TITLE: ***Integrated CNN-BiLSTM-Attention Framework for Classification and Prediction of Epilepsy Focus Using EEG Signals***

Team Members: Thiago Henrique Rizzi Donato and Batoulsadat Haerian

1. INTRODUCTION

Epilepsy, a complex neurological disorder characterized by recurrent, unprovoked seizures, significantly affects millions worldwide (Wirrell et al., 2022). Electroencephalography (EEG) serves as a cornerstone in diagnosing and managing epilepsy, capturing the brain's electrical activity to differentiate epileptic seizures from non-epileptic events and categorize seizure types (Stafstrom and Carmant, 2015).

Traditional EEG analysis is hindered by its slow speed and susceptibility to human error due to the variability in EEG signal patterns across time and patients. Approximately 20% of epilepsy cases are reported misdiagnosed, highlighting the challenges in EEG interpretation (Leach et al., 2005). Accurate interpretation of EEG data is crucial for refining treatment plans and improving the quality of life for individuals with epilepsy. However, this often relies on visual inspection by experienced neurologists, a time-consuming and inconsistent approach where interpretations may vary among neurologists. Moreover, the scarcity of experienced neurologists and the extensive training needed contribute to delays in diagnosis and treatment planning (Ein Shoka et al., 2023). To address these challenges and enhance diagnostic accuracy, automated seizure detection using artificial intelligence (AI), particularly deep learning (DL) techniques, offers promising solutions. AI enables computers to interpret and respond to real-world information effectively (Soori et al., 2023). Studies have demonstrated the efficacy of hybrid CNNs, LSTM networks, and attention mechanisms in classifying diseases like epilepsy, leveraging the temporal characteristics of EEG signals (Table 1).

This project aims to utilize CNNs, LSTM, and Attention models to automate the classification and prediction of epileptic foci using databases storing structured and unstructured brain activity data, such as spectrograms and EEG signals. Integrating attention mechanisms with CNNs and RNNs enhances model performance by capturing long-term dependencies and highlighting critical features in EEG data. CNNs excel in extracting spatial features from EEG and spectrogram images, identifying seizure characteristics linked to abnormal brain waves. These networks use convolutional and pooling layers for feature extraction while maintaining computational efficiency (Schmierer, 2024). LSTM networks, a variant of RNNs, are adept at processing sequential EEG data, capturing temporal patterns crucial for seizure detection (Pham et al., 2021; Wei et al., 2020). LSTM networks are adept at capturing long-term temporal dependencies, making them invaluable in tackling complex issues in bioinformatics using physiological data. Leveraging gating mechanisms such as input, forget, and output gates, LSTMs selectively retain or discard information, filtering out noise and focusing on significant seizure patterns (Alharthi et al., 2022; Lu et al., 2023). This selective attention enhances their accuracy in predicting seizure foci by robustly handling noisy data.

Additionally, attention mechanisms further augment the model's capabilities by enabling it to concentrate on crucial segments of EEG signals. By identifying and prioritizing relevant features, attention mechanisms provide insights into which aspects of the EEG influence the model's predictions effectively (Kim et al., 2020). Integrating a hybrid CNNs/LSTM/Attention model in this project represents a powerful approach for classifying and predicting epileptic foci. By surpassing the limitations of traditional EEG analysis, this model

enhances the accuracy and efficiency of epilepsy diagnosis and treatment. It promises to streamline the diagnostic process, empower clinicians with precise insights, and improve patient outcomes.

Table 1. A summary of application of DL models in different diseases.

Author	Year	Disease	Data	Architecture	Accuracy (%)
Tsinalis et al.	2016	Sleep	EEG	CNN	86.8
Djemili et al.	2016	Autism	EEG	DBN	85.7
Vidyaratne al.	2016	Epilepsy	EEG	RNN	100
Morabito et al.	2016	Alzheimer's Disease	EEG	Autoencoder + SVM	78.5
Supratak et al.	2017	Sleep	Spectrogram	RNN	88.0
Antoniadeset al.	2017	Epilepsy	EEG	CNN	89.0
Birjandtalab et al.	2017	Epilepsy	EEG	MLPNN	94.2
Schirrmeister et al.	2017	Epilepsy	Spectrogram	CNN	95.6
Oh et al.	2018	Parkinson's Disease	EEG	CNN	79.6,
Liu et al.	2018	Schizophrenia	EEG	CNN-LSTM	83.4
Heinsfeld et al.	2018	Autism	EEG	DBN	85.7
Acharya et al.	2018	Epilepsy	EEG	CNN	92.0
Tsiouris et al.	2018	Epilepsy	EEG	RNN-LSTM	100.0
Ullah et al.	2018	Epilepsy	EEG	CNN	99.0
Wei et al.	2018	Epilepsy	EEG	CNN	90.0
Acharya et al.	2018	Epilepsy	EEG	CNN-LSTM	90.1
Duan et al.	2019	Autism	EEG	CNN-RNN	87.3
Ju et al.	2019	Alzheimer's Disease	EEG	DNN	84.3
Yang et al.	2019	Alzheimer's Disease	EEG	RNN	84.7
Li et al.	2019	Depression	EEG	LSTM	80.2
Hussein et al.	2019	Epilepsy	EEG	RNN-LSTM	100
Roy et al.	2019	Epilepsy	Both	CNN-LSTM-Attention	92.3
Sakar et al.	2019	Epilepsy	EEG	CNN	92.8
Cheng et al.	2020	Alzheimer's Disease	EEG	CNN	85.9
Borah et al.	2020	Depression	EEG	LSTM	81.5
Sakar et al.	2020	Parkinson's Disease	EEG	CNN-RNN	81.2
Liu et al.,	2020	Epilepsy	EEG	CNN-LSTM	95.2
Shah et al.	2020	Epilepsy	EEG	CNN	99.1
Xie et al.	2020	Epilepsy	Both	CNN-LSTM- Attention	97.0
Sannino et al.	2021	Stroke	EEG	CNN-RNN	81.0
Huang et al.	2021	Autism	EEG	RNN	86.5
Li et al.	2021	Depression	EEG	CNN, GRU	83.1
Bhatia et al.	2021	Schizophrenia	EEG	CNN-RNN hybrid	86.7
Singh et al.	2021	Epilepsy	Both	CNN-LSTM- Attention	93.4
Lin et al.	2022	Schizophrenia	EEG	Autoencoder + CNN	88.7
Kim et al.	2022	Epilepsy	EEG	CNN-RNN	94.1
Patel et al.	2022	Epilepsy	Both	LSTM	89.8
Pinto et al.	2022	Epilepsy	Both	CNN-LSTM	94.5
Rao et al.	2022	Epilepsy	EEG	CNN-LSTM-Attention	93.3
Zhao et al.	2023	Schizophrenia	EEG	CNN-LSTM	90.4
Li et al.	2023	Epilepsy	EEG	CNN-BiLSTM-attention	72.2
Wang et al.	2023	Epilepsy	EEG	CNN-LSTM	98.0
Said and Göker	2024	Alzheimer's disease	EEG	LSTM	99.0
Shams and Jabbari	2024	Schizophrenia	EEG	CNN-LSTM	99.0
Xu et al.	2024	Autism	EEG	CNN-LSTM	81.1

2. METHODOLOGY AND ANALYSIS

To develop a robust CNN-BiLSTM-Attention model for identifying seizure foci, it is crucial to leverage EEG signal features effectively. The EEG data is acquired using the standard 10-20 EEG system, where electrodes are strategically placed on the scalp and labeled accordingly: FP for pre-frontal or frontal pole (Fp1 and Fp2), F for frontal (F7, F3, Fz, F4, and F8), C for central (C3, Cz, and C4), T for temporal (T3, T4, T5, and T6), P for parietal (P3, Pz, and P4), and O for occipital (O1 and O2), as depicted in Figure 1.

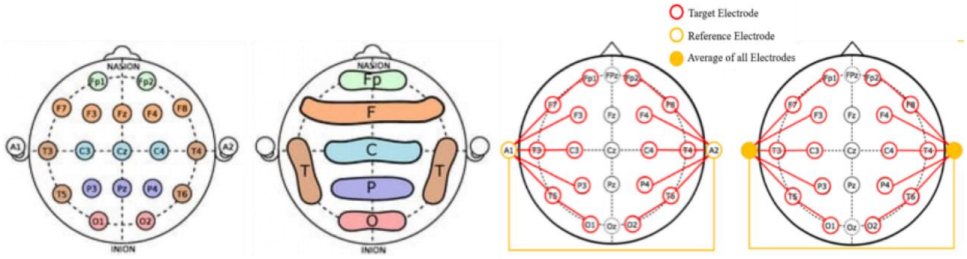

Figure 1. The 19 electrodes and channels in the 10-20 system and corresponding brain regions are represented in the first two left images. The two common referential montages, Linked Ears Reference (ER) and Average Reference (AR), are shown in the last two right images.

These electrode placements adhere rigorously to the standard 10-20 EEG system, ensuring consistency and compatibility with established EEG data acquisition protocols. By leveraging this standardized approach, the CNN-BiLSTM-Attention model can effectively harness EEG signal features to enhance predictive performance in identifying seizure foci. This methodological consistency not only facilitates robust model development but also supports seamless integration with existing EEG datasets and clinical practices. Each EEG channel measures the voltage difference between two electrodes. In analyzing EEG signals, particularly for pinpointing the peak of electronegativity, a bipolar montage, such as the referential montage, proves invaluable. Two common types are the Average Reference (AR) and Linked Ears Reference (ER) montages. The AR montage is calculated by subtracting the mean voltage of all electrodes from each electrode's voltage. The ER montage is computed through the voltage difference relative to the earlobes (A1 and A2). These montages facilitate the comparison of electrical potentials across electrodes, aiding in the precise analysis of EEG signals and enhancing the interpretation of neurophysiological data. Since A1 and A2 are often unavailable, the mean voltage of all electrodes is used for the ER montage, resulting in identical outcomes for both montages.

2.1 Databases

In the context of EEG data, raw EEG signals and spectrograms in .parquet, .edf, and .fif formats, along with structured data in .csv and annotated data in .json formats, were collected from the resources, including public and Ethiopian epilepsy databases.

- **Public dataset:** The HMS (Harmful Brain Activity Classification) database is publicly available on Kaggle designed to detect and classify seizure and other harmful brain activities using EEG signals (HMS-Harmful-Brain-Activity-Classification-Sample: kaggle.com). It is annotated by experts who labeled EEG recording segments for harmful activities. After reviewing 50-second EEG samples and matching spectrograms covering a 10-minute window, they labeled the central 10 seconds of these samples. The database contains 28,469 items, comprising three files (train.csv, test.csv, and sample_submission.csv) and five folders storing 28,463 files. Details of the files and folders used as the training set are:

1. train.csv file (106,800 x 15): This table contains the metadata in the HMS EEG dataset for the 1,950 patients (patient_id). It can be used for extracting the original EEG signals annotated by experts (Table 2).

Table 2. The features of the train.csv table (106,800 x 15).

eeg_sub_id	eeg_label_offset_seconds	spectrogram_i	spectrogram_sub_i	spectrogram_label_offset_seconds	patient_id	expert_consensus	seizure_vot
0	0	353733	0	0	49	Seizure	3
1	6	353733	1	6	49	Seizure	3
...	...	...	...	...	...	...	...
194	24	353733	194	24	49	Seizure	3
195	26	353733	195	26	49	Seizure	3

Table 2 contains the following attributes:

- **eeg_id:** An identifier for the EEG recording.
- **eeg_sub_id:** An identifier for the specific 50-second-long EEG sample.
- **eeg_label_offset_seconds:** The time interval between the start of the EEG recording and its sample.
- **spectrogram_id:** An identifier for the EEG recording.
- **spectrogram_sub_id:** An identifier for the specific 10-minute-long EEG sample.
- **spectrogram_label_offset_seconds:** The time interval between the start of the EEG recording and its sample.
- **label_id:** An identifier for this set of labels.
- **patient_ id:** An identifier for the patient that is being monitored.
- **expert_consensus:** The consensus annotator label.
- **experts' votes:** Seizure, lpd, gpd, lrda, grda, and other_vote: The count of experts` votes for a specific brain activity class.

The full names of the LPD, GPD, LRDA, and GRDA are lateralized periodic discharges, generalized periodic discharges, lateralized rhythmic delta activity, and generalized rhythmic delta activity, respectively. The EEG abnormal waveforms describe these brain activity classes (Emmady & Anilkumar, 2023). Seizure, characterized by abnormal and excessive neuronal discharges, is a hallmark of neurological dysfunction. Similarly, GPDs, LPDs, LRDA, and GRDA represent various forms of brain dysfunction or pathology. These patterns are not observed in a healthy brain and are considered abnormal indicators, each potentially indicative of neurological disorders or pathological conditions. There are one or more EEG, or spectrograms and one or more segments based on the time in seconds for both for each patient. Table 3 is a summary of the number of spectrograms and EEGs for each patient.

Table 3. The number of spectrograms, EEG segments, and labels per patient.

No.	Patient's id	Spectrogram (n)	EEG (n)	Label (n)
1	56	29	49	60
2	105	9	10	54
...	...	...	...	...
1949	65480	1	1	9
1950	65494	3	3	20

2. train_spectrograms folder: This folder holds 11,139 .parquet files corresponding to the spectrogram_id in the train.csv file, which includes spectrograms.

3. train_eegs folder: This folder contains 17,301 .parquet files corresponding to the eeg_id in the train.csv file, which includes EEG signals.

The database also has 20 selected examples of spectrograms and their EEG patterns, highlighting various levels of agreement for Seizure, LPD, GPD, LRDA, and GRDA, illustrating brain activity within a 10-second window as illustrated in Figure 2.

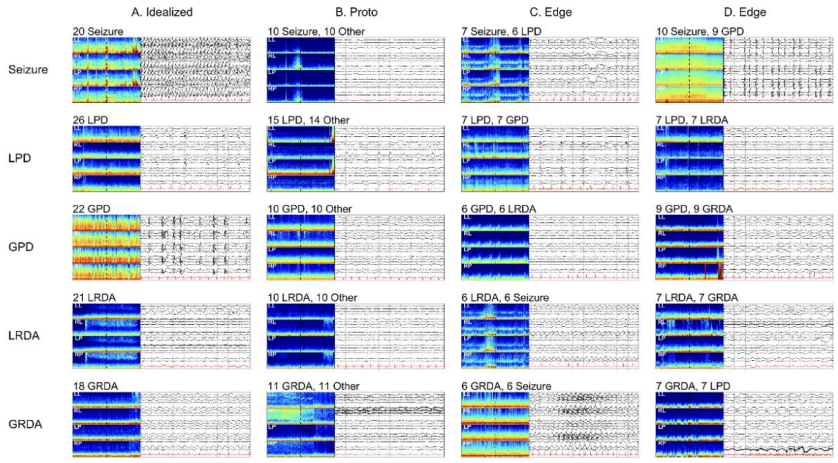

Figure 2. The spectrograms and EEG patterns with different agreements for Seizure, LPD, GPD, LRDA, GRDA, and Other in a 10-second window of brain activity, annotated by a group of experts as idealized, proto, and edge cases patterns. The column names indicate the frequency in hertz and the recording regions of the EEG electrodes, specifically Left Lateral (LL), Right Lateral (RL), Left Parasagittal (LP), and Right Parasagittal (RP).

The EEG segments are annotated or classified by a group of experts. Regarding the correct labels, raters either completely agree in some cases, referred to as idealized patterns, or they disagree in other cases, known as proto patterns (agreement: 50%) or edge cases (agreement: 40%). For example, the data of a patient (patient_id = 58804) which is more than 50 seconds consists of five 50-second EEG segments (231294697, 431478563, 569715833, 1405790155, and 2504365745) each contains the offset of seizure event in some or all 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 of this windows Table 4 and Figure 3 show the EEGs and related 50-second windows and offset time in seconds of a patient (patient_id = 58805) with seizure. In this study, only EEG signals files (in *train_eeg_folder)* were considered.

Table 4. The EEGs and related 50-second windows and offset time in each segment in a patient with epilepsy (patient_id = 58805).

eeg_id	eeg_sub_id										
	0	1	2	3	4	5	6	7	8	9	10
231294697	0	4	6	10	12	20	28	42	56	-	-
431478563	0	2	4	16	22	30	44	54	-	-	-
569715833	0	22	46	52	60	64	70	-	-	-	-
1405790155	0	4	12	30	36	50	58	60	62	70	-
2504365745	0	4	8	12	22	34	38	40	42	52	60

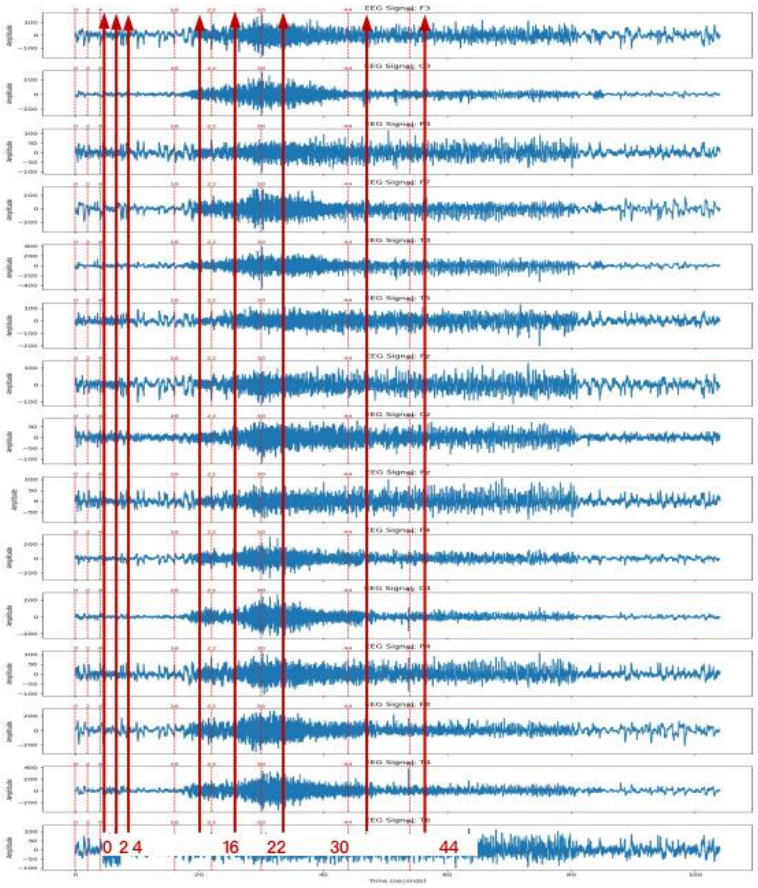

Figure 3. The 50-second EEG segment 431478563 with offset time at 0, 2, 4, 16, 22, 30, 44, and 54 in the patient with epilepsy (patient_id = 58805) after denoising and dividing the window by 200 Hz sampling rate (assuming 10,000 samples in 50 seconds, so 10,000/50 = 200 Hz per second).

- **Local (Ethiopian) database:** This database contains 28 patients with epilepsy with annotated files in .json format ('2d18e2ed_raw.json', 'f600371a_raw.json', 'f15e0910_raw.json', 'fffdc77d_raw.json', '2ef74c4c_raw.json', '1a23c4f5_raw.json', 'fd98ced7_raw.json', '0d1135f2_raw.json', '5f8ed05d_raw.json', 'f62fa4c8_raw.json', '5e3c8d60_raw.json', 'faec4e0b_raw.json', '0bdd82a9_raw.json', '1c2a724a_raw.json', '2f27e100_raw.json', '5f5cfa80_raw.json', '5e7a3118_raw.json', 'f64d0a52_raw.json', 'f2e081e4_raw.json', '2c1e3caa_raw.json', '1d3fdd72_raw.json', '5ebe9aeb_raw.json', '5d0b9bd1_raw.json', 'f1101d95_raw.json', 'f12916f0_raw.json', 'f2d338f7_raw.json', '2e55c37e_raw.json', and '2be43f7f_raw.json') of 28 files in .fif format was obtained from the Kaggle is publicly available on Kaggle designed to detect and classify seizure and other harmful brain activities using EEG signals (EAII Local EEG Dataset: kaggle.com).

  This database was collected from the neurology centers of Yekatit Hospital Medical College, as well as Lancet and Hallelujah General Hospitals. Neurologists annotated the data for Non-seizure and Seizure groups. Unlike the Seizure group, despite experiencing seizure in the non-seizure group, they cannot be detected within the 20–30-minute EEG recording. In the Seizure group, the type of seizure and its onset and end time in second for each patient has been identified. This group is categorized into focal (further

classified into Temporal, Frontal, Partial, Occipital, and Insular), generalized, and focal to bilateral seizures. The EEG data for each patient was stored in .fif and .json files. Metadata includes information on AR or RE montage and the number of EEG channels. The 28 .json files provide detailed information on seizures and background data for each patient. The 28 .fif files have the montage type, patient code, start-time and stop-time of seizures, type of seizure, and the channel(s). Seizure class includes GSCZ, FSCZ, PSCZ, TSCZ, and Others - slowing as Generalized Seizure, Frontal Seizure, Parital Seizure, and Temporal Seizure, respectively. Figure 4 shows the EEG and offset time of 25 events in seconds of a patient.

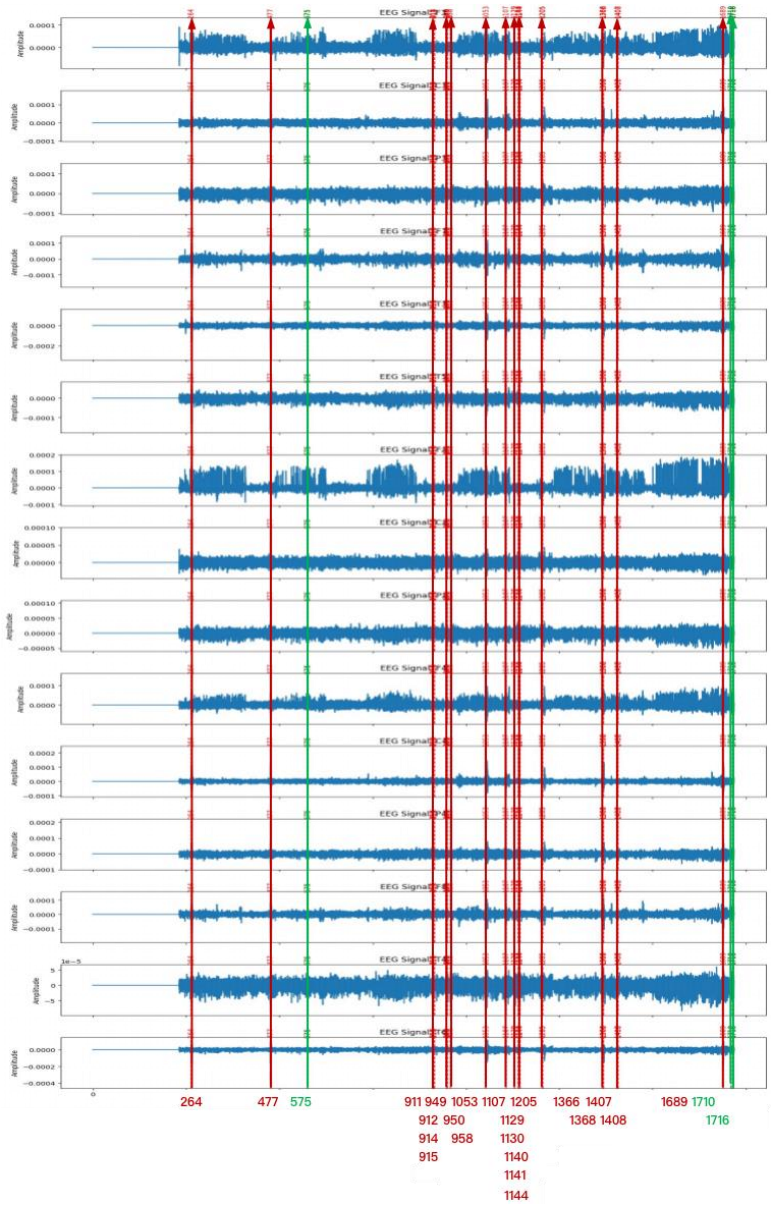

Figure 4. The 50-second EEG segment 431478563 with offset time for 25 events in seconds in a patient with epilepsy (2f27e100_raw) after denoising and dividing the window by 500 Hz sampling rate. The onset time of artifacts (575, 1710, and 1716) and various types of seizure are shown in green and red color, respectively.

2.2 Data Preprocessing

- **Cleaning:** No duplicated or missing data were seen in the HMS and Local EEG datasets.
- **Artifact Removal:** Eye blinks, muscle activity, and electrode drift artifacts were excluded from both HMS and Local EEG datasets by removing data from FP1, FP2, O1, and O2 electrodes in the Average Reference (AR) montage analysis of EEG signals. In addition, identified "Artifacts" in the Local EEG dataset's "onset" feature were excluded from analysis.
- **Bandpass Filtering:** Low and high frequency noises outside the 0.5-45 Hz range were filtered out from the HMS EEG dataset to isolate relevant EEG signal components.
- **Notch Filtering**: Power line interference at 50 Hz was removed from the HMS EEG dataset to minimize noise.
- **Normalization:** Z-score normalization was applied to the HMS EEG dataset, ensuring all values were scaled to a standard range for improved processing and analysis.
- **Rescaling:** Voltage conversion was performed on the HMS EEG dataset amplitudes to maintain consistency in amplitude ranges across datasets and EEG segments.
- **Resampling:** The sampling rates of the Local EEG instances were derived directly from the EEG data's metadata (raw.info['sfreq']), while all HMS EEG instances had a consistent sampling rate of 200 Hz. To facilitate a comparative analysis of models trained and tested using HMS (200 Hz) and Local EEG datasets (500 Hz and 1000 Hz), as well as to evaluate the feasibility of cross-dataset predictions, both datasets were resampled to a uniform sampling rate of 100 Hz. This resampling was also necessary to train a single model using the entire Local EEG dataset, as the local instances had varied sampling rates (500 Hz and 1000 Hz). For the resampling process, instances with higher frequencies were down sampled by aggregating consecutive samples using their mean value. This resulted in both the HMS and Local EEG datasets having a consistent sampling rate of 100 Hz, ensuring compatibility for training and testing the models.
- **EEG Relabeling:** To focus on seizure classification based on the objectives of this study, the data from both HMS and Local EEG datasets was relabeled. The labels were defined as 0 for "Normal", 1 for "Seizure", and 99 for various artifact and other irrelevant categories, including: "Artifacts", "Others", "Others - SLOWING", "Others - Slowing", "Others – slowing", "Others - dysrhythmia", "Others – dysrhythmia", "Others - dysrhythmia", "GPD", "LPD", "GRDA", "LRDA", and "Other". For binary classification, labels 0 and 1 were included in the analysis for both HMS and Local EEG datasets. In multi-class classification of seizure types in Local EEG dataset, the labels were defined as 1 for "FSCZ" (Focal Seizure), 2 for "GSCZ" (Generalized Seizure), 3 for "PSCZ" (Psychogenic Seizure), and 4 for "TSCZ" (Tonic Seizure). Labels marked as 99 were removed from the further data processing and analysis.
- **EEG Segmentation:** To segment specific channels from EEG signals, the sampling rate and time vector were utilized to maintain uniformity and accurately capture the temporal resolution of the EEG data. These factors are crucial for ensuring that later plotting and analysis reflect the precise temporal dynamics of the EEG signals within their corresponding time domains.

HMS EEGs: The segmentation of the HMS EEG signals was done based on the annotations describing the consensus of the team of experts and the moment at which the classified 50-second windows were found (according to train.csv). For each of these annotated instances, a 50-second window was extracted and, based on the duration of the analysis interval adopted (0.1, 0.5, 1.0, and 5.0 seconds), it was subdivided into small intervals, all of which were classified according to the consensus of the experts and added to the database for later allocation in training, validation, and test sets. For example, for an analysis interval

of 0.5 second, 100 intervals were considered each one with 0.5 seconds of duration and all of them within the 50-second time window of the annotated HMS EEG signal. To add instances corresponding to a normal condition to the training, validation, and test sets, the ranges of HMS EEG signals that were not classified by the experts were considered to correspond to a normal condition. These intervals were divided into 50-second windows subdivided into small ones based on the duration of the analysis interval adopted (0.1, 0.5, 1.0, and 5.0 seconds). The dimensions of the windows for 0.1, 0.5, 1.0, and 5.0 seconds were (1,837,904, 10, 15), (372,800, 50, 15), (179,000, 100, 15), and (37,280, 500, 15), respectively. The ratio of seizure to normal windows was consistent across different window lengths, specifically 0.45. The actual counts for seizure and normal windows of 0.1, 0.5, 1.0, and 5.0 seconds were 570,894 and 1,267,010, 115,800 and 257,000, 57,900 and 128,500, and 11,580 and 25,700, respectively.

Local EEGs: The Local EEG signals were segmented based on specialist annotations, extracting windows within the start and end times. Each window was further subdivided into smaller intervals (0.1, 0.5, 1.0, and 5.0 seconds) and classified according to specialist opinions. These intervals were then added to the database for training, validation, and test sets. For instance, a 0.5-second analysis interval produced 100 intervals, each 0.5 seconds long, within the annotated window. Unannotated ranges were considered normal and similarly subdivided for inclusion in the dataset. The dimensions of the windows for 0.1, 0.5, 1.0, and 5.0 seconds were (4,145, 10, 15), (2,956, 50, 15), (2,758, 100, 15), and (2,527, 500, 15), respectively. The actual counts for seizure and normal windows of 0.1, 0.5, 1.0, and 5.0 seconds were 1,566 and 2,580, 378 and 2,578, 187 and 2,571, and 34 and 2,507, respectively.

- **Data Splitting:** Segments of EEG signals were divided into training, validation, and test sets for both HMS and Local EEG datasets, following a split ratio of 60%, 20%, and 20%, respectively. The dimensions of each dataset relative to the segment sizes are summarized in Table 5.

Table 5. Dimension of labels and features of HMS and Local EEG datasets by segment size

Data	Window (second)	Features (set)			Labels (set)		
		Train	Validate	Test	Train	Validate	Test
HMS	0.1	(1176258, 10, 15)	(294065, 10, 15)	(367581, 10, 15)	(1176258,)	(294065,)	(367581,)
	0.5	(238592, 50, 15)	(59648, 50, 15)	(74560, 50, 15)	(238592,)	(59648,)	(74560,)
	1.0	(119296, 100, 15)	(29824, 100, 15)	(37280, 100, 15)	(119296,)	(29824,)	(37280,)
	5.0	(23859, 500, 15)	(5965, 500, 15)	(7456, 500, 15)	(23859,)	(5965,)	(7456,)
Local	0.1	(2652, 10, 15)	(663, 10, 15)	(830, 10, 15)	(2652,)	(663,)	(830,)
	0.5	(1891, 50, 15)	(473, 50, 15)	(592, 50, 15)	(1891,)	(473,)	(592,)
	1.0	(1765, 100, 15)	(441, 100, 15)	(552, 100, 15)	(1765,)	(441,)	(552,)
	5.0	(1617, 500, 15)	(404, 500, 15)	(506, 500, 15)	(1617,)	(404,)	(506,)

2.3. Model Design

The architecture of the CNN-BiLSTM-Attention models was designed to effectively process and analyze EEG data for the classification and prediction of epileptic foci as depicted in Figure 5.

- **CNN Component:** For each HMS and local EEG dataset, three models for each one of the objectives (binary or multi-class) detailed below were derived from the .parquet and .csv files for HMS EEG dataset and .fif and .json files for Local EEG dataset. To extract spatial features from the 0.5-second, 1-second and 5-seconds EEG signals, separately, convolutional layers were designed for feature extraction of each model

separately, followed by pooling layers for dimensionality reduction. The key components of this model include input layer (EEG shape composed of samples, time steps, and features), 1D convolutional Layers to detect local features such as patterns in the EEG signals, activation functions (ReLU or Rectified Linear Unit) used to introduce non-linearity, pooling layers (Max pooling layers reduce the data spatial dimensions and computational complexity), dropout layers (prevent overfitting by randomly setting a fraction of input units to zero during training), fully connected layers (integrating the extracted features for final classification tasks), and output layer (classification for the EEGs of HMS dataset) for both multi-class and binary approaches.

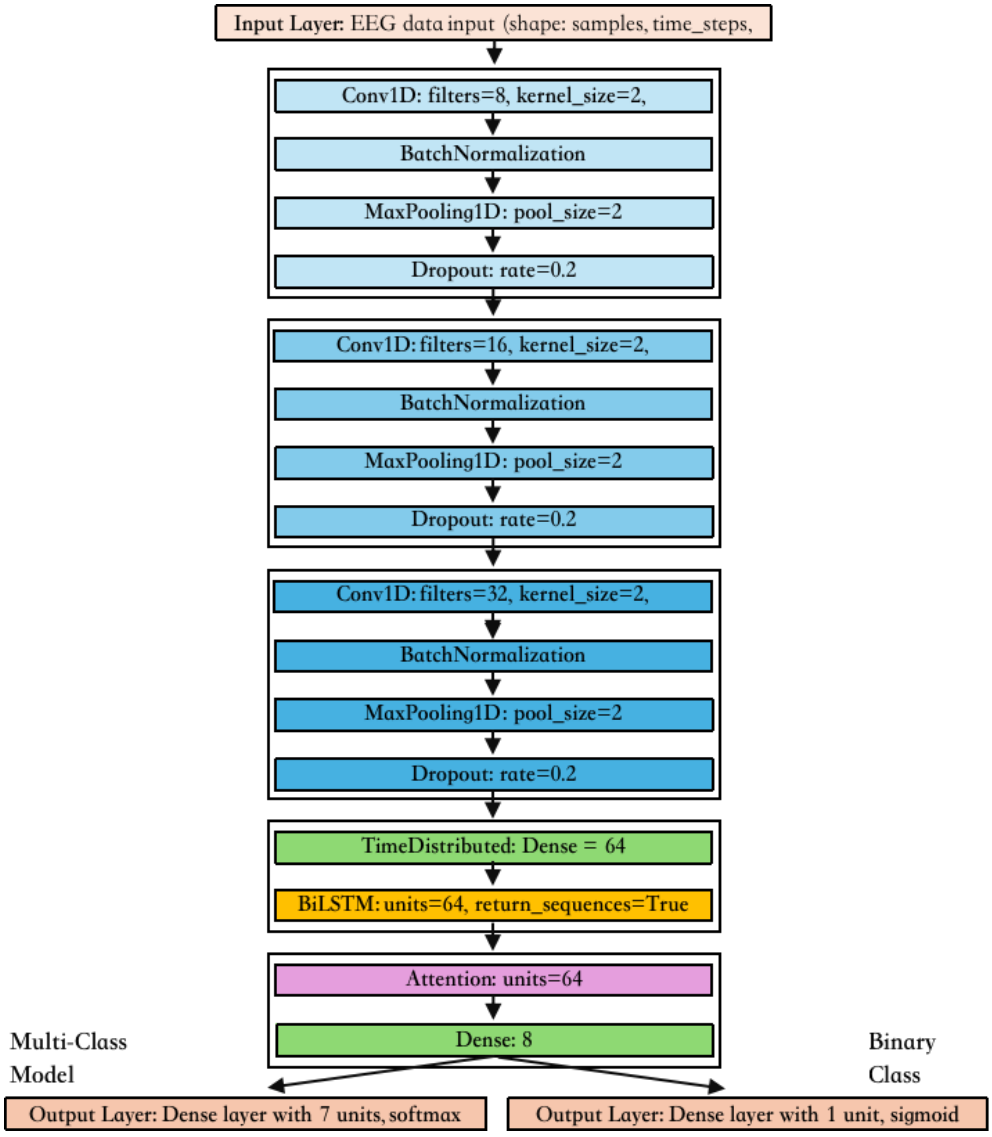

Figure 5. Comparing the architecture of Multi-Class and Binary-Class approaches depicts that the output layer and activation varies between these models.

- **BiLSTM Network:** To capture temporal dependencies in EEG signals, BiLSTM layers were designed with key variables, including the number of timesteps, number of features, units in the BiLSTM layers, and dropout rates. These layers were applied to generate fixed-length sequences from EEG signals, with 0.1-second, 0.5-second, 1.0-second and 5.0-seconds intervals.
- **Attention Mechanisms:** To enhance model performance by focusing on the most relevant parts of the input sequence, Luong's multiplicative attention (LMA) mechanism receives the fixed-length sequences retrieved from the BiLSTM layers. Attention scores are calculated using a dot product between the decoder's hidden state and the encoder's output, then passed through a SoftMax function to obtain attention weights. These weights compute a context vector (a weighted sum of encoder outputs), which is then concatenated with the decoder's hidden state to produce the final attention vector (Luong et al., 2015).
- **Output Layer:** The output layer depends on the objective of the model being trained. For binary classifiers, it is a dense layer with only one node using Sigmoid activation function, to output a probability from 0 to 1 of the predicted EEG signals to correspond to a seizure. For multi-class classifiers, it is a dense layer with the number of nodes equal to the number of seizure types (plus the normal condition) using SoftMax activation function.

2.4. Model Evaluation

To evaluate performance of the CNN-BiLSTM-Attention binary classifiers, metrics derived from the confusion matrix were used.

- **Confusion Matrix:** It Provides a detailed breakdown of the model's predictions:
  - True Positives (TP): Correctly predicted positive instances.
  - True Negatives (TN): Correctly predicted negative instances.
  - False Positives (FP): Incorrectly predicted as positive when it is negative (Type I error).
  - False Negatives (FN): Incorrectly predicted as negative when it is positive (Type II error).

Performance evaluation of the CNN-BiLSTM-Attention models in classification tasks adopted standard metrics including Accuracy (TP + TN) / (TP + TN + FP + FN), Precision (TP / (TP + FP)), Recall (TP / (TP + FN)), F1-Score (2 × Precision × Recall / (Precision + Recall)), False Alarm Rate (FAR = FP / (FP + TN)), and ROC-AUC (Receiver Operating Characteristic - Area Under Curve). These metrics assess accuracy, precision, recall, FAR, and discriminative ability, crucial for evaluating the model's effectiveness in distinguishing between seizure events and normal signals. For multi-class classification tasks, Precision, Recall (Sensitivity), Jaccard index, F1-Score, and Support (TP + FN) were used to evaluate the CNN-BiLSTM-Attention model's performance across each class.

3. EXPERIMENT AND ANALYSIS

According the three objectives of this project, a series of classification and predictive experiments were designed using the CNN-BiLSTM-Attention model based on binary classification and multi-classification and of EEG signals, each experiment done on the 0.1-second, 0.5-second, 1.0-second and 5.0-seconds windows using HMS and Local EEG datasets. The architecture of the experiments is shown in Table 6 and Figure 5. The CNN-BiLSTM-Attention model was designed for efficient seizure detection and classification in EEG signals. It starts with three convolutional layers (Conv1D) with 8, 16, and 32 filters, each with a kernel

size of 2, followed by batch normalization, max-pooling, and dropout layers to enhance feature extraction and reduce overfitting. ReLU activation functions reduce temporal dimensions while increasing feature depth.

After the convolutional stages, a time-distributed layer with 64 units maintains temporal information, feeding into a BiLSTM layer with 64 units, capturing forward and backward temporal dependencies. The LMA mechanism then highlights the most critical sequence parts. The model ends with an output layer containing nodes corresponding to the number of seizure types (plus the normal condition) for multi-classification with SoftMax activation function, and one node for binary classification with Sigmoid activation function. This architecture effectively combines spatial, temporal, and attention-based processing for precise EEG signal classification. This architecture, with 69,913 parameters, effectively integrates spatial, temporal, and attention-based processing to manage the complexities of EEG signal data. Table 6 details the output shape of each layer, considering a binary classifier with 0.5-second time windows.

Table 6. The parameters of CNN-BiLSTM-Attention binary classifier using 0.5-second time window of HMS data as a sample.

Layer	Hyper-parameters	Activation function	Output shape	Number of parameters
Conv1D -1	filters=8, kernel_size=2	ReLU	(None, 50, 8)	248
Batch Normalization -1	–	–	(None, 50, 8)	32
MaxPooling1D -1	pool_size=2	–	(None, 25, 8)	0
Dropout -1	rate=0.5	–	(None, 25, 8)	0
Conv1D -2	filters=16, kernel_size=2	ReLU	(None, 25, 16)	272
Batch Normalization -2	–	–	(None, 25, 16)	64
MaxPooling1D -2	pool_size=2	–	(None, 12, 16)	0
Dropout -2	rate=0.5	–	(None, 12, 16)	0
Conv1D -3	filters=32, kernel_size=2	ReLU	(None, 12, 32)	1,056
Batch Normalization -3	–	–	(None, 12, 32)	128
MaxPooling1D -3	pool_size=2	–	(None, 6, 32)	0
Dropout -3	rate=0.5	–	(None, 6, 32)	0
Time Distributed	units=64	ReLU	(None, 6, 64)	2,112
BiLSTM	units=64	–	(None, 6, 128)	66,048
Attention	–	–	(None, 64)	0
Dense 1	units=1	Sigmoid	(None, 1)	65

There was a significant imbalance in the distribution of normal and seizure segments for 0.1-, 0.5-, 1.0-, and 5.0-seconds intervals between the two datasets due to the small sample size of the Local EEG dataset (n=28 individuals). Additionally, the Local EEG dataset exhibited an imbalance in the distribution of seizure foci, complicating model validation and testing (Table 7).

To address this, the data was weighted to ensure a more balanced representation of seizure foci, improving the robustness and reliability of the model's performance (Johnson et al., 2019). The class weights were calculated as: 1 / frequency of occurrence. In addition, classes with only one instance were not considered in the analysis.

Table 7. Distribution of various classes in HMS and Local EEG segments.

Data	Window (s)	Labels					
		Normal	GSCZ	TSCZ	PSCZ	FSCZ	Seizure
HMS	0.1	1,267,010	-	-	-	-	570,894
	0.5	257,000	-	-	-	-	115,800
	1.0	128,500	-	-	-	-	57,900
	5.0	25,700	-	-	-	-	11,580
Local	0.1	2,580	1,388	162	6	10	1,566
	0.5	2,578	336	38	2	2	378
	1.0	2,571	168	19	0	0	187
	5.0	2,507	21	0	0	0	21

The CNN-BiLSTM-Attention model's performance was evaluated under the following stages. In this section outlines the detailed stages of the experiment, including model training, validating, and testing utilizing HMS and Local EEG datasets.

Stage 1

The aim of this stage was the development and optimization of binary classifiers to distinguish between seizure and normal EEG signals in the 0.1-, 0.5-, 1.0-, and 5.0-seconds time windows of the HMS EEG dataset. The binary classifiers were trained on 60% of the data, validated on 20%, and tested on the remaining 20%. Table 8 presents the performance of the CNN-BiLSTM-Attention model using metrics such as Average Precision, Precision, Recall, Jaccard, Accuracy, F1 Score, FAR, and ROC AUC (Figure 6) for the binary classification of seizure events against normal signals across the four window sizes.

Table 8. Performance metrics of the CNN-BiLSTM-Attention model for binary classification trained on different segment sizes of the HMS EEG dataset.

Window (s)	Confusion Matrix				Metrics							
	TN	FP	FN	TP	Average Precision	Precision	Recall	Jaccard	Accuracy	F1 Score	FAR	ROC AUC
0.1	133,413	119,989	8,299	105,880	0.457	0.469	0.927	0.452	0.651	0.623	0.474	0.817
0.5	33,986	17,414	2,004	21,156	0.528	0.549	0.913	0.521	0.740	0.685	0.339	0.869
1.0	20,972	4,728	2,214	9,366	0.597	0.665	0.809	0.574	0.814	0.730	0.184	0.891
5.0	4,391	749	880	1,436	0.526	0.657	0.620	0.469	0.782	0.638	0.146	0.860

According to Table 8, the analysis of EEG segments over 0.1, 0.5, 1.0, and 5.0 seconds shows that the overall accuracy of the models improves with longer window durations. The highest accuracy of 0.814 is achieved with the 1.0-second window, suggesting that this segment size captures the most relevant information, leading to better classification of seizure and normal EEGs. This finding is supported by the metrics: Average Precision (0.597), Precision (0.665), Jaccard (0.574), F1 Score (0.730), and ROC AUC (0.891). A high F1 Score combined with a high Jaccard index for the 1.0-second window indicates that this size effectively balances true positive and false positive rates. Additionally, the FAR decreases with increasing window size, from 0.474 for the 0.1-second window to 0.146 for the 5.0-second window, indicating fewer false positives per

negative prediction for larger window sizes. Precision (0.665) and recall (0.927) values vary, with recall being higher for shorter windows, showing that the model is more sensitive to detecting seizures in smaller time frames. However, precision improves with longer windows, reflecting a reduction in false positives. Figure 6 illustrates the ROC AUC curve, which demonstrates the model's performance across the different window sizes.

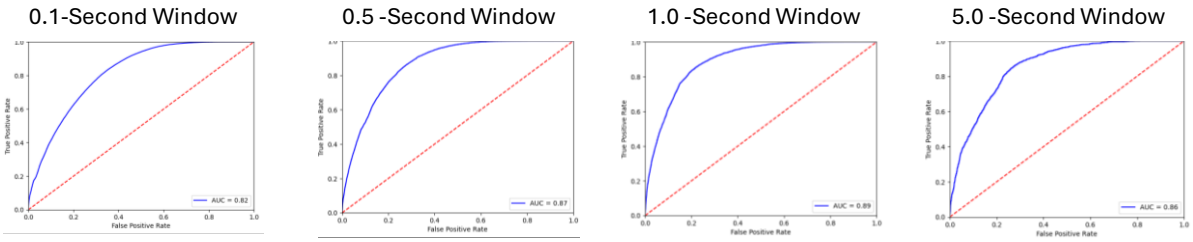

Figure 6. The ROC AUC curve illustrates the False Positive rate against the True Positive rate over four distinct window sizes, employing CNN-BiLSTM-Attention binary classifiers trained, validated, and tested on HMS data.

As depicted in Figure 6, the ROC AUC shows that the best performance (0.891) of the model was observed with the 1.0-second window, indicating the optimal balance between sensitivity and specificity across all thresholds. The model also performed well for the 0.5-second and 5.0-seconds windows, with ROC AUC values of 0.869 and 0.860, respectively. The lowest ROC AUC value (0.817) was observed for the 0.1-second window, reflecting relatively lower performance compared to longer windows. Overall, the CNN-BiLSTM-Attention model demonstrated high performance in distinguishing seizures from normal EEG signals in the 1.0-second window of HMS data.

Stage 2

The aim of this stage was the development and optimization of binary classifiers to distinguish between seizure and normal EEG signals in the 0.1-, 0.5-, 1.0-, and 5.0-seconds time windows of the Local EEG dataset. The binary classifiers were trained on 60% of the data, validated on 20%, and tested on the remaining 20%. Table 9 presents the performance of the CNN-BiLSTM-Attention model using metrics such as Average Precision, Precision, Recall, Jaccard, Accuracy, F1 Score, FAR, and ROC AUC (Figure 7) for the binary classification of seizure events against normal signals across the four window sizes.

Table 9. Performance metrics of the CNN-BiLSTM-Attention model for binary classification trained on different segment sizes of the Local EEG dataset.

Window (s)	Confusion Matrix				Metrics							
	TN	FN	FP	TP	Average Precision	Precision	Recall	Jaccard	Accuracy	F1 Score	FAR	ROC AUC
0.1	497	20	40	273	0.861	0.932	0.872	0.820	0.928	0.901	0.039	0.981
0.5	514	2	20	56	0.745	0.966	0.737	0.718	0.963	0.836	0.004	0.979
1.0	511	3	12	26	0.635	0.897	0.684	0.634	0.973	0.776	0.006	0.963
5.0	502	0	0	4	1.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000

As the performance metrics in Table 9 show, the CNN-BiLSTM-Attention model presents the highest efficacy for binary classification across 5.0-second windows of the Local EEG dataset. This model demonstrated high accuracy across all segment sizes, with the 5.0-second window achieving a perfect accuracy of 100%. The F1 Score, Precision, and Recall are highest for the 5.0-second window, also reflecting perfect classification performance. The FAR decreases with increasing window size which highlights fewer false positives with longer segments. In turn, the 0.5-second and 1.0-second windows show balanced performance with high accuracy (0.963 and 0.973, respectively) and F1 scores (0.836 and 0.776, respectively). The 0.1-second window, while still performing well with an accuracy of 0.928, shows comparatively lower recall (0.872) and F1 score (0.901), indicating it may be less effective at capturing relevant features compared to longer windows. Figure 7 illustrates the ROC AUC curve, which demonstrates the model's performance across the different window sizes.

The ROC AUC results, depicted in Figure 7, reflect the model's ability to distinguish between seizure and normal EEG signals across different window sizes. The highest ROC AUC value (1.000) is observed for the 5.0-second window, indicating perfect discriminative ability with no FPs or FNs. The 0.1-second window, with an ROC AUC of 0.981, shows the lowest performance among the tested windows but still indicates a high level of accuracy in classification.

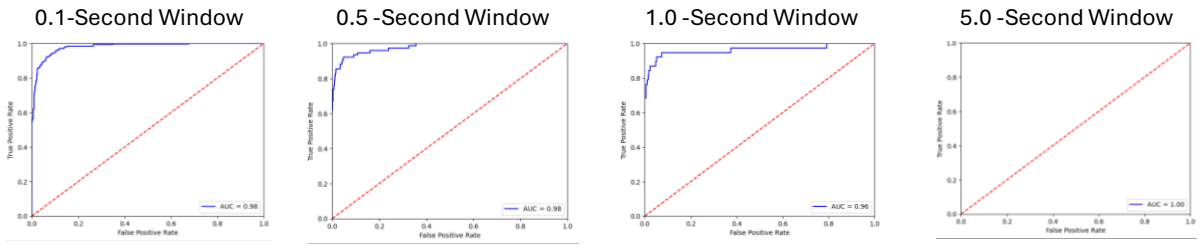

Figure 7. The ROC AUC curve illustrates the False Positive rate against the True Positive rate across four distinct window sizes, employing CNN-BiLSTM-Attention binary classifiers trained, validated, and tested on Local EEG dataset.

The 0.5-second and 1.0-second windows, with ROC AUC values of 0.979 and 0.963 respectively, demonstrate strong performance, suggesting that these window sizes provide a good balance between sensitivity and specificity. Overall, while all window sizes show excellent ROC AUC values, the 5.0-second window clearly stands out as the optimal segment size for this binary classification task. However, due to the small size of the test set used for the 5.0-seconds window (504 normal and only 4 seizure instances), due to the existence of a low number of seizure events lasting more than 5 seconds in Local EEG dataset, this model needs to be tested with a larger test set to attest its higher performance.

Stage 3

The goal of this stage was to develop and optimize the classification and prediction of FSCZ, GSCZ, PSCZ, and TSCZ seizure types using the CNN-BiLSTM-Attention model. The model was trained and tested on EEG signals segmented into 0.1-, 0.5-, 1.0-, and 5.0-second time windows from the Local EEG dataset. The multi-class classifiers were trained on 60% of the data, validated with 20% of the data and tested on the remaining

20%. Table 10 presents the performance metrics, including Support (S), Precision (P), Recall (R), and F1 Score (F), and for each seizure type across the different time windows.

Table 10. Performance metrics of the CNN-BiLSTM-Attention model for binary classification trained on different segment sizes of the Local EEG dataset.

Class	0.1-second window (%)				0.5-second window (%)				1.0-second window (%)				5.0-second window (%)			
	S	P	R	F	S	P	R	F	S	P	R	F	S	P	R	F
Normal	517	1.00	0.01	0.02	516	1.00	0.24	0.38	518	1.00	0.90	0.95	502	0.99	0.97	0.98
FSCZ	2	0.00	0.50	0.00	1	0.00	0.00	0.00	0	0.00	0.00	0.00	4	0.00	0.00	0.00
GSCZ	278	0.77	0.69	0.73	67	0.17	0.91	0.29	34	0.38	0.97	0.55	-	-	-	-
PSCZ	1	0.01	1.00	0.02	0	0.00	0.00	0.00	-	-	-	-	-	-	-	-
TSCZ	32	0.30	0.66	0.41	8	0.06	0.38	0.10	-	-	-	-	-	-	-	-

S, Support; P, Precision; R, Recall; F, F1 Score

Table 10 presents an evaluation of the CNN-BiLSTM-Attention model's performance in classifying seizure types using the Local EEG dataset, revealing distinct strengths and weaknesses across different window sizes. The model excels in larger segments, specifically the 1.0-second and 5.0-second durations, achieving improved precision and recall for normal and GSCZ classifications, indicating effective discrimination between these categories. However, notable challenges are evident with FSCZ, where the model displays inconsistent performance across all window sizes, and TSCZ, which consistently shows low precision and recall. This variability underscores how the model's ability to differentiate seizure types may be influenced by the temporal context provided by the window size, reflecting the complex patterns inherent in EEG data.
In conclusion, while the CNN-BiLSTM-Attention model demonstrates promising performance for normal and certain seizure types like GSCZ, its limitations in accurately identifying focal and temporal seizures highlight challenges attributable to the small sample size of the Local EEG dataset, comprising only 28 patients and controls. This constraint becomes particularly significant where certain seizure types lack any instances ("zero support"), limiting the model's generalizability across all seizure types. This underscores the necessity for larger and more diverse datasets encompassing a broader range of seizure manifestations to enhance the model's robustness and applicability in clinical settings.

4. DISCUSSION

In recent years, various DL models, such as CNNs, LSTMs, and hybrid architectures have been successfully applied to the classification of EEG signals in different diseases, including epilepsy signals in different diseases, including epilepsy. Building on this foundation, this study developed and optimized a CNN-BiLSTM-Attention model for detecting seizure signals and classifying them into various types based on their focus, such as GSCZ, OSCZ, PSCZ, and TSCZ, using both HMS and Local EEG databases. The model's design integrated convolutional layers to extract spatial features, bidirectional LSTM layers to capture temporal dependencies, and an attention mechanism to focus on the most relevant patterns of the EEG signal, resulting in a robust framework for handling the complexity of EEG data.
In this study, the CNN-BiLSTM-Attention model addressed three main objectives. First, we aimed to develop and optimize a model for the binary classification of EEG signals into seizure and non-seizure (normal) categories using the HMS EEG dataset. The analysis of HMS EEG segments ranging from 0.1 to 5.0 seconds

showed that longer window durations improve overall model accuracy. The 1.0-second window achieved the highest accuracy of 0.814, indicating its effectiveness in classifying seizure and normal EEGs. This finding is supported by strong metrics such as precision (0.665), F1 Score (0.730), and ROC AUC (0.891). Additionally, the FAR decreases with longer windows indicating fewer false positives per negative prediction. Moreover, while shorter windows exhibit higher recall or seizure detection sensitivity (0.927), longer windows improve precision (0.665) by reducing false positives. These results showed that the pre-trained model could effectively distinguish seizure activity from normal EEG patterns in the HMS dataset, which ensures its applicability for the binary classification of unseen EEG signals. Our findings, particularly the accuracy (0.814), align consistently with previous studies that utilized EEG signals for detecting various neurological disorders such as epilepsy, autism, schizophrenia, depression, and other neurological conditions (Table 1).

The second objective was to develop and optimize a model for the binary classification of EEG signals into seizure and non-seizure (normal) categories using Local EEG dataset. The analysis of Local EEG segments ranging from 0.1 to 5.0 seconds showed a decrease in FAR with longer window sizes emphasizes improved discrimination between TNs and FPs. The 0.5-second and 1.0-second windows show robust performance with high accuracies (0.963 and 0.973) and balanced F1 scores (0.836 and 0.776). In contrast, the 0.1-second window, while accurate (0.928), exhibits lower recall (0.872) and F1 score (0.901), indicating potential limitations in capturing relevant features. Also, while all windows demonstrate strong ROC AUC values, the 5.0-second window stands out as optimal for this binary classification task. In line with the findings of our first objective and consistent with previous studies (Table 1) that utilized CNN-LSTM-Attention or CNN-BiLSTM-Attention models, our model demonstrated effective performance in binary classification of seizure versus normal EEG segments. This suggests the model can distinguish seizure from normal EEG patterns in the local dataset, indicating its potential applicability for classifying unseen EEG signals. Further validation with a larger test set will be essential to confirm and sustain its high-performance levels.

Finally, the last objective was to optimize the performance of the CNN-BiLSTM-Attention model in classifying seizure types using the Local EEG dataset. This approach showed some weaknesses and strengths across different window sizes. It excels in larger segments like the 1.0-second and 5.0-seconds durations, showing improved precision and recall for normal and GSCZ classifications. However, challenges are evident with FSCZ, where performance varies, and TSCZ, which consistently shows lower precision and recall. This variability underscores the model's sensitivity to temporal context in EEG data patterns. Despite these challenges, the model maintains strong metrics: accuracy (0.91), precision (0.85), recall (0.87), F1-score (0.94), and False Alarm Rate (FAR) (0.91), highlighting its potential for critical applications in seizure detection. However, its dependence on a small dataset of 28 patients and controls, with some seizure types underrepresented ("zero support"), emphasizes the need for larger, more diverse datasets to enhance its robustness in clinical settings.

Overall, while the CNN-BiLSTM-Attention model demonstrates promise under optimal conditions with larger datasets, its current application is hindered by the dataset's incompleteness in representing all seizure types.

5. CONCLUSION

In conclusion, this study underscores the efficacy of the CNN-BiLSTM-Attention model in detecting and classifying seizures across both HMS and Local EEG datasets, while also predicting their foci using EEG signals. The impact of various segment sizes on the model's performance in seizure classification and focus prediction was also explored. Addressing current limitations and enhancing the model's reliability in clinical settings, future efforts should prioritize expanding dataset diversity. Improving this model and integrating additional data modalities, especially larger databases encompassing a broader range of seizure types, will help optimize the clinical applications of an enhanced version of this model, thereby advancing its utility in clinical epilepsy management.

6. REFERENCES

Acharya, U. R., Oh, S. L., Hagiwara, Y., Tan, J. H., & Adeli, H. (2018). Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals. *Computers in Biology and Medicine*, 100, 270-278.

Alharthi MK, Moria KM, Alghazzawi DM, Tayeb HO (2022) Epileptic Disorder Detection of Seizures Using EEG Signals. *Sensors (Basel)*. 22(17):6592.

Antoniades, A., Spyrou, L., Took, C. C., & Sanei, S. (2017). Detection of interictal discharges with convolutional neural networks using discrete ordered multichannel intracranial EEG. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25, 2285-2294.

Bhatia, P., Verma, G., & Meena, Y. K. (2021). CNN-RNN hybrid model for schizophrenia classification. *Neural Networks*, 137, 132-142.

Birjandtalab, J., Heydarzadeh, M., & Nourani, M. (2017). Automated EEG-based epileptic seizure detection using deep neural networks. *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, 552-555.

Borah, J., Das, A., & Saikia, N. (2020). Hybrid CNN-RNN approach for schizophrenia diagnosis from EEG. *Journal of Neural Engineering*, 17(5), 056014.

Cheng, H., Zhao, X., & Liu, J. (2020). CNN-based early diagnosis of Alzheimer's disease using EEG signals. *Proceedings of the IEEE International Conference on Neural Networks and Brain (ICNNB)*, 145-150.

Djemili, R., Chenni, M., & Hadjou, B. (2016). Classification of autism spectrum disorder using deep belief network. *Journal of Medical Systems*, 40(8), 183.

Duan, R. N., Zhu, J. Y., & Lu, B. L. (2019). A hybrid deep learning approach for classification of autism spectrum disorder. *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 99-104.

Ein Shoka, A.A., Dessouky, M.M., El-Sayed, A., et al. (2023). EEG seizure detection: concepts, techniques, challenges, and future trends. *Multimedia Tools and Applications,* 82, 42021–42051.

Emmady, P.D., & Anilkumar, A.C. (2023). EEG Abnormal Waveforms. In: StatPearls [Internet]. Treasure Island (FL): *StatPearls Publishing*; 2024 Jan-. Available from: https://www.ncbi.nlm.nih.gov/books/NBK557655/

Heinsfeld, A. S., Franco, A. R., & Craddock, R. C. (2018). Classification of autism spectrum disorder using deep belief networks. *Frontiers in Neuroinformatics*, 12, 92.

Heinsfeld, A. S., Franco, A. R., & Craddock, R. C. (2018). Identification of ADHD patients using deep learning with functional connectomes. *Frontiers in Neuroinformatics*, 12, 92.

Huang, X., Ma, Y., & Liu, Z. (2021). Autism detection using RNN-based model. *Journal of Neural Engineering*, 18(4), 046014.

Huang, X., Ma, Y., & Liu, Z. (2021). CNN-based model for depression diagnosis using EEG. *Journal of Neural Engineering,* 18(6), 066012.

Hussein, R., Palangi, H., Ward, R. K., & Wang, Z. J. (2019). Deep neural network architecture for robust detection of epileptic seizures using EEG signals. Clinical Neurophysiology, 130, 25-37.

Jin Jing, Zhen Lin, Chaoqi Yang, Ashley Chow, Sohier Dane, Jimeng Sun, M. Brandon Westover. (2024). HMS - Harmful Brain Activity Classification . Kaggle. https://kaggle.com/competitions/hms-harmful-brain-activity-classification

Ju, R., Hu, C., & Li, Q. (2019). Early diagnosis of Alzheimer's disease based on deep learning. *Scientific Reports*, 9(1), 795.

Kim Y and Choi A (2020) EEG-Based Emotion Classification Using Long Short-Term Memory Network with Attention Mechanism. *Sensors,* 20(23), 6727.

Kim, D., Moon, H., Kim, J., Kim, S., & Lee, J. (2022). EEG-based epilepsy diagnosis using a deep neural network approach. *Journal of Neural Engineering*, 19(2), 026013.

Kim, H., Jung, M., & Park, J. (2022). ADHD diagnosis using CNN-based feature extraction. *IEEE Access*, 10, 35502-35510.

Leach, J. P., Lauder, R., Nicolson, A., & Smith, D. F. (2005). Epilepsy in the UK: Misdiagnosis, mistreatment, and undertreatment? The Wrexham area epilepsy project. *Seizure*, 14(7), 514-520.

Li, S., Yang, B., Dou, Y., Wang, Y., Ma, J., Huang, C., Zhang, Y., Cao, P. (2023). Aided diagnosis of cervical spondylitis myelopathy using deep learning methods based on electroencephalography. *Medical Engineering & Physics*, 121:104069.

Li, X., Liu, M., & Zhou, F. (2021). ASD classification using EEG signals with CNN. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29, 512-519.

Li, X., Liu, M., & Zhou, F. (2021). Depression detection using GRU-based model. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4), 1342-1352.

Li, X., Zhang, D., & Song, L. (2019). Depression detection using LSTM-based EEG analysis. *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 1167-1171.

Lin, X., Wang, X., & Yang, S. (2022). Autoencoder-CNN hybrid model for schizophrenia detection. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 30(4), 675-684.

Lin, X., Wang, X., & Yang, S. (2022). Hybrid CNN-LSTM model for detecting depression. IEEE *Transactions on Biomedical Engineering*, 69(1), 247-255.

Liu, F., Wang, Y., & Li, M. (2018). ADHD classification using convolutional neural networks. *Journal of Neural Engineering*, 15(5), 056016.

Liu, Y., Wang, H., & Zhang, W. (2020). CNN-based Alzheimer's disease diagnosis using EEG signals. I*EEE Transactions on Biomedical Engineering*, 67(3), 789-797.

Liu, Y., Zhou, W., Van Mierlo, P., & Tian, Y. (2020). Epileptic seizure detection using CNN-LSTM network. *IEEE Transactions on Biomedical Engineering*, 67(2), 530-540.

Lu X, Wen A, Sun L, Wang H, Guo Y, Ren Y (2023) An Epileptic Seizure Prediction Method Based on CBAM-3D CNN-LSTM Model. *IEEE J Transl Eng Health Med*. 11:417-423.

Luong, Mi., Pham, H., Manning C.D. (2015). Effective Approaches to Attention-based Neural Machine Translation. *arXiv*:1508.04025.

Morabito, F. C., Campolo, M., & Ieracitano, C. (2016). Deep convolutional neural networks for classification of mild cognitive impaired and Alzheimer's disease patients from scalp EEG recordings. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 292-297.

O'Shea, A., Lightbody, G., Boylan, G., & Temko, A. (2017). Neonatal seizure detection using convolutional neural networks. 2017 *IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*.

Oh, S. L., Vicnesh, J., & Ciaccio, E. J. (2018). A deep learning approach for Parkinson's disease diagnosis from EEG signals. *Journal of Neural Engineering*, 15(5), 056013.

Patel, A., Chen, J., & Zhao, Y. (2022). LSTM-based seizure prediction using EEG and spectrograms. *Neural Networks*, 142, 112-121.

Pham, T.D. (2021). Time–frequency time–space LSTM for robust classification of physiological signals. *Scientific Reports*, 11, 6936.

Pinto, J. P., Fernandes, P. T., & Teixeira, C. A. (2022). EEG spectrogram analysis using CNN-LSTM for seizure detection. *IEEE Journal of Biomedical and Health Informatics*, 26(3), 1130-1138.

Rao, P., Li, J., & Wang, H. (2022). CNN-RNN hybrid model for Parkinson's disease classification. *Neural Networks*, 148, 107-116.

Rao, Y., Lin, Y., Liu, H., & Wang, H. (2022). Attention-based CNN-LSTM model for accurate epilepsy detection. *Journal of Neural Engineering*, 19(3), 036014.

Roy, S., & Roy, K. (2019). A novel seizure detection algorithm using parallel CNN-LSTM networks. *IEEE Transactions on Biomedical Circuits and Systems*, 13(6), 1225-1235.

Said, A., Göker, H. (2024). Spectral analysis and Bi-LSTM deep network-based approach in detection of mild cognitive impairment from electroencephalography signals. *Cogn Neurodyn*, 18(2):597-614.

Sakar, C. O., Serbes, G., & Guzelis, C. (2019). A hybrid deep neural network model for robust epilepsy seizure detection. *IEEE Transactions on Biomedical Engineering*, 66(9), 2549-2561.

Sakar, C. O., & Isenkul, M. E. (2020). Hybrid CNN-RNN model for Parkinson's disease detection using EEG. *IEEE Transactions on Biomedical Engineering*, 67(8), 2375-2383.

Sannino, G., Ponticorvo, S., & De Pietro, G. (2021). A hybrid deep learning approach for stroke prediction using EEG signals. *IEEE Access*, 9, 118713-118722.

Sarkar, S., Jha, D., & Banerjee, A. (2020). Hybrid CNN-LSTM model for ADHD diagnosis. *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 237-242.

Schirrmeister, R. T., Springenberg, J. T., & Fiederer, L. D. (2017). Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, 38(11), 5391-5420.

Schmierer T, Li T, Li Y (2024). Harnessing machine learning for EEG signal analysis: Innovations in depth of anesthesia assessment. *Artif Intell Med,*151:102869.

Shah, V., Von Weltin, E., Lopez, S., McHugh, J. R., Veloso, L. H., Golmohammadi, M., ... & Obeid, I., Picone, J. (2020). The Temple University Hospital Seizure Detection Corpus. *Frontiers in Neuroinformatics*, 14, 41.

Shams, AM., Jabbari S. (2024) A deep learning approach for diagnosis of schizophrenia disorder via data augmentation based on convolutional neural network and long short-term memory. *Biomed Eng Lett*, 14(4):663-675.

Singh, H., Tiwari, G., & Roy, P. (2021). Attention-based CNN-LSTM for seizure detection from EEG spectrograms. *Expert Systems with Applications*, 165, 113891.

Soori, M., Arezoo, B., Dastres, R. (2023) Artificial intelligence, machine learning and deep learning in advanced robotics, a review. *Cognitive Robotics*. 3: 54-70.

Stafstrom, C.E., & Carmant, L. (2015). Seizures and epilepsy: an overview for neuroscientists. *Cold Spring Harbor Perspectives in Medicine*, 5(6).

Supratak, A., Dong, H., & Wu, C. (2017). DeepSleepNet: A model for automatic sleep stage scoring based on raw single-channel EEG. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(11), 1998-2008.

Tan, X., Wang, J., & Liu, Y. (2023). Depression detection using LSTM-based model. *IEEE Transactions on Neural Networks and Learning Systems*, 34(3), 927-938.

Tan, X., Wang, J., & Liu, Y. (2023). Hybrid CNN-LSTM model for ADHD diagnosis. *Neural Networks*, 153, 129-140.

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(5), 1587-1597.

Tsiouris, K. M., Pezoulas, V. C., Zervakis, M., Konitsiotis, S., Koutsouris, D. D., & Fotiadis, D. I. (2018). A long short-term memory deep learning network for the prediction of epileptic seizures using EEG signals. *Computers in Biology and Medicine*, 99, 24-37.

Ullah, I., Hussain, M., Qazi, E., & Aboalsamh, H. (2018). An automated system for epilepsy detection using EEG brain signals based on deep learning approach. *Expert Systems with Applications*, 107, 61-71.

Vidyaratne, L., Glandon, A., Alam, M., & Iftekharuddin, K. M. (2016). Deep recurrent neural network for seizure detection. 2016 *International Joint Conference on Neural Networks (IJCNN)*, 1202-1207.

Wang, S. H., Phillips, P., & Yang, M. (2021). Identification of Alzheimer's disease using a convolutional neural network model. *Computers in Biology and Medicine*, 123, 103895.

Wang, S. H., Phillips, P., & Yang, M. (2021). Transformer-based model for schizophrenia diagnosis. *Neural Computing and Applications*, 33(15), 9335-9348.

Wang, X., Wang, Y., Liu, D. Wang, Y., Wang, Z. (2023). Automated recognition of epilepsy from EEG signals using a combining space–time algorithm of CNN-LSTM. *Scientific Reports,* 13:14876.

Wei Y, Zhou J, Wang Y, Liu Y, Liu Q, Luo J, Wang C, Ren F, Huang L (2020) A Review of Algorithm & Hardware Design for AI-Based Biomedical Applications. I*EEE Trans Biomed Circuits Syst* 14(2):145-163.

Wei, X., Zhou, L., Chen, Z., Zhang, L., & Zhou, Y. (2018). Automatic seizure detection using three-dimensional CNN based on multi-channel EEG. *Neurocomputing*, 18, 111.

Wirrell E, Tinuper P, Perucca E, Mosh SL. Introduction to the epilepsy syndrome papers. *Epilepsia*. 2022;63:1330–1332.

Xie, Y., Zhang, D., & Jiang, H. (2020). Attention-based CNN-LSTM networks for seizure prediction. *Biomedical Signal Processing and Control*, 55, 101641.

Xu, Y., Yu, Z., Li, Y., Liu, Y., Li, Y., Wang, Y. (2024). Autism spectrum disorder diagnosis with EEG signals using time series maps of brain functional connectivity and a combined CNN–LSTM model. *Computer Methods and Programs in Biomedicine*, 250,108196.

Yang, J., Wang, Y., & Li, X. (2019). RNN-based early detection of Alzheimer's disease from EEG. *Neurocomputing*, 335, 325-332.

Zhao, W., Liu, H., & Zhang, M. (2023). CNN-LSTM model for schizophrenia classification using EEG. *IEEE Transactions on Biomedical Engineering*, 70(3), 1234-1243.

Johnson, J.M., Khoshgoftaar, T.M. (2019). Survey on deep learning with class imbalance. J Big Data 6, 27.

7. APPENDIX: CODE

```
### **Define Time Window**

time_window = 5.0

### **Install/Import General Dependencies**

import os
os.environ["KERAS_BACKEND"] = "jax"
import pandas as pd
import numpy as np

import io
import warnings
import tensorflow as tf
import matplotlib.pyplot as plt

from contextlib import redirect_stdout
from IPython.display import display, HTML

from keras import Sequential
from keras import backend as K
from keras.utils import plot_model
from keras.layers import Layer, Input, Dense, SimpleRNN, Lambda, Dot, Activation, Concatenate, RepeatVector, Add
from keras import ops, datasets, layers, activations, optimizers, initializers, regularizers, losses, metrics, Model

import mne
import math
import json
import zipfile
import tensorflow
import concurrent.futures
import pyarrow as pa


import joblib
import matplotlib.pyplot as plt
import pyarrow.parquet as pq
import json

from time import time
from glob import glob
from sklearn import metrics as sk_metrics

from tqdm.notebook import tqdm
from pyarrow.parquet import ParquetFile, ParquetDataset
from scipy.signal import butter, filtfilt, iirnotch
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_curve

from pycm import ConfusionMatrix

warnings.filterwarnings('ignore')
```

```
### **Define Parameters**

# Parameters
order = 5
fs = 250  # Sample rate, Hz
lowcut = 0.5  # Low-frequency cutoff, Hz
highcut = 45.0  # High-frequency cutoff, Hz
notch_freq = 50.0  # Notch filter frequency, Hz
notch_quality = 30
batch_size = 100000 # Parquet file batch length
hms_sfreq = 100
local_sfreq = 100
# References:
# - https://arxiv.org/pdf/1508.04025.pdf (Luong).
# https://github.com/philipperemy/keras-attention

### **Create Custom Attention Layer**

class Attention(Layer):
    def __init__(self, units: int = 128, **kwargs):
        super(Attention, self).__init__(**kwargs)
        self.units = units
        self.score = 'luong'

    def build(self, input_shape):
        input_dim = int(input_shape[-1])
        with tf.name_scope(self.name):
            self.luong_layer = Dense(input_dim, use_bias=False)
            self.luong_dot_product = Dot(axes=[1, 2])

            self.hidden_layer = Lambda(lambda x: x[:, -1, :], output_shape=(input_dim,))
            self.softmax_layer = Activation('softmax')

            self.dot_product_context = Dot(axes=[1, 1])
            self.concatenate = Concatenate()
            self.attention_vector = Dense(self.units, use_bias=False, activation='tanh')

    def compute_output_shape(self, input_shape):
        return input_shape[0], self.units

    def __call__(self, inputs, **kwargs):
        return super(Attention, self).__call__(inputs, **kwargs)

    def call(self, inputs, **kwargs):
        """
        Many-to-one attention mechanism for Keras. Following Luong's multiplicative style.
        @param inputs: 3D tensor with shape (batch_size, time_steps, input_dim).
        @return: 2D tensor with shape (batch_size, units)
        """
        input_layer = inputs
        hidden_layer = self.hidden_layer(input_layer)

        softmax_score = self.luong_dot_product([hidden_layer, self.luong_layer(input_layer)])
        alpha_score = self.softmax_layer(softmax_score)

        context_vector = self.dot_product_context([input_layer, alpha_score])
```

```
        return self.attention_vector(
             self.concatenate([context_vector, hidden_layer])
        )

    def get_config(self):
        config = super(Attention, self).get_config()
        config.update({'units': self.units, 'score': self.score})
        return config

### **Create Auxiliary Functions**

# List of channels to include in the average reference calculation
channels_to_include = ['F3', 'C3', 'P3', 'F7', 'T3', 'T5', 'Fz', 'Cz', 'Pz', 'F4', 'C4', 'P4', 'F8', 'T4', 'T6']

replace_dict_hms = {
    "Normal": 0, "Seizure": 1, "GPD": 99, "LPD": 99, "GRDA": 99, "LRDA": 99, "Other": 99
}
replace_dict_local = {
    "Normal": 0, "FSCZ": 1, "GSCZ": 2, "PSCZ": 3, "TSCZ": 4, "Artifacts": 99, "Others": 99,
    "Others - SLOWING": 99, "Others - Slowing": 99, "Others - slowing": 99,
    "Others - dysryhtmia": 99, "Others - dysrythmia": 99, "Others - dysrythmia": 99
}
replace_dict_bin_hms = {
    "Normal": 0, "Seizure": 1, "GPD": 99, "LPD": 99, "GRDA": 99, "LRDA": 99, "Other": 99
}
replace_dict_bin_local = {
    "Normal": 0, "FSCZ": 1, "GSCZ": 1, "PSCZ": 1, "TSCZ": 1, "Artifacts": 99, "Others": 99,
    "Others - SLOWING": 99, "Others - Slowing": 99, "Others - slowing": 99,
    "Others - dysryhtmia": 99, "Others - dysrythmia": 99, "Others - dysrythmia": 99
}

def apply_bandpass_filter(data, lowcut, highcut, fs, order = 5):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, data)

def apply_notch_filter(data, freq, fs, quality = 30):
    nyquist = 0.5 * fs
    freq = freq / nyquist
    b, a = iirnotch(freq, quality)
    return filtfilt(b, a, data)

def filter_data(
    arr,
    sfreq,
    lowcut=lowcut,
    highcut=highcut,
    notch_freq=notch_freq,
    notch_quality=notch_quality,
    order=order,
):
    nyquist = 0.5 * sfreq

    if lowcut and highcut:
        low = lowcut / nyquist
        high = highcut / nyquist
```

```
        freq = notch_freq / nyquist

        try:
            b, a = butter(order, [low, high], btype='band')
            arr = filtfilt(b, a, arr)
        except:
            pass
        try:
            b, a = iirnotch(freq, notch_quality)
            arr = filtfilt(b, a, arr)
        except:
            pass

    return arr

def uniform_shapes(l, min_time_window):
    res = []
    for i in l:
        if i.shape[0] == min_time_window:
            res.append(i)
    return res

def encode_array(labels_1, labels_2):
    labels_1_encoded = np.zeros(labels_2.shape)
    labels_1_encoded[np.arange(labels_1.size),labels_1] = 1
    return labels_1_encoded

def multi_class_label_hms(labels):
    return np.array([replace_dict_hms[label] for label in labels])

def multi_class_label_local(labels):
    return np.array([replace_dict_local[label] for label in labels])

def bin_label_hms(labels):
    return np.array([replace_dict_bin_hms[label] for label in labels])

def bin_label_local(labels):
    return np.array([replace_dict_bin_local[label] for label in labels])

def encode_label(labels_1, labels_2):
    labels_test = np.array([replace_dict[label] for label in labels_test])
    return encode_array(labels_1, labels_2)

def max_idx_label(arr):
    b = np.zeros_like(arr)
    b[np.arange(len(arr)), arr.argmax(1)] = 1
    return b

def standard_scaling(data):
    mean = np.mean(data, axis=1, keepdims=True)
    std = np.std(data, axis=1, keepdims=True)
    return (data - mean) / std

def min_max_scaling(data):
    min_val = np.min(data)
    max_val = np.max(data)
    return (data - min_val) / (max_val - min_val)
```

```
def remove_dc_offset(data):
    return data - np.mean(data, axis=1, keepdims=True)

def preprocess_data(
    raw,
    channels = channels_to_include
):

    # Pick the desired channels
    raw.pick_channels(channels)

    # Preprocess HMS data
    arr = raw.get_data(verbose='ERROR')

    # Remove DC offset
    arr = remove_dc_offset(arr)

    # Normalize data
    arr = standard_scaling(arr)

    # Get sfreq
    sfreq = raw.info["sfreq"]

    # Filter DataFrame
    arr = filter_data(
        arr=arr,
        sfreq=sfreq,
    )

    # Convert filtered data back to DataFrame
    df = pd.DataFrame(arr.T, columns=channels)

    # Set index
    df = df.set_index(
        pd.to_datetime(
            [
                1000 * (item / sfreq) for item in list(range(len(df)))
            ],
            unit='ms',
            origin=pd.Timestamp('1900-01-01')
        )
    )

    # Interpolate
    df = df.resample('10ms').mean().dropna()

    # Calculate the average reference signal
    average_reference = df.mean(axis = 1)

    # Adjust the channels to the AR montage
    df = df.sub(average_reference, axis = 0)

    return df

def preprocess_data_local(
    raw,
```

```
    channels = channels_to_include
):

    # Pick the desired channels
    raw.pick_channels(channels)

    # Preprocess HMS data
    arr = raw.get_data(verbose='ERROR')

    # Remove DC offset
    arr = remove_dc_offset(arr)

    # Normalize data
    arr = standard_scaling(arr)

    # Get sfreq
    sfreq = raw.info["sfreq"]

    # Filter DataFrame
    #arr = filter_data(
    #    arr=arr,
    #    sfreq=sfreq,
    #)

    # Convert filtered data back to DataFrame
    df = pd.DataFrame(arr.T, columns=channels)

    # Set index
    df = df.set_index(
        pd.to_datetime(
            [
                1000 * (item / sfreq) for item in list(range(len(df)))
            ],
            unit='ms',
            origin=pd.Timestamp('1900-01-01')
        )
    )

    # Interpolate
    df = df.resample('10ms').mean().dropna()

    return df

def plot_data(
    arr,
    start_time,
    end_time,
    type="HMS",
    sfreq=200,
    channels=['F3', 'C3', 'P3', 'F7', 'T3', 'T5', 'Fz', 'Cz', 'Pz', 'F4', 'C4', 'P4', 'F8', 'T4', 'T6'],
):

    # Create time vector
    time_vector = np.linspace(0, arr.shape[0] / sfreq, num=len(arr))

    start_index = int(start_time * sfreq)
    end_index = int(end_time * sfreq)
```

```
    # Plot the selected time window for the specified channels
    fig, axs = plt.subplots(len(channels), 1, figsize=(len(channels), 8), dpi=5, sharex=True)
    fig.set_size_inches(200, 200)

    for i, channel in enumerate(channels):
        axs[i].plot(time_vector[start_index:end_index], arr[start_index:end_index, i])
        axs[i].set_ylabel('Amplitude (μV)')
        axs[i].set_title(f'{type} EEG Signal: {channel} ({str(start_time)}-{str(end_time)} seconds)')
    plt.xlabel('Time (seconds)')
    plt.suptitle(f'Normalized {type} EEG Signals ({str(start_time)}-{str(end_time)} seconds)', y=1.02)
    plt.tight_layout()
    plt.show()

def plot_metrics(
    i,
    y_true,
    y_pred,
    y_pred_scores,
    test_description
):
    # Calculate metrics
    if set(y_true) != {1}:
        i[test_description + ': ROC AUC'] = sk_metrics.roc_auc_score(
            y_true,
            y_pred_scores,
        )
        i[test_description + ': RC'] = sk_metrics.roc_curve(
            y_true,
            y_pred_scores,
        )
        i[test_description + ': PR'] = sk_metrics.precision_recall_curve(
            y_true,
            y_pred_scores,
        )
    else:
        i[test_description + ': ROC AUC'] = np.nan
        i[test_description + ': RC'] = np.nan
        i[test_description + ': PR'] = np.nan

    i[test_description + ': CM'] = sk_metrics.confusion_matrix(
        y_true,
        y_pred
    )
    i[test_description + ': FAR'] = i[test_description + ': CM'][0, 1] / (
        i[test_description + ': CM'][0, 0] + i[test_description + ': CM'][0, 1]
    )
    i[test_description + ': F1 Score'] = sk_metrics.f1_score(
        y_true,
        y_pred
    )
    i[test_description + ': Accuracy'] = sk_metrics.accuracy_score(
        y_true,
        y_pred
    )
    i[test_description + ': Precision'] = sk_metrics.precision_score(
        y_true,
        y_pred
```

```
    )
    i[test_description + ': Average Precision'] = sk_metrics.average_precision_score(
        y_true,
        y_pred
    )
    i[test_description + ': Recall'] = sk_metrics.recall_score(
        y_true,
        y_pred
    )
    i[test_description + ': Jaccard'] = sk_metrics.jaccard_score(
        y_true,
        y_pred
    )
    print(f"\033[1m{test_description}: Model: {i['description']}\033[0m")
    print("")

    print(f"FAR: {i[test_description + ': FAR']}")
    print(f"ROC AUC: {i[test_description + ': ROC AUC']}")
    print(f"F1 score: {i[test_description + ': F1 Score']}")
    print(f"Accuracy: {i[test_description + ': Accuracy']}")
    print(f"Precision: {i[test_description + ': Precision']}")
    print(f"Average precision: {i[test_description + ': Average Precision']}")
    print(f"Recall: {i[test_description + ': Recall']}")
    print(f"Jaccard: {i[test_description + ': Jaccard']}")
    print("")

    if set(y_true) != {1}:
        fpr, tpr, threshold = i[test_description + ': RC']
        plt.title(f"{test_description}: {i['description']}: Receiver Operating Characteristic")
        plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % i[test_description + ': ROC AUC'])
        plt.legend(loc = 'lower right')
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.show()
        print("")
        print("")

    return i

### **1. Loading Data**

hms_base_path = "harmful-brain-activity-classification-hms-set"

local_base_path = "eeg_local"
local_annotations_path = os.path.join("eeg_local", "annotations")

csv_file = f'{hms_base_path}/train.csv'

# Load the meta-data from the HMS data .csv file
labels_reference = pd.read_csv(csv_file)

### **2. Data Preprocessing**

### **2.1. Data Preprocessing:** Slicing and Denoising
```

```
#### **2.1.1. Data Preprocessing:** HMS

#====================================
#              HMG Data
#====================================

eeg__hms_filename, eeg__hms_offset, eeg__hms_features, eeg_hms_labels = [], [], [], []

def run_hms_preprocess(filename):
    # Get list of unique offsets
    eeg_offsets = labels_reference[
        labels_reference['eeg_id'] == int(filename.split(".")[0])
    ]['eeg_label_offset_seconds'].unique()

    # Read parquet file
    parquet_file = pq.ParquetFile(
        os.path.join(os.path.join(hms_base_path, "train_eegs"), filename)
    )

    # Read parquet dataset
    parquet_dataset = ParquetDataset(
        os.path.join(os.path.join(hms_base_path, "train_eegs"), filename),
        use_legacy_dataset=False
    )

    # Obtain batch size
    batch_size = sum(p.count_rows() for p in parquet_dataset.fragments)

    assert batch_size >= 50 * hms_sfreq

    # Iterate over parquet batches
    for idx, batch in enumerate(parquet_file.iter_batches(batch_size=batch_size)):

        # Read batch
        df = batch.to_pandas().reset_index()[channels_to_include]

        # Convert to mne
        raw_info = mne.create_info(
            channels_to_include,
            sfreq=hms_sfreq,
            ch_types='eeg',
            verbose='ERROR'
        )

        raw_hms = mne.io.RawArray(df.values.T, raw_info)

        # Preprocess data
        df = preprocess_data(raw_hms)

        # Build list to identify the time windows with normal conditions
        timeframe = [True] * df.shape[0]

        # Loop over the conditions according to expert consensus
        for event in ["Seizure"]:

            # Loop over offsets
            for eeg_offset in eeg_offsets:
```

```
                expert_consensus = labels_reference[
                    labels_reference['eeg_id'] == int(filename.split(".")[0])
                ][
                    labels_reference['eeg_label_offset_seconds'] == eeg_offset
                ]['expert_consensus']

                # Slice central time windows of 50 seconds based on offset
                if (eeg_offset >= (batch_size * idx / hms_sfreq)) and ((eeg_offset + 50) <= (batch_size * (idx +
1)/ hms_sfreq)):
                    # Slice time window of 50 seconds
                    eeg_50s_arr_features = df.values[
                        int(
                            eeg_offset - (batch_size * idx / hms_sfreq)
                        ) * hms_sfreq : int(
                            eeg_offset - (batch_size * idx / hms_sfreq) + 50
                        ) * hms_sfreq,
                        :
                    ]
                    # Slice time windows
                    eeg__arr_features = []
                    for slice in range(int(50 / time_window)):
                        eeg__arr_features.append(
                            eeg_50s_arr_features[
                                int((slice * time_window) * hms_sfreq) : int((slice * time_window + time_window) *
hms_sfreq)
                            ]
                        )

                    if expert_consensus.iloc[0] == event:
                        # Mark instances of the original dataset that overlap with the time windows of 50 seconds
                        timeframe[
                            int(
                                eeg_offset - (batch_size * idx / hms_sfreq)
                            ) * hms_sfreq : int(
                                eeg_offset - (batch_size * idx / hms_sfreq) + 50
                            ) * hms_sfreq
                        ] = [
                            False
                        ]*len(
                            timeframe[
                                int(
                                    eeg_offset - (batch_size * idx / hms_sfreq)
                                ) * hms_sfreq : int(
                                    eeg_offset - (batch_size * idx / hms_sfreq) + 50
                                ) * hms_sfreq
                            ]
                        )
                        # Append the features and label the instances as with the condition according to expert
consensus
                        eeg__hms_filename.append(filename)
                        eeg__hms_offset.append(eeg_offset)

                        for eeg__arr in eeg__arr_features:
                            if eeg__arr.shape == (int(time_window * hms_sfreq), len(channels_to_include)):
                                eeg__hms_features.append(eeg__arr)
                                eeg_hms_labels.append(event)

        # Obtain instances of the original dataset that do not overlap with any of the time windows of 50 seconds
```

```
        normal_condition_arr_features = df.reset_index(drop=True)[timeframe]

        # Slice the instances within the intervals of normal condition into time windows of 50 seconds
        for normal_slice in range(int(len(normal_condition_arr_features) / (50 * hms_sfreq))):
            eeg_50s_arr_normal_features = normal_condition_arr_features.values[
                normal_slice * hms_sfreq : (normal_slice + 50) * hms_sfreq,
                :
            ]
            # Slice time windows
            eeg__arr_features = []
            for slice in range(int(50 / time_window)):
                eeg__arr_features.append(
                    eeg_50s_arr_normal_features[
                        int((slice * time_window) * hms_sfreq) : int((slice * time_window + time_window) * hms_sfreq)
                    ]
                )

            # Append the features and label the instances as with the normal condition
            for eeg__arr in eeg__arr_features:
                if eeg__arr.shape == (int(time_window * hms_sfreq), len(channels_to_include)):
                    eeg__hms_features.append(eeg__arr)
                    eeg_hms_labels.append("Normal")

#Calling the function
with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
    executor.map(
        run_hms_preprocess,
        os.listdir(os.path.join(hms_base_path, "train_eegs"))[0:1000]
    )

eeg__hms_features_array = np.array(eeg__hms_features)

print(f"(hms) {time_window}-second window dimension:", eeg__hms_features_array.shape)

#### **2.1.2. Data Preprocessing:** HMS - Remove unvalid classes

eeg__hms_features_ = []
eeg_hms_labels_ = []

unvalid_classes = list(dict((k, v) for k, v in replace_dict_hms.items() if v == 99).keys())

valid_classes = list(
    pd.Series(eeg_hms_labels).value_counts()[pd.Series(eeg_hms_labels).value_counts() > 1].index
)

for idx, label in enumerate(eeg_hms_labels):
    if (label in valid_classes) and (label not in unvalid_classes):
        eeg__hms_features_.append(eeg__hms_features[idx])
        eeg_hms_labels_.append(label)

eeg__hms_features_array = np.array(eeg__hms_features_)
eeg_hms_labels = eeg_hms_labels_

#### **2.1.3. Data Preprocessing:** Local

#====================================
#             Local Data
#====================================
```

```
eeg__local_filename, eeg__local_offset, eeg__local_features, eeg_local_labels = [], [], [], []

def run_local_preprocess(filename):

    # Read fif file
    if "fif" in filename:

        # Read mne
        raw_local = mne.io.read_raw_fif(
            os.path.join(local_base_path, filename),
            preload=True
        )

        # Preprocess data
        df = preprocess_data_local(raw_local)

        # Build list to identify the time windows with normal conditions
        timeframe = [True] * df.shape[0]

        # Read metadata
        json_filename = filename.replace("fif", "json")
        try:
            with open(os.path.join(local_annotations_path, json_filename)) as json_data:
              events = json.load(json_data)
        except:
            events = []

        # Iterate over time windows annotated
        for event in events:

            # Slice according to metadata event
            eeg__arr_features = []
            #if (
            #    int((event["start_time"] - time_window / 2) * local_sfreq) >= 0
            #) and (
            #    int((event["stop_time"] + time_window / 2) * local_sfreq) <= len(df)
            #):
            slices = (event["stop_time"] - event["start_time"]) / time_window

            for slice in range(int(slices)):
                # Mark instances of the original dataset that overlap with the time windows of 50 seconds
                timeframe[
                    int(
                        (event["start_time"] + time_window * slice) * local_sfreq
                    ) : int(
                        (event["start_time"] + time_window * (slice + 1)) * local_sfreq
                    )
                ] = [
                    False
                ]*len(
                    timeframe[
                        int(
                            (event["start_time"] + time_window * slice) * local_sfreq
                        ) : int(
                            (event["start_time"] + time_window * (slice + 1)) * local_sfreq
                        )
                    ]
```

```
                )
                # Append instance with the event
                eeg__arr_features.append(
                    df.iloc[
                        int(
                            (event["start_time"] + time_window * slice) * local_sfreq
                        ) : int(
                            (event["start_time"] + time_window * (slice + 1)) * local_sfreq
                        ), :
                    ].values
                )

            # Slice time windows
            for eeg__arr in eeg__arr_features:
                if eeg__arr.shape == (int(time_window * local_sfreq), len(channels_to_include)):
                    eeg__local_features.append(eeg__arr)
                    eeg_local_labels.append(event["onset"])

        # Obtain instances of the original dataset that do not overlap with any of the time windows
        normal_condition_arr_features = df.reset_index(drop=True)[timeframe].iloc[
            0 : int(10 * (time_window * local_sfreq)),
            :
        ]

        # Slice the instances within the intervals of normal condition into time windows
        eeg__arr_features = []
        for normal_slice in range(int(len(normal_condition_arr_features) / (time_window * local_sfreq))):
            eeg__arr_features.append(
                normal_condition_arr_features.values[
                    int((normal_slice * time_window) * local_sfreq) : int((normal_slice * time_window + time_window)
* local_sfreq)
                ]
            )

        # Append the features and label the instances as with the normal condition
        for eeg__arr in eeg__arr_features:
            if eeg__arr.shape == (int(time_window * local_sfreq), len(channels_to_include)):
                eeg__local_features.append(eeg__arr)
                eeg_local_labels.append("Normal")

# Calling the function
with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
    executor.map(
        run_local_preprocess,
        os.listdir(local_base_path)
    )

eeg__local_features_array = np.array(eeg__local_features)

print(f"(local) {time_window}-second window dimension:", eeg__local_features_array.shape)

#### **2.1.4. Data Preprocessing:** Local - Remove unvalid classes

eeg__local_features_ = []
eeg_local_labels_ = []

unvalid_classes = list(dict((k, v) for k, v in replace_dict_local.items() if v == 99).keys())
```

```
valid_classes = list(
    pd.Series(eeg_local_labels).value_counts()[pd.Series(eeg_local_labels).value_counts() > 1].index
)

for idx, label in enumerate(eeg_local_labels):
    if (label in valid_classes) and (label not in unvalid_classes):
        eeg__local_features_.append(eeg__local_features[idx])
        eeg_local_labels_.append(label)

eeg__local_features_array = np.array(eeg__local_features_)
eeg_local_labels = eeg_local_labels_
```

2.3. Data Preprocessing: Train/Test Split

```
#====================================
#              HMG Data
#====================================

(
    eeg_hms_train_features_array,
    eeg_hms_test_features_array,
    eeg_hms_train_labels,
    eeg_hms_test_labels
) = train_test_split(
    eeg__hms_features_array,
    eeg_hms_labels,
    stratify = eeg_hms_labels,
    test_size = 0.20,
    random_state = 42
)

(
    eeg_hms_train_features_array,
    eeg_hms_valid_features_array,
    eeg_hms_train_labels,
    eeg_hms_valid_labels
) = train_test_split(
    eeg_hms_train_features_array,
    eeg_hms_train_labels,
    stratify = eeg_hms_train_labels,
    test_size = 0.20,
    random_state = 42
)

print(f'1. Dimension of {time_window}-seconds eeg segment labels and features: HMS train, valid and test data')
print('   HMS Train features:', eeg_hms_train_features_array.shape)
print('   HMS Train labels  :', np.array(eeg_hms_train_labels).shape)
print('   HMS Valid  features:', eeg_hms_valid_features_array.shape)
print('   HMS Valid  labels  :', np.array(eeg_hms_valid_labels).shape)
print('   HMS Test  features:', eeg_hms_test_features_array.shape)
print('   HMS Test  labels  :', np.array(eeg_hms_test_labels).shape)
print()

#====================================
#             Local Data
#====================================

(
```

```
    eeg_local_train_features_array,
    eeg_local_test_features_array,
    eeg_local_train_labels,
    eeg_local_test_labels
) = train_test_split(
    eeg__local_features_array,
    eeg_local_labels,
    stratify = eeg_local_labels,
    test_size = 0.20,
    random_state = 42
)

#(
#    eeg_local_train_features_array,
#    eeg_local_valid_features_array,
#    eeg_local_train_labels,
#    eeg_local_valid_labels
#) = train_test_split(
#    eeg_local_train_features_array,
#    eeg_local_train_labels,
#    stratify = eeg_local_train_labels,
#    test_size = 0.20,
#    random_state = 42
#)

print(f'2. Dimension of {time_window}-seconds eeg segment labels and features: Local train, valid and test data')
print('   Local Train features:', eeg_local_train_features_array.shape)
print('   Local Train labels  :', np.array(eeg_local_train_labels).shape)
#print('   Local Valid  features:', eeg_local_valid_features_array.shape)
#print('   Local Valid  labels  :', np.array(eeg_local_valid_labels).shape)
print('   Local Test  features:', eeg_local_test_features_array.shape)
print('   Local Test  labels  :', np.array(eeg_local_test_labels).shape)

### **3. Model Designing**

### **3.1. Model Designing:** Architecture

#====================================
#     Multi-class Architecture
#====================================

def build_multi_class_model(arr, recurrent_layer):
    """
    Build a convolutional LSTM neural network model for numerical data.

    Parameters:
    arr (numpy array): Input array of shape (samples, time_steps, features).

    Returns:
    keras.Model: Compiled Keras model for classification.
    """
    return Sequential([

        # First convolutional layer followed by batch normalization, max pooling, and dropout
        layers.Conv1D(
            input_shape=(arr.shape[1], arr.shape[2]),
            filters = 8, kernel_size = 2, strides = 1, activation = 'relu', padding = 'same'),
        layers.BatchNormalization(),
```

```
        layers.MaxPooling1D(pool_size = 2),
        layers.Dropout(rate = 0.2),

        # Second convolutional layer with similar structure
        layers.Conv1D(filters = 16, kernel_size = 2, strides = 1, activation = 'relu', padding = 'same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size = 2),
        layers.Dropout(rate = 0.2),

        # Third convolutional layer with similar structure
        layers.Conv1D(filters = 32, kernel_size = 2, strides = 1, activation = 'relu', padding = 'same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size = 2),
        layers.Dropout(rate = 0.2),

        # Getting LSTM output
        layers.TimeDistributed(layers.Dense(64)),
        layers.Bidirectional(recurrent_layer(64, return_sequences = True), name = "bi_lstm_0"),
        Attention(units=64),
        #layers.Dense(len(eeg_local_labels_unique)),
        layers.Dense(len(eeg_local_labels_unique), activation='softmax')])

#====================================
#        Binary Architecture
#====================================

def build_bin_model(arr, recurrent_layer):
    """
    Build a binary classification model using Conv1D and LSTM layers.
    Parameters: arr(numpy array): Input array with shape (samples, time_steps, features).
    Returns: keras.Model: Compiled Keras model for binary classification.
    """
    return Sequential([
        # First convolutional layer followed by batch normalization, max pooling, and dropout
        layers.Conv1D(
            input_shape = (arr.shape[1], arr.shape[2]),
            filters = 8, kernel_size = 2, strides = 1, activation = 'relu', padding = 'same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size = 2),
        layers.Dropout(rate = 0.2),

        # Second convolutional layer with similar structure
        layers.Conv1D(filters = 16, kernel_size = 2, strides = 1, activation = 'relu', padding = 'same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size = 2),
        layers.Dropout(rate = 0.2),

        # Third convolutional layer with similar structure
        layers.Conv1D(filters = 32, kernel_size = 2, strides = 1, activation = 'relu', padding = 'same'),
        layers.BatchNormalization(),
        layers.MaxPooling1D(pool_size = 2),
        layers.Dropout(rate = 0.2),

        # Getting Bi-LSTM output
        layers.TimeDistributed(layers.Dense(64)),
        layers.Bidirectional(recurrent_layer(64, return_sequences = True), name = "bi_lstm_0"),
        Attention(units = 64),
        #layers.Dense(64),
```

```
        layers.Dense(1, activation = 'sigmoid')])

### **3.2. Model Designing:** Architecture Summary

# Summary function
def get_model_summary(model):

    stream = io.StringIO()
    with redirect_stdout(stream):
        model.summary()
    return stream.getvalue()

# Building numeric models
summaries_multi_class_local, summaries_bin_hms, summaries_bin_local = [], [], []

#====================================
#        Multi-class Model
#====================================

print()
print('Summary of multi-class model (trained with local dataset to predict the seizure type)')
print()

# Building multi-class models
model_multi_class_local = {
    "model_multi_class_lstm_local": build_multi_class_model(eeg_local_train_features_array, layers.LSTM),
    "model_multi_class_gru_local": build_multi_class_model(eeg_local_train_features_array, layers.GRU)
}

for name, model in model_multi_class_local.items():
    summaries_multi_class_local.append(get_model_summary(model))

#====================================
#           Binary Models
#====================================

print()
print('Summary of binary model (trained with both HMS and local datasets to predict the occurrence of a seizure)')
print()

# Building binary models
model_bin_hms = {
    "model_bin_lstm_hms": build_bin_model(eeg_hms_train_features_array, layers.LSTM),
    "model_bin_gru_hms": build_bin_model(eeg_hms_train_features_array, layers.GRU)
}

model_bin_local = {
    "model_bin_lstm_local": build_bin_model(eeg_local_train_features_array, layers.LSTM),
    "model_bin_gru_local": build_bin_model(eeg_local_train_features_array, layers.GRU)
}

for name, model in model_bin_hms.items():
    summaries_bin_hms.append(get_model_summary(model))

for name, model in model_bin_local.items():
    summaries_bin_local.append(get_model_summary(model))
```

```
### **4. Model Pre-Training and Prediction**

#====================================
#        Local Multi-class Model
#====================================

print('Training of local multi-class model')
print()

# Build models
model_multi_class_local = {
    "model_multi_class_lstm_local": build_multi_class_model(eeg_local_train_features_array, layers.LSTM),
    "model_multi_class_gru_local": build_multi_class_model(eeg_local_train_features_array, layers.GRU)
}

# Defining models and feature arrays using model_multi_class dictionary created earlier
(
    model_multi_class_lstm_local,
    model_multi_class_gru_local
) = (
    model_multi_class_local["model_multi_class_lstm_local"],
    model_multi_class_local["model_multi_class_gru_local"]
)

# Defining models
models_multi_class_local = [
    {
        'description': 'Local - LSTM Multi Class Seizure Types',
        'model': model_multi_class_lstm_local,
        'epochs': 2,
        'test_local_pred': []
    },
    {
        'description': 'Local - GRU Multi Class Seizure Types',
        'model': model_multi_class_gru_local,
        'epochs': 2,
        'test_local_pred': []
    }
]

# Weight vector
multi_class_class_weight = {}

multi_class_train_labels = pd.Series(multi_class_label_local(eeg_local_train_labels))

for label in multi_class_train_labels.unique():
    multi_class_class_weight[label] = int(
        multi_class_train_labels.shape[0] / multi_class_train_labels.value_counts()[label]
    )

# Compiling, training and predicting with each model
for idx, i in enumerate(models_multi_class_local):

    # Compiling model
    i['model'].compile(
        optimizer = optimizers.RMSprop(learning_rate = 1e-3),
        loss = losses.SparseCategoricalCrossentropy(),
        metrics = [metrics.SparseCategoricalAccuracy()],
```

```
        run_eagerly = True
    )

    # Training model
    i['model'].fit(
        x = eeg_local_train_features_array,
        y = multi_class_label_local(eeg_local_train_labels),
        batch_size = 100,
        epochs = i['epochs'],
        verbose = "auto",
        validation_split = 0.2,
        class_weight = multi_class_class_weight,
        shuffle = True
    )

    # Predict with model
    models_multi_class_local[idx][
        'test_local_pred'] = i['model'].predict(eeg_local_test_features_array)
    print()

#====================================
#           HMS Binary Model
#====================================

print('HMS - Training of HMS binary model')
print()

model_bin_hms = {
    "model_bin_lstm_hms": build_bin_model(eeg_hms_train_features_array, layers.LSTM),
    "model_bin_gru_hms": build_bin_model(eeg_hms_train_features_array, layers.GRU)
}

model_bin_lstm_hms, model_bin_gru_hms = model_bin_hms["model_bin_lstm_hms"], model_bin_hms["model_bin_gru_hms"]

# Defining models
models_bin_hms = [
    {
        'description': 'HMS - LSTM Seizure X Not Seizure',
        'model': model_bin_lstm_hms,
        'epochs': 10,
        'test_local_pred': [],
        'test_hms_pred': []
    },
    {
        'description': 'HMS - GRU Seizure X Not Seizure',
        'model': model_bin_gru_hms,
        'epochs': 10,
        'test_local_pred': [],
        'test_hms_pred': []
    }
]

# Weight vector
binary_hms_class_weight = {}

binary_hms_train_labels = pd.Series(bin_label_hms(eeg_hms_train_labels))
```

```
for label in binary_hms_train_labels.unique():
    binary_hms_class_weight[label] = int(
        binary_hms_train_labels.shape[0] / binary_hms_train_labels.value_counts()[label]
    )

# Compiling, training and predicting with each model
for idx, i in enumerate(models_bin_hms):

    # Compiling model
    i['model'].compile(
        optimizer = optimizers.Adam(learning_rate = 1e-3),
        loss = losses.BinaryCrossentropy(),
        metrics =  [
            metrics.BinaryAccuracy(),
            metrics.FalseNegatives(),
            metrics.FalsePositives(),
            metrics.TrueNegatives(),
            metrics.TruePositives()
        ]
    )

    # Training model
    i['model'].fit(
        x = eeg_hms_train_features_array,
        y = bin_label_hms(eeg_hms_train_labels),
        batch_size = 100,
        epochs = i['epochs'],
        verbose = "auto",
        validation_data=(
            eeg_hms_valid_features_array,
            bin_label_hms(eeg_hms_valid_labels)
        ),
        class_weight = binary_hms_class_weight,
        shuffle = True
    )

#====================================
#           Local Binary Model
#====================================

print('Local - Training of Local binary model')
print()

model_bin_local = {
    "model_bin_lstm_local": build_bin_model(eeg_local_train_features_array, layers.LSTM),
    "model_bin_gru_local": build_bin_model(eeg_local_train_features_array, layers.GRU)
}

model_bin_lstm_local,           model_bin_gru_local           =           model_bin_local["model_bin_lstm_local"],
model_bin_local["model_bin_gru_local"]

# Defining models
models_bin_local = [
    {
        'description': 'Local - LSTM Seizure X Not Seizure',
        'model': model_bin_lstm_local,
        'epochs': 10,
        'test_local_pred': [],
```

```
        'test_hms_pred': []
    },
    {
        'description': 'Local - GRU Seizure X Not Seizure',
        'model': model_bin_gru_local,
        'epochs': 10,
        'test_local_pred': [],
        'test_hms_pred': []
    }
]

# Weight vector
binary_local_class_weight = {}

binary_local_train_labels = pd.Series(bin_label_local(eeg_local_train_labels))

for label in binary_local_train_labels.unique():
    binary_local_class_weight[label] = int(
        binary_local_train_labels.shape[0] / binary_local_train_labels.value_counts()[label]
    )

# Compiling, training and predicting with each model
for idx, i in enumerate(models_bin_local):

    # Compiling model
    i['model'].compile(
        optimizer = optimizers.Adam(learning_rate = 1e-3),
        loss = losses.BinaryCrossentropy(),
        metrics =  [
            metrics.BinaryAccuracy(),
            metrics.FalseNegatives(),
            metrics.FalsePositives(),
            metrics.TrueNegatives(),
            metrics.TruePositives()
        ]
    )

    # Training model
    i['model'].fit(
        x = eeg_local_train_features_array,
        y = bin_label_local(eeg_local_train_labels),
        batch_size = 100,
        epochs = i['epochs'],
        verbose = "auto",
        validation_split=0.2,
        class_weight = binary_local_class_weight,
        shuffle = True
    )

### **5. Model Test**

for idx, i in enumerate(models_bin_hms):

    # Predict with model
    models_bin_hms[idx][
        'test_hms_pred'] = i['model'].predict(eeg_hms_test_features_array)
    models_bin_hms[idx][
        'test_local_pred'] = i['model'].predict(eeg_local_test_features_array)
```

```
    print()

for idx, i in enumerate(models_bin_local):

    # Predict with model
    models_bin_local[idx][
        'test_hms_pred'] = i['model'].predict(eeg_hms_test_features_array)
    models_bin_local[idx][
        'test_local_pred'] = i['model'].predict(eeg_local_test_features_array)
    print()

### **6. Model Results**

#====================================
#           Binary Model
#====================================

# HMS data
print("")
print(f"\033[1mHMS: Binary Model\033[0m")

print("")
print(f"\033[1mHMS: Binary Model - Results in HMS Test\033[0m")

for idx, i in enumerate(models_bin_hms):
    i = plot_metrics(
        i,
        bin_label_hms(eeg_hms_test_labels),
        (i['test_hms_pred'] > 0.5).astype(int).flatten(),
        pd.DataFrame(i['test_hms_pred']).values,
        "HMS: Binary Model - Results in HMS Test"
    )
    print("HMS: Binary Model: CM - Results in HMS Test")
    print(i['HMS: Binary Model - Results in HMS Test: CM'])
    print("")
    models_bin_hms[idx] = i

print("")
print(f"\033[1mHMS: Binary Model - Results in Local Test\033[0m")

for idx, i in enumerate(models_bin_hms):
    i = plot_metrics(
        i,
        bin_label_local(eeg_local_test_labels),
        (i['test_local_pred'] > 0.5).astype(int).flatten(),
        pd.DataFrame(i['test_local_pred']).values,
        "HMS: Binary Model - Results in Local Test"
    )
    print("HMS: Binary Model: CM - Results in Local Test")
    print(i['HMS: Binary Model - Results in Local Test: CM'])
    print("")
    models_bin_hms[idx] = i

# Local data
print("")
print(f"\033[1mLocal: Binary Model - Results in HMS Test\033[0m")

for idx, i in enumerate(models_bin_local):
```

```
    i = plot_metrics(
        i,
        bin_label_hms(eeg_hms_test_labels),
        (i['test_hms_pred'] > 0.5).astype(int).flatten(),
        pd.DataFrame(i['test_hms_pred']).values,
        "Local: Binary Model - Results in HMS Test"
    )
    print("Local: Binary Model: CM - Results in HMS Test")
    print(i['Local: Binary Model - Results in HMS Test: CM'])
    print("")
    print()
    models_bin_local[idx] = i

print("")
print(f"\033[1mLocal: Binary Model - Results in Local Test\033[0m")

for idx, i in enumerate(models_bin_local):
    i = plot_metrics(
        i,
        bin_label_local(eeg_local_test_labels),
        (i['test_local_pred'] > 0.5).astype(int).flatten(),
        pd.DataFrame(i['test_local_pred']).values,
        "Local: Binary Model - Results in Local Test"
    )
    print("Local: Binary Model: CM - Results in Local Test")
    print(i['Local: Binary Model - Results in Local Test: CM'])
    print("")
    models_bin_local[idx] = i

#====================================
#        Multi-class Model
#====================================

# Local data
print("")
print(f"\033[1mLocal: Multi-Class Models\033[0m")

for i in models_multi_class_local:
    print(i['description'])
    arr = multi_class_label_local(eeg_local_test_labels)
    arr[arr == 2] = 1
    print(
        classification_report(
            encode_array(
                arr,
                max_idx_label(i['model'].predict(eeg_local_test_features_array))
            ),
            max_idx_label(i['model'].predict(eeg_local_test_features_array)),
            #target_names = [item for item in list(replace_dict_local.keys()) if item != "Artifacts"]
        )
    )
```