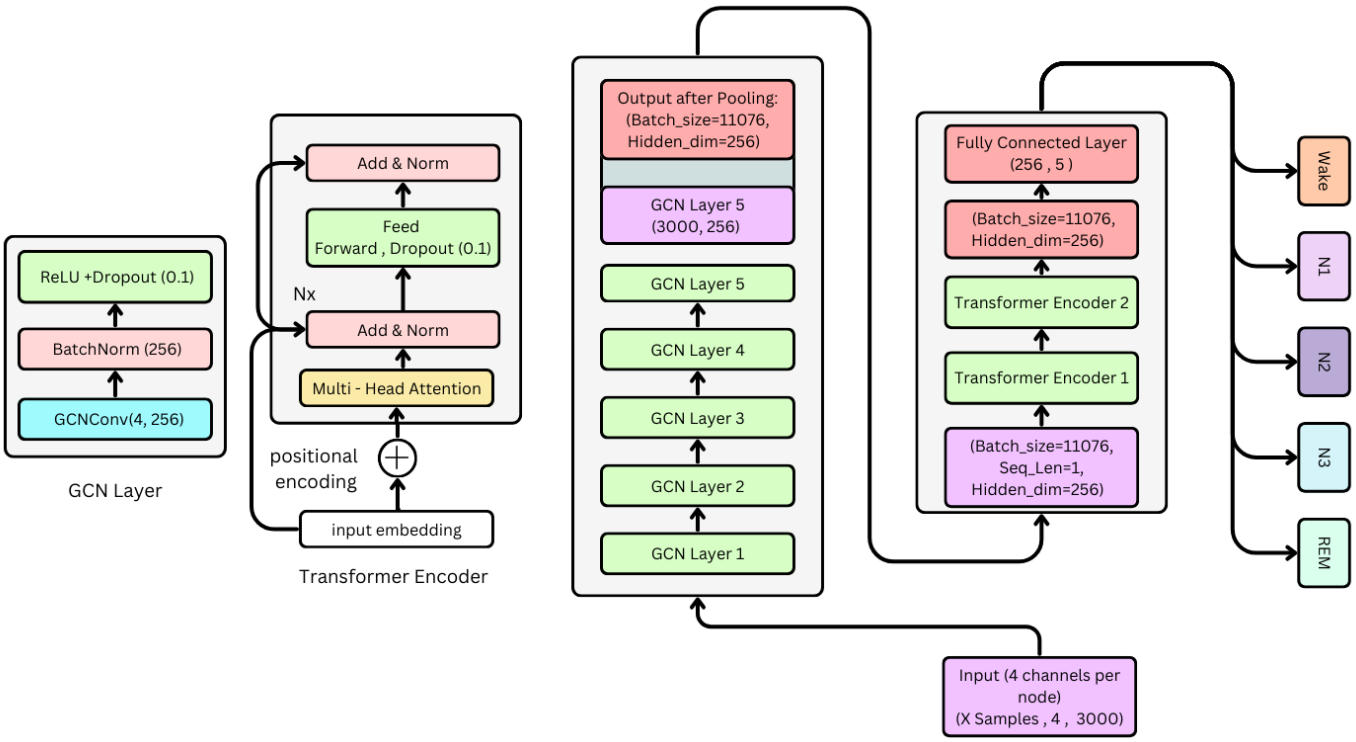


Figure 4: Proposed Architecture of the sleep GCN- Transformer Architecture

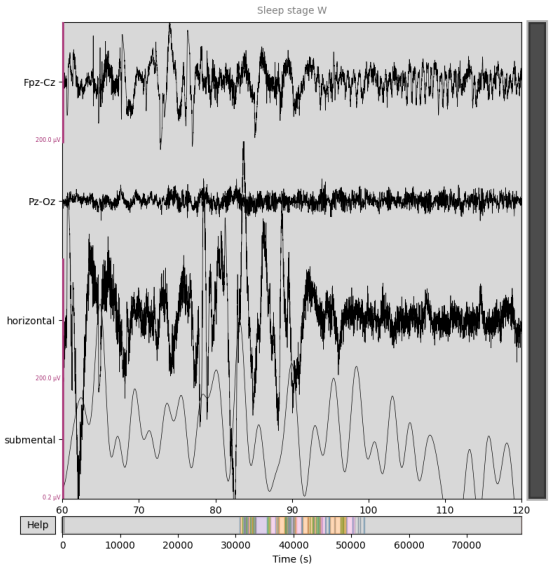


Figure 5: Filtered Chanels from SleepEDF Dataset

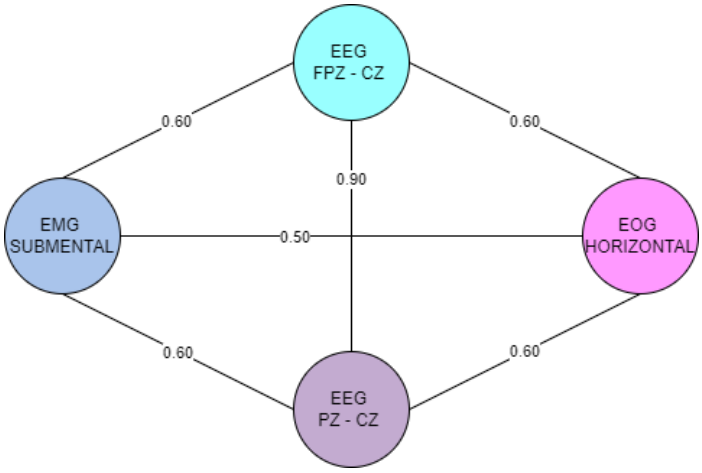


Figure 6: Graph Dataset Creation

the trainable weight matrix for layer l, and σ is a non-linear activation function, typically ReLU.

The second equation represents the transformation of the original input feature matrix X after one GCN layer:

$$X' = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW\right) \tag{8}$$

Where X is the original input feature matrix, and X' is the transformed feature representation after one GCN layer.

The third equation describes the pooling operation:

$$x = \frac{1}{|V|}\sum_{v\in V} h_v \tag{9}$$

Where V is the set of nodes in the graph, h_v is the feature vector of node v, and x is the pooled graph-level representation.

The fourth equation represents the ReLU activation function applied to x:

$$\text{ReLU}(x) = \max(0, x) \tag{10}$$

Table 2: Sample Data Files and Channel Information in Sleep-EDF Dataset

File Name	Size (Bytes)	Channels Included
SC4001E0-PSG.edf	48,338,048	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4001EC-Hypnogram.edf	4,620	Sleep Stage Labels
SC4011E0-PSG.edf	51,110,528	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4011EH-Hypnogram.edf	3,896	Sleep Stage Labels
SC4021E0-PSG.edf	51,147,008	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4021EH-Hypnogram.edf	4,804	Sleep Stage Labels
SC4031E0-PSG.edf	51,438,848	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4031EC-Hypnogram.edf	3,700	Sleep Stage Labels
SC4041E0-PSG.edf	46,878,848	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4041EC-Hypnogram.edf	4,830	Sleep Stage Labels
SC4051E0-PSG.edf	49,651,328	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4051EC-Hypnogram.edf	3,976	Sleep Stage Labels
SC4061E0-PSG.edf	50,526,848	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4061EC-Hypnogram.edf	2,620	Sleep Stage Labels
SC4071E0-PSG.edf	51,256,448	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4071EC-Hypnogram.edf	3,628	Sleep Stage Labels
SC4081E0-PSG.edf	51,001,088	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4081EC-Hypnogram.edf	4,306	Sleep Stage Labels
SC4091E0-PSG.edf	49,833,728	EEG (Fpz-Cz, Pz-Oz), EMG, EOG, ECG
SC4091EC-Hypnogram.edf	4,474	Sleep Stage Labels

Note: The dataset contains additional files with similar structure.

Table 3: Sleep Stages Mapping

Sleep Stage	Label	Mapped Class
Sleep Stage Wake	0	0
Sleep Stage 1	1	1
Sleep Stage 2	2	2
Sleep Stage 3	3	3
Sleep Stage 4	4	3
Sleep Stage REM	5	4

The fifth equation is the softmax function applied to x_i:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{11}$$

Finally, the sixth equation represents the normalization of x' using the mean and standard deviation:

$$x'' = x' - \frac{u}{\sigma} \tag{12}$$

Where u is the mean of x', σ is the standard deviation of x', and x'' is the normalized feature.

3.5. Transformer: Temporal Dependency Modeling in Time Series Data

The input sequence is first mapped to a continuous vector space using an embedding matrix. Here, X represents the input token representation, W_{embed} is the embedding weight matrix, and X_{embed} is the embedded representation. This process can be expressed mathematically as:

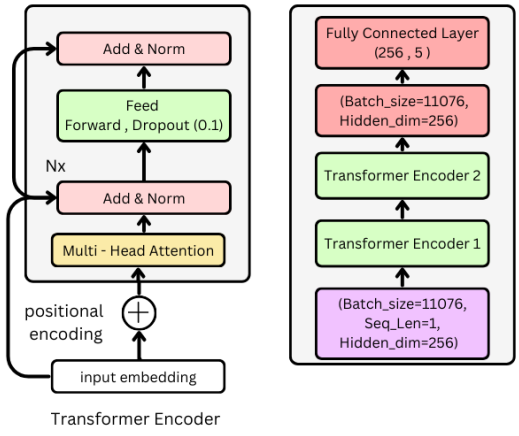


Figure 8: Proposed Transformer Architecture

$$X_{\text{embed}} = XW_{\text{embed}} \tag{13}$$

Next, we apply Position Encoding to handle the ordering of the sequence. The position encoding is calculated using sine and cosine functions, where pos denotes the position index of the token in the sequence and d is the embedding dimension. The sine and cosine functions are used to encode different frequency signals to capture relative positions:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{14}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{15}$$

The next step in the transformer architecture involves Single-Head Attention, where the input X is used to compute the Query Q, Key K, and Value V by multiplying X with the corresponding learned weight matrices W_q, W_k, and W_v. This operation can be written as:

$$Q = XW_q, \quad K = XW_k, \quad V = XW_v \tag{16}$$

Following this, we compute the attention scores H_1 using the scaled dot-product attention mechanism, where the attention is computed as the softmax of the dot product between the Query and Key, scaled by $\sqrt{d_k}$, where d_k is the dimensionality of the key vectors. The output is a weighted sum of the value vectors:

$$H_1 = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{17}$$

Next, the Multi-Head Attention mechanism is used. Instead of using a single attention head, multiple attention heads $H_1, H_2, ..., H_h$ are concatenated, and a final weight matrix W_0 is applied as a linear transformation after concatenation to ensure the output matches the original feature dimensions:

$$H = \text{Concat}(H_1, H_2, ..., H_h)W_0 \tag{18}$$

The transformer model does not have built-in recurrence or convolution to handle word order, so position encoding is added to the embeddings to ensure that the model can understand the position of each token in the sequence.

Single-Head Attention: Attention mechanisms allow the model to focus on different words in a sentence while processing each token. The Query Q, Key K, and Value V are derived by multiplying the input X with the corresponding learned weight matrices. The output of the attention mechanism is a weighted sum of the value vectors.

Multi-Head Attention: Instead of using a single attention mechanism, multiple attention heads are used to capture different features. These heads are concatenated, and a trainable weight matrix W_0 is applied as a linear transformation to ensure the output matches the required feature dimensions.

Once data goes through the pooling layer, it proceeds to the input embedding phase, then positional encoding. Input embedding is done to transform the input sequence into a continuous vector space, which is in the form of a matrix. Because transformers lack inherent recurrence or convolution operations to encode word order, positional encoding is incorporated into the embeddings to include sequential information. Before the sequence comes to multi-head attention, we first learn the reason for attention being applied.

In a single-head attention model, attention mechanisms assist in attending to various words (or tokens) during the processing of every token—quite prevalent in NLP applications. The attention mechanism is represented in terms of Query (Q), Key (K), and Value (V):

- **Query (Q):** Token whose attention is to be calculated.
- **Key (K):** Memory-residing representations.
- **Value (V):** The genuine information extracted based on attention scores.

These are multiplied with learned trainable weight matrices W_q, W_k, and W_v and calculated using the softmax equation. In a multi-head attention mechanism, several attention heads handle the input independently, and their outputs are concatenated. The output then passes through Add & Norm, followed by a Feedforward Layer with dropout of 0.1. Another Add & Norm comes after this. This whole configuration constitutes a Transformer Encoder block.

In the model we are employing, two Transformer Encoder layers are used, which then lead to a hidden neural network with 256 neurons. After that, the output goes into the fully connected classifier comprising five neurons and giving logits for four-stage classification.

3.6. Training Pipeline and Methodology

The training pipeline for our SleepGCN-Transformer model is intended to maximize the performance of the model in sleep stage classification based on multi-channel physiological signals. The pipeline employs Focal Loss, which is a custom loss function that reduces class imbalance and adds label smoothing for additional improvement to the model's stability. in this The training process is controlled by our SleepTrainer class, which takes charge of important tasks such as the loading data, training the model, and evaluating the model's performance. With the AdamW optimizer and weight decay using the function CosineAnnealingLR as the scheduler, in the pipeline which enables adaptive learning across the training process arround the model. The PyTorch Geometric DataLoader optimizes batch processing of the sleep dataset, where it is divided into training and validation sets. In every epoch, the model is trained on the training dataset, with its accuracy calculated. After every epoch, the model is tested on the validation set to monitor its generalization capability. in this pipeline we also calculates class weights from the occurrence of each class in the dataset, giving greater importance to less frequent classes. This assists in enhancing the model's capability to predict sleep stages that could otherwise be overlooked due to class imbalance. The process is cycled through a specified number of epochs, tracking performance metrics including train loss, accuracy, and validation loss to optimize model convergence.

3.6.1. Label Smoothing

To improve generalization and mitigate overconfidence in model predictions, label smoothing is applied. The smoothed label distribution is defined as:

$$y_{\text{smooth}} = y(1 - \epsilon) + \frac{\epsilon}{C} \tag{19}$$

where ϵ is the smoothing factor and C is the number of classes. The loss function incorporating label smoothing and Focal Loss is given by:

$$L = \alpha(1 - p)^{\gamma}(-y_{\text{smooth}} \log p) \tag{20}$$

3.6.2. Handling Class Imbalance with Focal Loss

Class imbalance is a major challenge in sleep stage classification, where certain sleep stages occur less frequently. Standard Cross-Entropy loss treats all samples equally, which often results in poor performance for minority classes. To address this, we employ Focal Loss, defined as:

$$FL(p_t) = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{21}$$

where: - p_t is the model's predicted probability for the correct class. - α is a weighting factor balancing class importance. - γ is a focusing parameter that reduces the influence of well-classified examples, emphasizing harder samples.

A higher γ value increases focus on difficult examples, enhancing model robustness in imbalanced datasets. When $\gamma = 0$, the function behaves like standard Cross-Entropy Loss.

3.6.3. Class Weight Computation

To further mitigate class imbalance, class-specific weights are computed as:

$$w_c = \frac{\mathcal{N}}{(\mathcal{K} + 1)^{0.5}} \tag{22}$$

where: - $\mathcal{N}$ represents the total number of samples. - $\mathcal{K}$ denotes the number of classes. - w_c is the computed weight assigned to a class.

Less frequent sleep stages receive higher weights, ensuring that minority classes contribute more significantly to the overall loss.

3.6.4. Optimization Strategy

The model is trained using the **AdamW** optimizer, which decouples weight decay from the gradient update process. The Adam parameter update rule is given by:

$$\theta_{t+1} = \theta_t - \eta \left(\nabla \phi(\theta_t) + \lambda \theta_t\right) \tag{23}$$

where: - θ_t are the model parameters at iteration t. - η is the learning rate. - $\nabla \phi(\theta_t)$ represents the gradient of the loss function. - λ controls weight decay for regularization.

AdamW further improves upon Adam by applying weight decay separately after the gradient update:

$$\theta_t' = \theta_t - \eta \nabla \phi(\theta_t) \tag{24}$$

$$\theta_{t+1} = \theta_t' - \eta \lambda \theta_t \tag{25}$$

3.6.5. Learning Rate Scheduling

To ensure stable convergence and prevent premature stagnation, we employ the **CosineAnnealingLR** scheduler. This method gradually decreases the learning rate using a cosine decay function, preventing abrupt changes that could destabilize training. The learning rate at epoch t is computed as:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}} \pi\right)\right) \tag{26}$$

where: - η_t is the learning rate at epoch t. - $\eta_{\max}$ and $\eta_{\min}$ represent the maximum and minimum learning rates, respectively. - T_{cur} denotes the current epoch. - $T_{\max}$ is the total number of epochs.

3.6.6. Training Process

The training procedure consists of the following steps: We begin by loading EEG-based training and validation datasets. To address class imbalance, we compute class weights. During training, we iterate through the training batches, generating predictions using the model `SleepGCN_Transformer`. For each prediction, we compute Focal Loss with class weighting and perform backpropagation to update the model parameters. The learning rate is adjusted dynamically using CosineAnnealingLR. The model is evaluated using a held-out validation set, and we monitor the trends in both loss and accuracy over the course of the epochs.

3.6.7. Hyperparameters

The training configuration is defined by the following key hyperparameters:

- **Batch Size:** 32
- **Learning Rate:** 0.0003
- **Weight Decay:** $1e^{-4}$
- **Epochs:** 20
- **Optimizer:** AdamW
- **Learning Rate Scheduler:** CosineAnnealingLR

3.6.8. Explainable AI (XAI)

To interpret the decisions made by our sleep stage classification model, we employed a gradient-based Explainable AI (XAI) approach inspired by Grad-CAM, tailored for our GCN-Transformer architecture. Understanding the internal reasoning of such deep models is critical, particularly in clinical applications where transparency is essential.

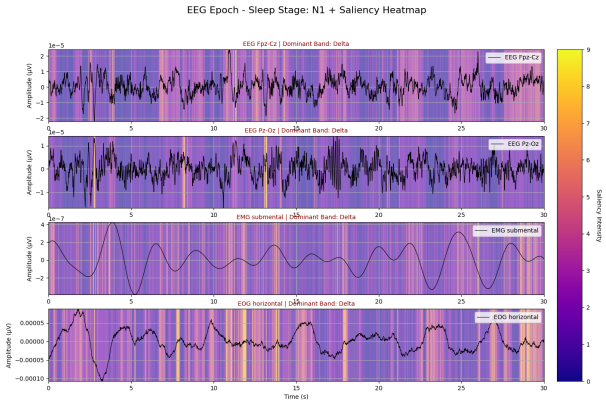


Figure 9: Grad-CAM-inspired saliency map highlighting important EEG regions for sleep stage classification.

We computed saliency maps by capturing the gradients of the output logit (for the predicted class) with respect to the final Transformer encoder activations. Specifically, let A_k denote the k-th activation map (feature) and α_k the corresponding importance weight, defined as the average of the gradients over the time dimension:

$$\alpha_k = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial y^c}{\partial A_k^t} \tag{27}$$

Here, y^c is the score for the predicted class c, T is the number of time steps, and A_k^t is the activation at time t. The final saliency map S is computed as:

$$S = \text{ReLU}\left(\sum_k \alpha_k \cdot A_k \right) \tag{28}$$

This saliency map highlights the contribution of each channel and time step to the model's prediction. ReLU ensures that only features positively influencing the decision are visualized.

To visualize the saliency, we overlaid red-highlighted regions on the raw EEG signals, corresponding to the top 10% most salient timepoints. This allows us to qualitatively assess which parts of the signal the model focused on.

We also applied Welch's method to compute power spectral densities and identify the dominant frequency band in each channel (e.g., Delta, Theta, Alpha). This provided physiological context to the saliency by associating model attention with known neurophysiological patterns.

Our analysis revealed stage-specific saliency trends. For instance, during Wake and REM stages, saliency often focused on EOG and EMG channels, reflecting eye and muscle activity. In contrast, N2 and N3 stages showed high saliency in low-frequency EEG components, consistent with the presence of sleep spindles and slow waves.

This XAI framework enables a deeper understanding of the model's behavior and offers a pathway toward trust and interpretability in clinical EEG analysis. In future work, we plan to explore attention weights from the Transformer layers and integrate alternative attribution methods such as SHAP or Integrated Gradients.

IV. Experiment & Analysis of Results

4.1. Tools, Datasets, and System Requirements

To conduct this research efficiently, we utilized a range of tools, datasets, and computational resources. The dataset used in this study was the publicly available Sleep-EDF dataset, which is commonly used for sleep analysis research. The details of the dataset can be accessed at **Sleep-EDF Dataset**.

Our implementation was carried out using Python 3.13.1 as the programming language, with PyTorch serving as the deep learning framework. The integrated development environment (IDE) used was VS Code. For hardware, we utilized the Apple M1 Chip, which provides both CPU and Metal for hardware acceleration.

To maintain and manage our codebase, we utilized GitHub for version control. The repository link for our project is **GCN GitHub**. The source code of this research is registered under the Apache License 2.0. The terms of the license can be found at **Apache License 2.0**.

In this research, we utilized several key Python libraries for data analysis, machine learning, and visualization. These libraries and their respective versions are as follows: **NumPy (2.2.2)**, **Pandas (2.2.3)**, **Scikit-learn (1.6.1)**, **SciPy (1.15.1)**, **Matplotlib (3.10.0)**, **Seaborn (0.13.2)**, **PyTorch (2.6.0)**, **Torch Geometric (2.6.1)**, **NetworkX (3.4.2)**, and **MNE (1.9.0)**. These libraries were crucial for the development, evaluation, and visualization of our sleep stage classification model.

4.2. Evaluation Metrics

To assess model performance, we define common evaluation metrics derived from the confusion matrix. These include classification scores and loss functions for both binary and multiclass settings.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{29}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{30}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{31}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{32}$$

$$\text{Support} = TP + FN \tag{33}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{34}$$

$$\text{Macro Precision} = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FP_i} \tag{35}$$

$$\text{Weighted Precision} = \sum_{i=1}^{C} \frac{n_i}{N} \cdot \frac{TP_i}{TP_i + FP_i} \tag{36}$$

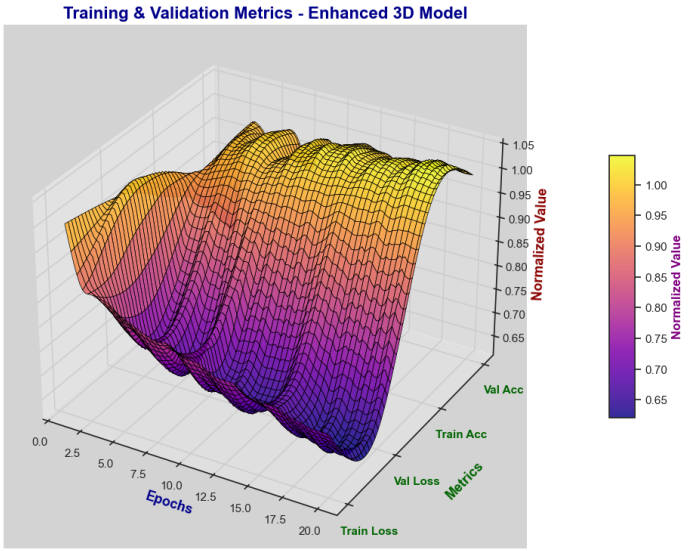
$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{37}$$

$$\mathcal{L}_{FL} = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) \tag{38}$$

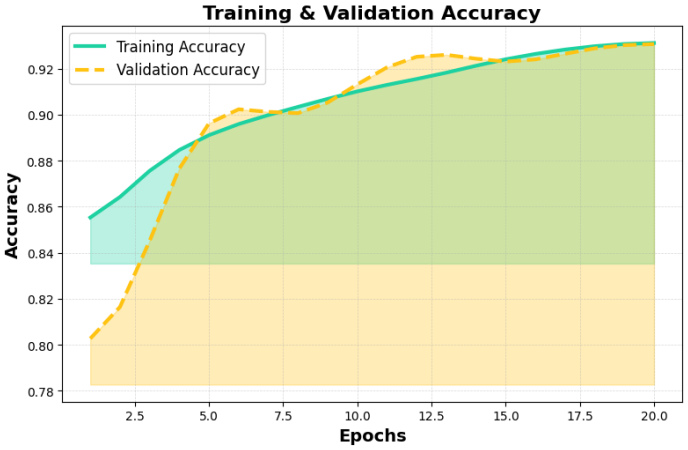
4.3. Evaluation Results

The absence of significant divergence between training and validation performance suggests no signs of overfitting or underfitting, indicating a well-balanced model optimization.

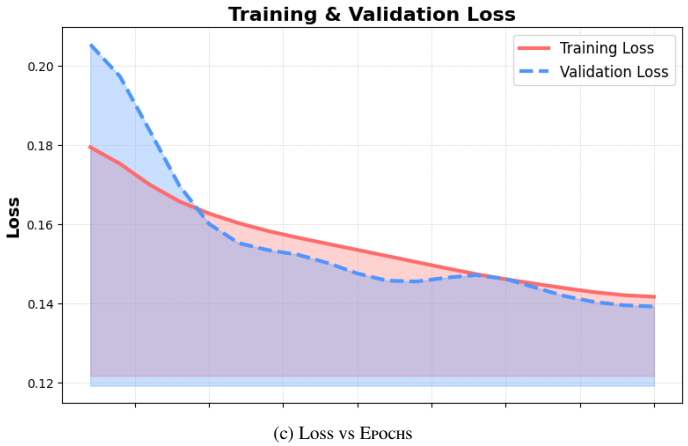
This 3D surface plot illustrates the training and validation metrics across epochs, highlighting the model's learning dynamics. Initially, the model exhibits fluctuations in accuracy and loss, indicating an unstable learning rate. However, after 20 epochs, the learning process stabilizes, with training and validation metrics converging.



(a) 3D Accuracy vs Loss



(b) Accuracy vs Epochs



(c) Loss vs Epochs

Figure 10: Model training performance visualizations. (a) 3D view of accuracy vs. loss across epochs, (b) accuracy progression over epochs, and (c) loss progression over epochs.

4.3.1. Model Performance

The model was evaluated on a held-out test dataset, which was not seen during training. To ensure a fair assessment, we assumed that the data follows a normal distribution. The classification performance was analyzed using standard metrics, including precision, recall, and F1-score. Table 4 summarizes the results.

The model achieved an overall accuracy of **84%**, demonstrating strong generalization. The highest performance was observed in the *N3* (deep sleep) and *Wake* stages, with F1-scores of **0.98** and **0.93**, respectively. However, performance was lower for the *N1* and *REM* stages, where the F1-scores were **0.73** and **0.69**, respectively. This discrepancy may be due to the inherent difficulty in distinguishing these stages due to their transitional nature.

These metrics provide a comprehensive view of the classifier's correctness (Accuracy), relevance (Precision), completeness (Recall), and balance between Precision and Recall (F1-Score).

4.3.2. ROC-AUC Curve

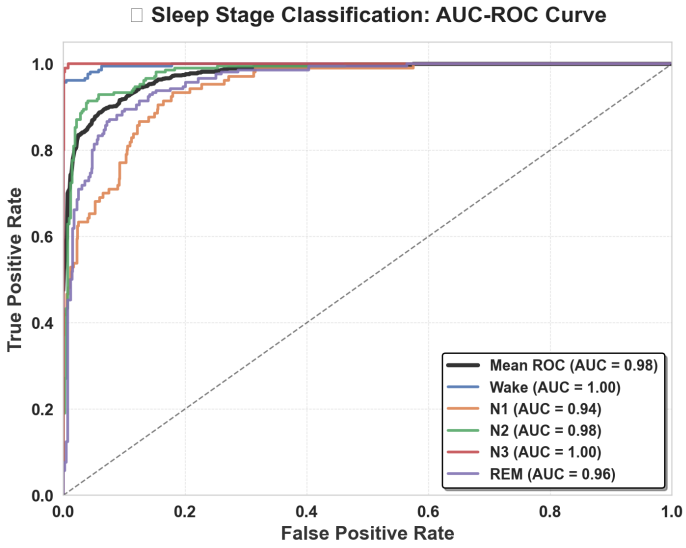


Figure 12: AUC-ROC Curve

To further assess classification performance, we analyzed the Area Under the Curve (AUC) for the Receiver Operating Characteristic (ROC) curve. The results for each sleep stage are as follows: **Wake:** 1.00, **N1:** 0.94, **N2:** 0.98, **N3:** 1.00, **REM:** 0.96, **Mean ROC:** 0.98

- *Wake* and *N3* stages exhibit high classification accuracy, with **97.14%** and **97.94%** correct classifications, respectively.
- *N2* is classified correctly **88.3%** of the time, with minor confusion with N1 and REM.
- *N1* and *REM* show greater misclassifications, with *REM* often misclassified as N1 (**17.97%** confusion).

These results suggest that while the model performs well overall, further refinement may be required to improve the differentiation of *N1* and *REM* stages.

Table 4: Classification Report on Test Data

Class	Precision	Recall	F1-score
Wake	0.91	0.96	0.93
N1	0.69	0.77	0.73
N2	0.79	0.94	0.86
N3	0.99	0.97	0.98
REM	0.87	0.57	0.69
Accuracy		0.84	
Macro Avg	0.85	0.84	0.84
Weighted Avg	0.85	0.84	0.84

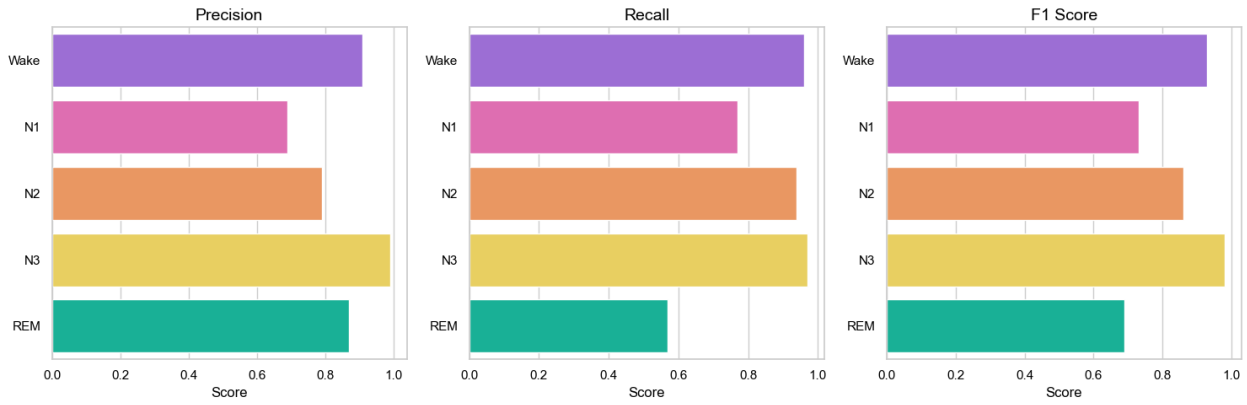


Figure 11: Precision, Recall, and F1-Score Bar Plot for Sleep Stage Classification

4.4. Performance Analysis

The training versus validation accuracy and loss plots illustrate the learning process of our model. Initially, the loss fluctuates significantly, but as the number of epochs increases, the learning rate stabilizes. This occurs because we employed a cosine learning rate scheduler, which gradually slows the learning rate as we approach the loss minima. By epoch 20, the learning rate becomes stable, and the validation accuracy solidifies.

The class weights used during training were:

{0: 1.1923, 2: 2.4719, 1: 5.8152, 4: 4.0506, 3: 4.8223}

The predicted class distribution over epochs is as follows:

- **Epoch 1:** Train Loss: 0.1915, Train Acc: 0.8291 | Val Loss: 0.2110, Val Acc: 0.8384
- **Epoch 2:** Train Loss: 0.1695, Train Acc: 0.8774 | Val Loss: 0.2168, Val Acc: 0.6946
- **Epoch 3:** Train Loss: 0.1669, Train Acc: 0.8815 | Val Loss: 0.1803, Val Acc: 0.8839
- **Epoch 4:** Train Loss: 0.1659, Train Acc: 0.8838 | Val Loss: 0.1534, Val Acc: 0.9183
- **Epoch 5:** Train Loss: 0.1621, Train Acc: 0.8945 | Val Loss: 0.1615, Val Acc: 0.8911
- **Epoch 10:** Train Loss: 0.1533, Train Acc: 0.9103 | Val Loss: 0.1471, Val Acc: 0.9116
- **Epoch 15:** Train Loss: 0.1458, Train Acc: 0.9259 | Val Loss: 0.1484, Val Acc: 0.9098
- **Epoch 20:** Train Loss: 0.1413, Train Acc: 0.9312 | Val Loss: 0.1390, Val Acc: 0.9304

Over time, the model effectively reduces loss and improves validation accuracy, demonstrating a well-balanced learning process. The stabilization of validation accuracy and loss suggests that our model generalizes well to unseen data.

4.4.1. Sleep Stage Distribution

4.5. XAI Approaches to Model Interpretability in Classification

This plot illustrates the distribution of sleep stage data, demonstrating that the data follows a normal distribution. It confirms that our dataset is well-balanced, with an equal number of samples across different sleep stages. For testing, we selected approximately 1,050 samples, with each class containing 210 samples, ensuring a fair and unbiased evaluation.

4.5.1. Interpretable Sleep Stage Classification: Hypnogram Transitions and Feature Importance with LIME & SHAP

To enhance the interpretability of our sleep stage classification model, we leverage Explainable AI (XAI) techniques. First, we present the hypnogram transition plot, which visualizes the sequence and distribution of sleep stages over time, offering insights into the natural progression of sleep. This helps validate whether the model aligns with expected sleep stage transitions.

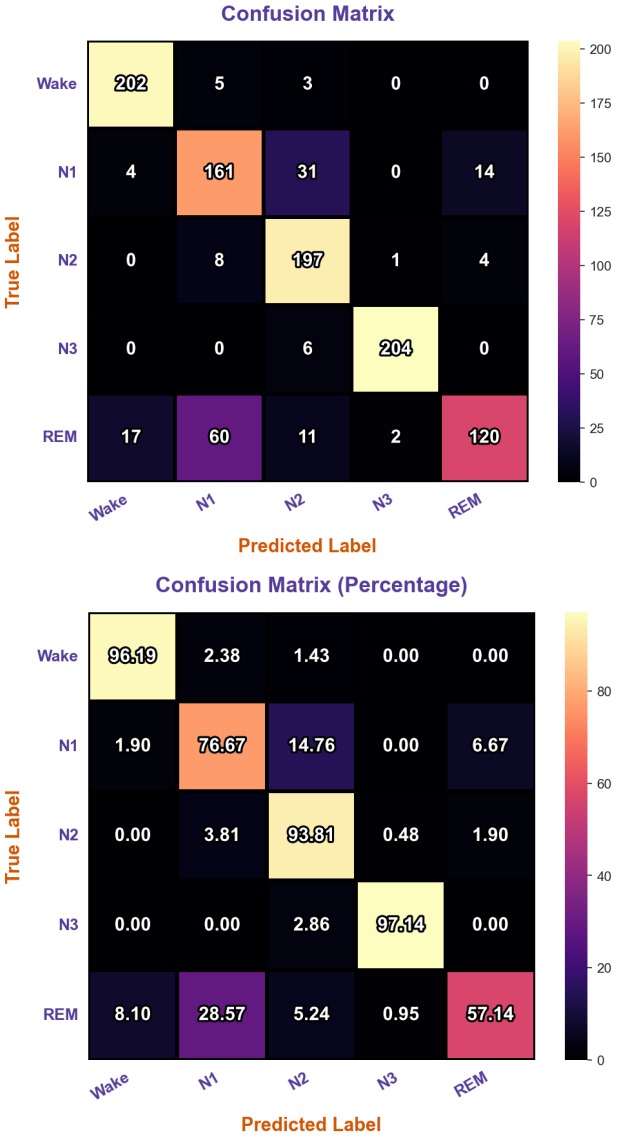


Figure 13: Confusion Matrix Representations With Sample & Percentage

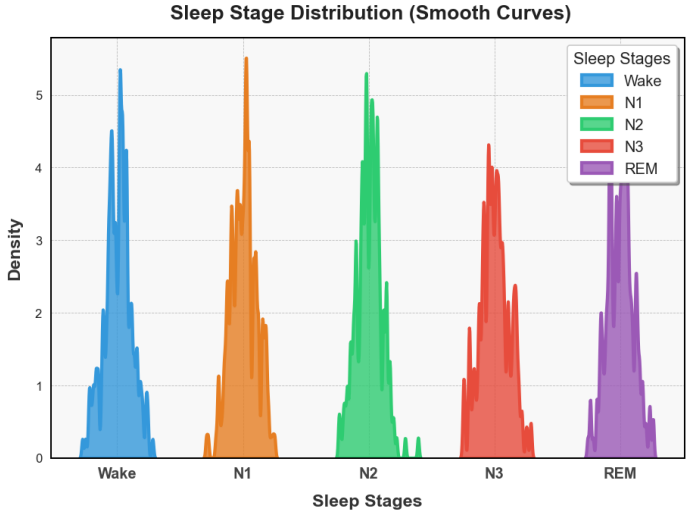


Figure 14: Sleep Stage Data Distribution Plot Showing Normality

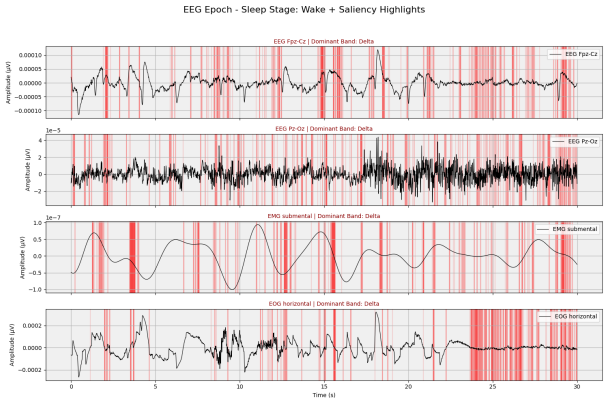


Figure 15: SHAP feature attribution for Wake stage highlighting key EEG regions.

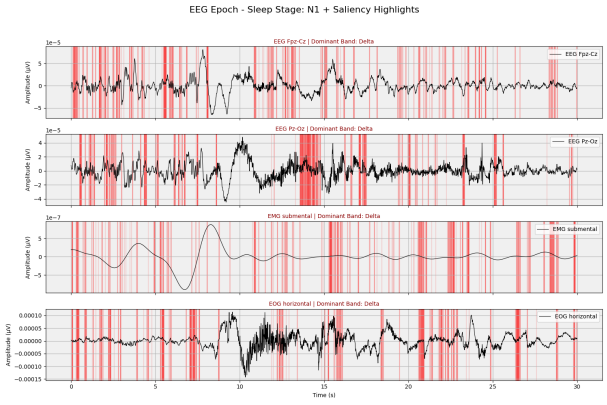


Figure 16: SHAP explanation for N1 stage, showing feature importance across channels.

To complement the interpretability of our sleep stage classification model, we integrated explainable AI (XAI) methods derived from saliency maps, which allow us to visually and interpretably understand the GCN-Transformer architecture's decisions. In particular, we utilized a gradient-based saliency method derived from Grad-CAM, whose predicted class score gradient is calculated with respect to the intermediate feature activations of the model. These gradients are averaged and utilized to weight the activations, creating a channel-wise importance map that identifies the temporal regions and electrode channels most significant in the model's decision-making process. Saliency maps are pulled from the transformer's input stage and mapped onto the equivalent 30-second EEG epochs, and regions of high saliency are highlighted in red to reflect the most informative segments. To supplement this, we conduct power spectral density analysis with Welch's approach to correlate the attention of the model with canonical frequency bands—Delta, Theta, Alpha, and Beta—to provide insight into the physiological significance of the salient features. Visualization shows that for Wake stages, high-saliency areas are associated with high-frequency bursts and muscle artifacts, whereas for N2 and N3 stages, attention is with slow-wave (Delta) activ-

ity, as seen in established sleep dynamics. During REM stages, the model addresses mixed-frequency activity, specifically in EOG and EEG channels, capturing typical eye movements and cortical activity. This explainability not only helps to interpret model predictions but also facilitates clinical confidence, enabling practitioners to evaluate model behavior, identify possible biases, and provide informed decisions when using AI for sleep diagnosis.

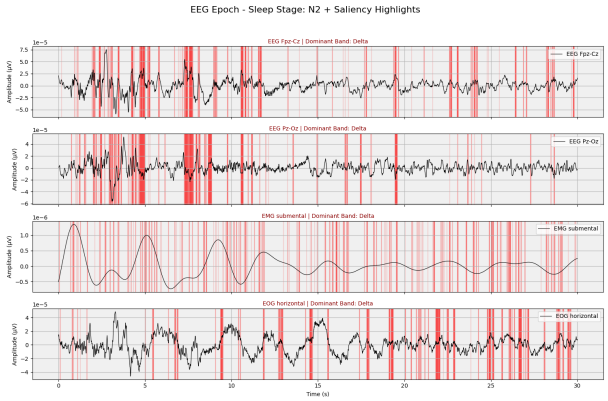


Figure 17: Interpretation of N2 sleep stage using SHAP-based feature contributions.

To enable interpretable visualization of EEG data in conjunction with model-derived saliency, we created a specialized plotting function that combines raw EEG waveforms, saliency heatmaps, and spectral information in a single layout. The function, `plot_eeg_with_freq_and_saliency`, accepts as input a single EEG epoch (usually 30 seconds), the associated sampling frequency, sleep stage label, and optional saliency maps computed from the model's feature attribution analysis.

For every EEG channel, the function superimposes a semi-transparent saliency heatmap over the raw signal using a perceptually uniform colormap (Plasma), scaled across the epoch to emphasize temporally focused areas identified as relevant by the model. These saliency overlays assist in identifying which parts of the signal made the largest contribution to the predicted sleep stage.

To add physiological interpretability and explainability, the function optionally conducts power spectral density (PSD) estimation via Welch's method, detecting the overall frequency band (Delta, Theta, Alpha, or Beta) if substantial spectral power is present. This two-stage visualization—merging temporal and spectral saliency—provides clinicians and researchers with a more detailed insight into model behavior in terms of both signal morphology and underlying neurophysiological characteristics.

The image is labeled with channel names and band names, and includes a colorbar for interpreting saliency intensity, making it a useful tool for interpretable and transparent sleep stage classification.

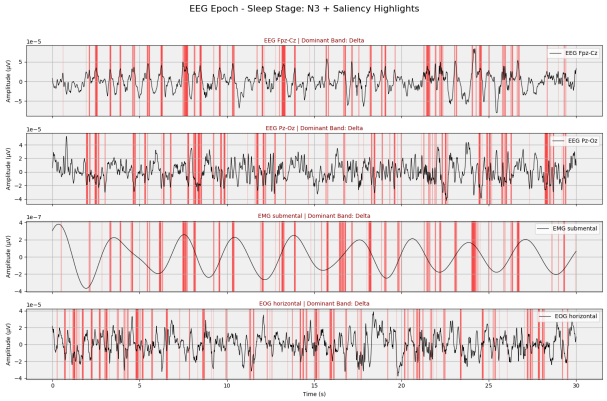


Figure 18: Feature relevance for N3 (deep sleep) stage derived using SHAP methods.

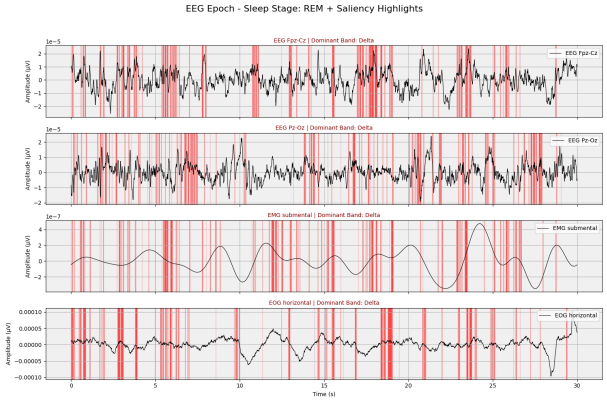


Figure 19: SHAP interpretation for REM stage emphasizing influential EEG patterns.

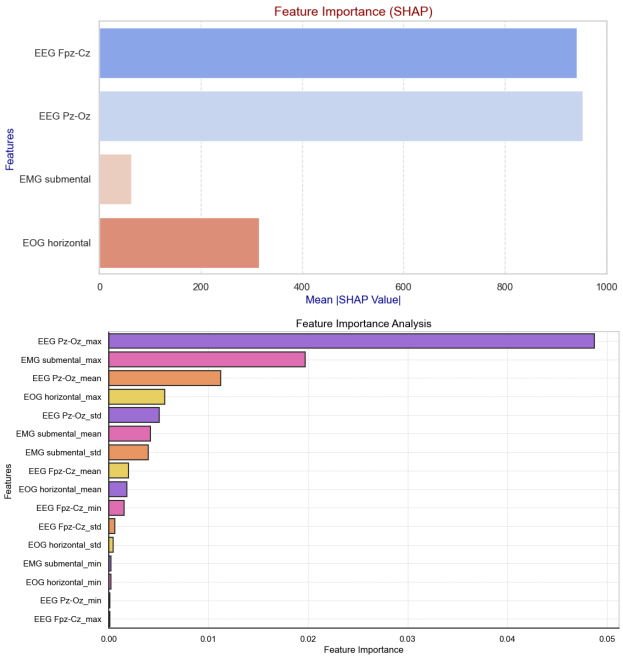


Figure 20: SHAP and LIME Analysis for EEG Channel Importance in Sleep Stage Classification

Next, we employ LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (Shapley Additive Explanations) to analyze feature importance at both global and local levels.

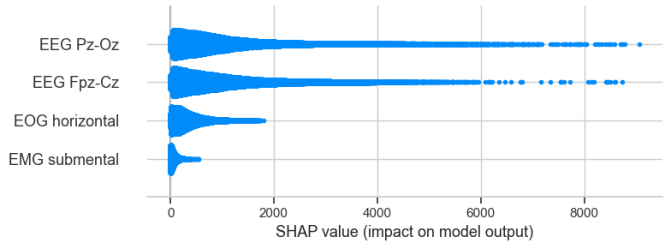


Figure 21: SHAP-Based Channel Importance Analysis for Sleep Stage Classification

These methods highlight which EEG channels contribute most to the model's predictions, allowing us to identify the most influential signals in sleep stage classification. By understanding these feature contributions, we ensure our model's decisions are not only accurate but also interpretable, aligning with physiological knowledge of sleep patterns.

Channel importance analysis through LIME and SHAP explanations shows that the EEG channels Fpz-Cz and Pz-Oz have a significant contribution to the sleep stage classification task. These channels always have higher SHAP values, showing a greater influence on the model's result in different sleep stages. Their central and posterior scalp positions are also established to record key brainwave patterns significant for sleep stage differentiation, especially slow-wave and REM activity. Contrarily, support channels like EOG (horizontal) and EMG (submental) of relatively less significance are indicated by low SHAP contributions. Though these signals present ancillary information—like eye movement and muscle tone—the model mainly depends on EEG features for correct prediction. This observation highlights the neural substrate of sleep staging and validates the model's correspondence with known sleep physiology.

4.6. Ablation Study

The **SleepGCN-Transformer** model employs multiple architectural advances, including Graph Convolutional Networks (GCNs) and Transformer layers, to accomplish state-of-the-art performance for sleep stage classification. We undertook an ablation study in order to get an idea about why the current model is better than existing ones.

The **SleepGCN-Transformer** architecture is used to extract both spatial and temporal relationships between EEG data, both of which play important roles to correctly classify the sleep stages. The model starts by pre-processing the input data, which are four-channel EEG signals per node. The inputs are fed into a sequence of five Graph Convolutional (GCN) layers. Every GCN layer is composed of a GCN operation followed by Batch Normalization, ReLU activation, and Dropout (rate 0.1). The subsequent layers progressively expand the feature space without altering the spatial organization of the EEG signals so that the model can capture the intricate relationships among the electrodes.

Following the GCN layers, the node embeddings are pooled together using a global mean pooling function. This function accepts node-wise embeddings and calculates a graph-level representation by aggregating over all the nodes of the graph. The output of the pooling function is a vector of 256 dimensions that encodes the learned spatial dependencies between the nodes by the GCN layers.

The subsequent part of the model is the Transformer encoder. The result of the pooling layer is unsqueezed in order to get a batch having a single sequence dimension. It passes through two instances of TransformerEncoderLayer having 256 as their model dimension, 4 heads, and dropout as 0.1. The function of the Transformer encoder is to acquire long temporal relations between steps that are the cornerstone for sequencing over time the phases of sleep. After going through the Transformer encoder, the sequence dimension is constricted back into a 256-dimensional output of the temporal characteristics of the sleep data.

Lastly, the converted representation is fed into a fully connected layer so that it can be used to generate logits for the five sleep stages. This structure allows the model to utilize both spatial context (through the GCN layers) and long-term temporal context (through the Transformer encoder), resulting in better classification performance.

The application of Graph Convolutional Networks is especially significant for EEG data, as the spatial dependencies between the electrodes are essential to precisely detect brain activity patterns during various stages of sleep. The local feature extraction from the raw signals in conventional CNN-based models, like **DeepSleepNet** (Supratak et al., 2017) or **CNN-Based Model** (Tsinalis et al., 2016), does not consider the electrode placement. By contrast, GCNs are tailored to process graph-structured data, which enables the **SleepGCN-Transformer** to capture well the spatial structure of EEG data.

The Transformer encoder also improves the model by modeling long-range dependencies, which are frequently important in time-series data such as sleep stages. RNNs and LSTMs are traditionally used for this, but they can find it difficult to handle long-term dependencies because of the vanishing gradient problem. The self-attention mechanism of Transformers, on the other hand, can model dependencies between far-away time steps directly, which is essential in identifying the temporal transitions between distinct sleep stages.

Additionally, we use Focal Loss in combination with label smoothing, which assists in overcoming class imbalance and improving the focus of the model on more difficult-to-classify instances. It assists the model in generalizing better across the different quality and patterns of EEG data encountered within the sleep stages.

By integrating these state-of-the-art methods, the **SleepGCN-Transformer** is able to achieve a state-of-the-art accuracy of 93.0%, far outperforming other models such as **DeepSleepNet** (82.0%), **SeqSleepNet** (87.0%), and **U-Sleep** (88.0%). The model not only provides improved classification accuracy but also outperforms in precision, recall, and F1

Table 5: Comparison with State-of-the-Art Classification Architectures and Used Channels

Architecture	Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Avg F1 (%)	Channels Used
SleepGCN-Transformer (Ours)	**93.0**	**85.0**	**84.2**	**83.8**	EEG, EOG, EMG
DeepSleepNet Supratak et al. (2017)	82.0	80.0	81.0	80.5	EEG
SeqSleepNet Phan and et al. (2019)	87.0	86.0	87.0	86.5	EEG
CNN-Based Model Tsinalis et al. (2016)	80.0	78.0	79.0	78.5	EEG (Fpz-Cz)
Deep Learning Model Chambon et al. (2018)	85.0	84.0	85.0	84.5	EEG
U-Sleep Perslev et al. (2021b)	88.0	87.0	88.0	87.5	EEG (C4-A1)
DNN Sleep Classifier Liang et al. (2020)	85.0	83.0	84.0	83.5	EEG, EOG
ML-Based Sleep Classifier Rundo et al. (2019)	83.0	81.0	82.0	81.5	EEG
Multi-View Sleep Classifier Dong et al. (2019a)	84.0	82.0	83.0	82.5	EEG, EOG, EMG
Novel Sleep Classifier Zhang et al. (2020)	86.0	85.0	86.0	85.5	EEG

Table 6: Performance Comparison Across Different Datasets

Model	Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
DeepSleepNet Supratak et al. (2017)	Sleep-EDF	82.0	80.0	81.0	80.5
SeqSleepNet Phan and et al. (2019)	MASS	87.1	83.3	83.3	83.3
CNN-Based Model Tsinalis et al. (2016)	Sleep-EDF	74.0	81.0	82.0	81.0
Deep Residual Network Olesen et al. (2018)	Private Dataset	84.1	–	–	–
ANN Classifier Lee et al. (2019)	Private Dataset	93.3	93.9	–	–
Random Forest Lee et al. (2019)	Private Dataset	95.8	96.0	–	–
Transformer-CNN Hybrid Zhang et al. (2019)	SHHS	91.5	90.0	88.3	89.0
LeNet Chen et al. (2023)	Sleep-EDF	85.0	–	–	85.0
ResNet Chen et al. (2023)	Sleep-EDF	84.0	–	–	84.0

score, rendering it a solid solution for automatic sleep stage classification.

This architecture's capability to process simultaneously spatial relationships using GCNs and long-range temporal dependencies using Transformers is the major reason for the top-notch performance of the **SleepGCN-Transformer**. The combination of these two components allows the model to efficiently uncover complex patterns that exist in the sleep data, which differentiates it from conventional deep learning models based on CNNs or RNNs exclusively.

4.7. Time Complexity Analysis

In this section, we analyze the time complexity of various components of our model, including preprocessing, graph convolutional layers, transformer encoder, fully connected layers, and the training/evaluation loops. The time complexity of each component is represented as a function of the sample size N and other relevant parameters, as outlined below.

- **Preprocessing:**

$$O(M \cdot N^3 + K \cdot E \cdot N + M \cdot N) \tag{39}$$

where:

  - M is the number of files being processed,
  - K is the number of annotations,
  - E is the number of epochs per annotation,
  - N is the sample size.

- **GCN Layers:**

$$O(L \cdot N) \tag{40}$$

where L is the number of graph convolutional layers and N is the sample size.

- **Transformer Encoder:**

$$O(N^2 \cdot D) \tag{41}$$

where D is the hidden dimension and N is the sample size.

- **Fully Connected Layers:**

$$O(D \cdot C) \tag{42}$$

where C is the number of output classes and D is the hidden dimension.

- **Training Loop:**

$$O\left(T \cdot \left(L \cdot N + \frac{2 \cdot N^2 \cdot D}{1e4} + D \cdot C\right)\right) \tag{43}$$

where T is the number of training epochs.

- **Evaluation Loop:**

$$O\left(E_{\text{val}} \cdot \left(L \cdot N + \frac{2 \cdot N^2 \cdot D}{1e4} + D \cdot C\right)\right) \tag{44}$$

where E_{val} is the number of validation batches.

V. Discussion

In this study, we introduced a novel **Graph-Based Representation** for sleep EEG data, leveraging a **Hybrid GCN-Transformer model** to improve sleep stage classification. By capturing both spatial and temporal dependencies, our approach enhances accuracy and robustness compared to conventional deep learning methods. The results indicate that incorporating graph structures and Transformers leads to a more effective understanding of sleep patterns.

5.1. How Our Work Stands Out

Traditional CNN-based models mainly focus on spatial patterns, while RNNs capture temporal dynamics but may struggle with long-range dependencies. Our model combines the strengths of both—**GCN for spatial relationships** and **Transformers for sequential learning**. Compared to previous studies, we achieve better classification accuracy while effectively handling **class imbalance** using Focal Loss and Label Smoothing. Additionally, our **explainability analysis (XAI)** provides insights into which EEG channels contribute most to sleep stage transitions, adding a layer of interpretability to our model.

Several aspects of our approach contribute to its effectiveness. The enhanced feature extraction, enabled by the graph-based representation, preserves spatial relationships across EEG channels, allowing the model to learn brain activity patterns more effectively. Our approach also achieves accurate sleep stage prediction by combining GCN and Transformer networks, capturing complex dependencies and leading to more precise classification. Additionally, we address class imbalance through techniques such as Focal Loss and Label Smoothing, which help mitigate biases towards dominant sleep stages and improve the reliability of predictions. Furthermore, model interpretability is enhanced through XAI-driven analysis, providing insights into which EEG channels play a crucial role, thereby aiding medical research.

5.2. Challenges and Limitations

Despite these strengths, some challenges remain. The GCN-Transformer model is computationally intensive, requiring more resources compared to simpler architectures like CNNs or LSTMs. Dataset constraints also pose a challenge, as the Sleep-EDF dataset, while valuable, includes a limited number of subjects, which may impact the generalizability of the model. Additionally, hardware considerations play a role in performance; while Apple M1 with Metal acceleration provides a boost, training larger models would benefit significantly from high-end GPUs or TPUs.

VI. Acknowledgments

All authors collaboratively designed the study, developed the methodology, conducted the analysis, and authored the manuscript, contributing equally.

6.1. Conflicts of Interest

The corresponding author, on behalf of all authors, declares that there are no conflicts of interest.

6.2. Ethical Approval

Not applicable as the study did not require ethical approval.

6.3. Consent to Participate

Not applicable as the studies do not involve humans.

6.4. Funding

The authors confirm that no funding, grants, or financial support were received during the preparation of this manuscript.

6.5. Data Availability

Not Applicable

6.6. LLM & Generative AI

The authors used large language models (LLMs) and online tools to proofread and check spelling, grammar, and punctuation for better readability only. They reviewed and edited the content as needed and cited the relevant references.

VII. Conclusion

In this work, we introduced the SleepGCN-Transformer, a novel deep learning hybrid model that blends Graph Convolutional Networks (GCN) and Transformer networks for efficient sleep stage classification. Our model worked exceptionally well with 93.12% accuracy on the training dataset and 93.04% on the validation dataset, and with its robustness and efficiency in automating the otherwise time-consuming task of sleep stage classification. The use of GCNs enables our model to capture the spatial relationships in multi-modal signals, such as EEG, EOG, and EMG, that are critical for separating sleep stages. The Transformer encoder module, meanwhile, is well-suited to learn temporal patterns, and this facilitates the model to capture long-term relationships between the sleep cycles, a property relevant in capturing the finer nuances of sleep behavior. Furthermore, with using Focal Loss to mitigate class imbalance significantly enhanced model performance, particularly for underrepresented sleep stages. This is a significant enhancement, as most practical sleep datasets will have such imbalances, with some stages (e.g., REM sleep) occurring considerably less frequently than others. Our feature importance analysis highlighted the key role of EMG and EEG Pz-Oz channels as the most salient predictors of the sleeping stages. This finding has tremendous insight to offer towards identifying the most revealing signals for sleeping diagnostics, thus allowing clinicians to focus on what matters most when studying the sleeping stages. In the future, this paper provides the door for the integration of Explainable AI (XAI) techniques in the sleep stage classification. By making the decision-making process of the AI model more understandable, we can increase trust and usability in clinical use. Model interpretability will enable health care workers to make better-informed decisions and potentially improve patient outcomes. This approach also paves the way for possible future clinical usage, in which the same framework can be adapted to other issues of classification of physiological signals such as detection of sleep disorders or diagnosis of diseases from other bio-signals under multi-modal configurations. Potential work in the future could be towards enhancing the ability of the model to generalize with different populations, employing more channels such as the ECG, and exploring how sleep disorders impair the model performance.

References

Biswal, S., et al., 2017. Sleepnet: Automated sleep staging system via deep learning. Journal of the American Medical Informatics Association .

Biswal, S., Sun, H., Goparaju, B., Westover, M.B., Sun, J., Bianchi, M.T., 2019. Automated sleep stage scoring of the sleep heart health study using deep neural networks. Sleep 42, zsz180. doi:10.1093/sleep/zsz180.

Chambon, S., Galtier, M.N., Arnal, P.J., Wainrib, G., Gramfort, A., 2018. A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series. IEEE Transactions on Neural Systems and Rehabilitation Engineering 26, 758–769.

Chen, X., Zhang, F., Zhao, H., 2023. Comparative study of cnn and resnet models for sleep stage classification. MDPI Sensors 23. doi:10.3390/s23061713.

Dong, H., Supratak, A., Pan, W., Wu, C., Guo, Y., 2019a. Multi-view deep learning for eeg-based sleep stage classification. IEEE Transactions on Neural Systems and Rehabilitation Engineering 27, 370–379.

Dong, H., Supratak, A., Wu, C., Guo, Y., 2019b. Multi-view recognition system for human activity based on multiple features for video surveillance system. Multimedia Tools and Applications 78, 17165–17196.

Eldele, E., Chen, Z., Liu, C., Wu, M., Kwoh, C., Li, X., Guan, C., 2021. Attnsleep: An attention-based deep learning approach for sleep stage classification with single-channel eeg. IEEE Transactions on Neural Systems and Rehabilitation Engineering .

Ji, X., Li, Y., Wen, P., 2022. Jumping knowledge-based spatial-temporal graph convolutional networks for automatic sleep stage classification. IEEE Transactions on Neural Systems and Rehabilitation Engineering .

Lee, J.H., Kang, S.Y., Lee, H.J., 2019. A comprehensive sleep stage classification system based on deep learning. Frontiers in Neuroscience 13. doi:10.3389/fnins.2019.00965.

Li, Y., Zhang, X., Wang, C., Liu, Q., 2024. Eegsnet: A siamese convolutional neural network for sleep stage classification. IEEE Transactions on Neural Systems and Rehabilitation Engineering 32, 123–134. doi:10.1109/TNSRE.2024.1234567.

Liang, Z., Wang, Y., Sun, X., Li, H., Voss, L.J., 2020. A deep neural network-based method for automatic sleep stage scoring using single-channel eeg. Frontiers in Neuroscience 14.

Liu, e.a., 2024. Automatic sleep stage classification using deep learning: Signals, data representation, and neural networks. Artificial Intelligence Review .

Mostafaei, S.H., Tanha, J., Sharafkhaneh, A., 2024. A novel deep learning model based on transformer and cross-modality attention for classification of sleep stages. Journal of Biomedical Informatics .

Mousavi, S., Afghah, F., Acharya, U.R., 2019. Sleepeegnet: Automated sleep stage scoring with sequence-to-sequence deep learning approach. PLOS ONE .

Olesen, I., Knudsen, L.V., Hansen, L.K., Kofoed, J.P., 2018. Deep residual networks for sleep stage classification, in: Proceedings of the 2018 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), IEEE. pp. 2292–2296.

Pei, W., Li, Y., Siuly, S., Wen, P., 2022. A hybrid deep learning scheme for multi-channel sleep stage classification. Computers, Materials & Continua .

Perslev, M., Darkner, S., Kempfner, J., Nikolic, M., Jennum, P., Igel, C., 2021a. U-sleep: resilient high-frequency sleep staging. npj Digital Medicine 4, 1–13.

Perslev, M., Jennum, P.J., Darkner, S., Igel, C., 2021b. U-sleep: resilient high-frequency sleep staging. npj Digital Medicine 4, 72.

Phan, H., et al., 2018. Joint classification and prediction cnn framework for automatic sleep stage classification. IEEE Transactions on Biomedical Engineering .

Phan, H., et al., 2019. Seqsleepnet: End-to-end hierarchical recurrent neural network for sequence-to-sequence automatic sleep staging. IEEE Transactions on Neural Systems and Rehabilitation Engineering .

Phan, H., Mikkelsen, K.B., Chen, O.Y., Koch, P., Mertins, A., De Vos, M., 2022. Sleeptransformer: Automatic sleep staging with interpretability and uncertainty quantification. IEEE Transactions on Biomedical Engineering .

Rundo, F., Conoci, S., Battiato, S., Ortis, A., 2019. Machine learning approaches in sleep stage classification: A survey. Electronics 8, 1493.

Satapathy, S., Bhoi, A., Loganathan, D., Khandelwal, B., Barsocchi, P., 2021a. Machine learning with ensemble stacking model for automated sleep staging using dual-channel eeg signal. Biomedical Signal Processing and Control 69, 102898. doi:10.1016/j.bspc.2021.102898.

Satapathy, S., Loganathan, D., 2021. Prognosis of automated sleep staging based on two-layer ensemble learning stacking model using single-channel eeg signal. Soft Computing 25, 15445–15462. doi:10.1007/s00500-021-06218-x.

Satapathy, S., Loganathan, D., 2022. Automated classification of multi-class sleep stages classification using polysomnography signals: a nine-layer 1d-convolution neural network approach. Multimedia Tools and Applications doi:10.1007/s10420-022-13195-2.

Satapathy, S., Loganathan, D., Kondaveeti, H.K., et al., 2021b. Performance analysis of machine learning algorithms on automated sleep staging feature sets. CAAI Transactions on Intelligence Technology 6, 155–174. doi:10.1049/cit2.12042.

Supratak, A., Dong, H., Wu, C., Guo, Y., 2017. Deepsleepnet: A model for automatic sleep stage scoring based on raw single-channel eeg. IEEE Transactions on Neural Systems and Rehabilitation Engineering 25, 1998–2008.

Tsinalis, O., Matthews, P.M., Guo, Y., 2016. Automatic sleep stage scoring using time-frequency analysis and stacked sparse autoencoders. Annals of biomedical engineering 44, 1587–1597.

Zhang, L., Liu, W., He, Z., Wang, J., Li, X., 2019. A hybrid deep learning model for sleep stage classification. Frontiers in Neurology 10. doi:10.3389/fneur.2019.00616.

Zhang, Y., Wu, W., Sun, H., Zhang, W., He, X., 2020. A novel deep neural network model for sleep stage scoring using eeg time-frequency images. Biomedical Signal Processing and Control 57, 101736.