CS 395 Deep Learning Project Final Report: Mind Wandering Classifier

Alaina Birney & Omar Awajan

1. Introduction

Maintaining concentration while reading is crucial for understanding concepts within a piece of text and retaining information. Often, one may find their mind "wandering" while reading. In these instances of mind wandering (MW), one thinks about topics unrelated to the text they are reading and fails to truly ingest the meaning behind it. This can lead to lost time because the passage of text must be re-read or missed information because one fails to recognize that their mind was wandering and correct the behavior.

Various other studies, such as "*Mind Wandering in a Multimodal Reading Setting: Behavior Analysis & Automatic Detection Using Eye-Tracking and an EDA Sensor*" [2], have classified mind wandering utilizing standard machine learning approaches, such as random forest. However, these approaches require significant feature engineering, for example calculating the length and angle of saccades from eye gaze coordinate data. We would like to use a neural network instead to see if we can limit the amount of feature engineering that is necessary to create a useful classifier for this purpose.

In this paper, we describe the process of creating a CNN-LSTM to classify whether or not a subject experienced MW while reading a passage of text. These classifications are made at a millisecond-level resolution, using overlapping windows of data as input. To accomplish this, we first created separate models, CNN and LSTM and assessed their ability to classify MW or not MW alone. After running various experiments, we combined the models into a single CNN-LSTM architecture where the CNN acts as a feature extractor for the LSTM.

Ultimately, this could benefit society through integrating such a network into an interface that can notify the user when their mind begins to wander so that the behavior can be corrected. Related work, such as *The Eye–Mind Wandering Link: Identifying Gaze Indices of Mind Wandering Across Tasks* [1], has shown that longer fixations and fewer eye movements are associated with MW. Additionally, studies such as *Mind Wandering in a Multimodal Reading Setting: Behavior Analysis & Automatic Detection Using Eye-Tracking and an EDA Sensor* [2] have found that pupil diameter tends to be smaller during episodes of MW. For these reasons, we intend to use x y coordinates collected from a gaze tracking system while subjects were reading and measures of those subjects' pupil dilation while reading as features for our models.

Our hypothesis was that a clear distinction between focused attention (not MW) and mind wandering (MW) during reading sessions can be made by analyzing temporal patterns in eye behavior without first performing significant feature engineering. Our goal was to achieve an F1 score of at least 0.8 to signify that a clear distinction between not MW and MW during reading sessions was found. This benchmark was chosen based on the results of the random forest classifier in *Mind Wandering in a Multimodal Reading Setting: Behavior Analysis & Automatic Detection Using Eye-Tracking and an EDA Sensor* [2].

In this report, we will present results for the CNN-LSTM model as well as individual CNN and LSTM models, including a description of various strategies that were experimented with. Results will show that our hypothesis is not supported; it is not possible for the CNN-LSTM, CNN, or LSTM to make a clear distinction between not MW and MW through use of the eye tracking data without feature engineering under any of the conditions tested. Because of the significant experimentation we performed, we feel confident in this conclusion and will recommend that more data from a wider variety of subjects is used or additional features are incorporated in future attempts to research this problem.

2. Problem Definition and Algorithm

2.1 Task Definition

To test our hypothesis that a clear distinction between not MW and MW during reading sessions can be made by analyzing temporal patterns in eye behavior without first performing significant feature engineering, we created a CNN-LSTM to classify MW or not MW based on data regarding the x and y coordinates of subjects' left and right eye gazes and the dilation of the subjects' left and right pupils. To create the CNN-LSTM, first, the CNN and LSTM were created individually and their ability to classify MW vs. not MW was evaluated separately. Architecture and hyperparameters were refined until ultimately combining the models. The specific features used in all models are described in the following table (figure 1).

Feature Name	**Description**
LX	Left eye x-coordinate
LY	Left eye y-coordinate
RX	Right eye x-coordinate
RY	Right eye y-coordinate
LPupil_normalized	Left pupil dilation (within-subject normalization applied)
RPupil_normalized	Right pupil dilation (within-subject normalization applied)

Table 1: CNN-LSTM Features

Data was obtained from the Glass Brain Lab's Mindless Reading study. In this study, subjects were each given five passages of text to read, each new passage constitutes a "run" and each "run" consists of 10 pages. While reading, gaze tracking and pupil dilation information was obtained with an EyeLink eye tracker. While reading the text, if at any point the subject noticed that they were experiencing or had experienced MW, they were instructed to press a button, then highlight the portion of text for which their mind was wandering. In total, we have data for nineteen subjects. On average, each passage of text took each subject approximately 7.6 minutes to read. Our data is sampled in milliseconds, so we have approximately 43,320,000 samples before downsampling the train dataset. More details about downsampling can be found in the "Data Loading & Preprocessing" subsection within the "Algorithm

Definition" section of this report. The CNN-LSTM output consists of a binary classification- MW or not MW for each sample.

This problem is intriguing because of the impact that recognizing instances of MW can have on information retention and efficiency. If MW within a person can be detected in near real-time and this information can be provided to that person, an opportunity to correct this behavior and thus retain information more efficiently arises. Further, solving this problem without significant feature engineering is valuable because eliminating the need for feature engineering would accelerate the development process, simplify the workflow, and eliminate some reliance on domain expertise. Creating a neural network to detect instances of MW is the first step in creating a device such as this.

2.2 Algorithm Definition

2.2.1 Data Loading & Preprocessing

The first step of our algorithm is to load and preprocess data. We begin with separate CSVs containing samples for each subject and run, CSVs containing information about when each subject blinked during the sampling process, and CSVs containing information about when each subject exhibited a saccade during the sampling process. Saccades are rapid eye movements between fixation points. Typically, blinks fall between saccades, so this saccade and blink data is used to interpolate pupil dilation and eye gaze coordinates over blink periods. Whenever a blink occurs, a linear interpolation is applied to pupil dilation and eye gaze coordinates from the end of the last saccade before the blink to the start of the last saccade after the blink. During blink periods, the eye tracking system registers pupil dilation as zero and registers eye coordinates as missing values, so this interpolation is necessary to eliminate artifacts from our data and retain as much meaningful information as possible. We implement within-subject normalization for the pupil dilation after performing the interpolation by dividing each pupil dilation by the subject's mean pupil dilation to ensure the model does not overfit to the differences in pupil sizes between subjects.

Next, raw samples are extracted from the CSVs containing data for each subject and run. This is accomplished through use of a script developed in collaboration with other members of the Glass Brain Lab. The raw, unprocessed eye-tracking data is combined with metadata about the experiment (such as whether or not MW was reported to be occurring during each timepoint) to result in labeled samples for each millisecond of recorded time. In the next preprocessing step, data for each subject and run is concatenated into a single dataset and saved as a CSV file. Two new columns, "Subject" and "run_num," are introduced to enable stratified splitting during the train test split and to allow for subject-run level analysis of results. Rows with missing run numbers are dropped from this dataset as these samples relate to times prior to the start of the study (for example, while a subject was reading instructions for the experiment). Additionally, a new column, "tSample_normalized" is created to represent the normalized time of each sample, relative to the start of each run such that each run begins at time 0. This helps to make figures interpretable when performing exploratory data analysis and detailed analysis of model errors.

The process for performing a train test split is designed to ensure a balanced distribution of occurrences of mind wandering across the train and test sets while ensuring the same subjects do not appear in both the train and test sets. This is accomplished in the following way. First, the mean number

of occurrences of MW for each subject is calculated and saved as a percentage relative to the number of samples present for each subject. Next, this proportion is separated into bins to represent whether it is low, medium, or high and a stratified 80/20 train test split is performed based on the binned MW proportions. This step is crucial for maintaining balanced representation across different subjects so that the models can generalize well instead of learning the specificities of how any single subject experiences MW

After running various experiments with the full train dataset where the ratio of MW/not MW was .1, it became clear that our models were struggling due to class imbalance. For this reason, we implemented selective downsampling where periods of MW with a specified surrounding buffer of not MW are selected from the train dataset and included in a new, downsampled dataset to be used for training. It was necessary for us to be careful in how we downsample to avoid eliminating meaningful transition periods of not MW to MW or vice-versa. In all cases, the full test set was still used for evaluation. The table below (figure 2) shows results of the downsampling in terms of the new MW/Not MW ratios according to different buffers as well as the rows retained when different buffers are used. The following plots (figures 3 and 4) show the patterns of MW and not MW occurrences over time for the original train dataset and the downsampled dataset with the 6,600 ms buffer.. For each plot, data from a single subject is shown in the interest of saving space, plots for all subjects are available upon request. Results for the 10,000 ms and 6,600 ms buffer are both shown in the table as both those datasets were used in experiments; we were interested to see if reducing the imbalance to a reasonable ratio of .73 MW/ not MW was sufficient to yield acceptable results or if having an almost perfectly balanced ratio of MW/ not MW was more beneficial although it resulted in approximately one million fewer rows of data. A plot for the downsampled data with the 10,000 ms buffer is not shown as the differences between the 6,600 ms buffer and the 10,000 ms buffer are not clearly apparent visually.

Buffer	**MW/ not MW ratio**	**Rows Retained**
10,000 ms	.73	6907968
6,600 ms	1.01	5836988

Table 2. Selective Downsampling results with various buffers.

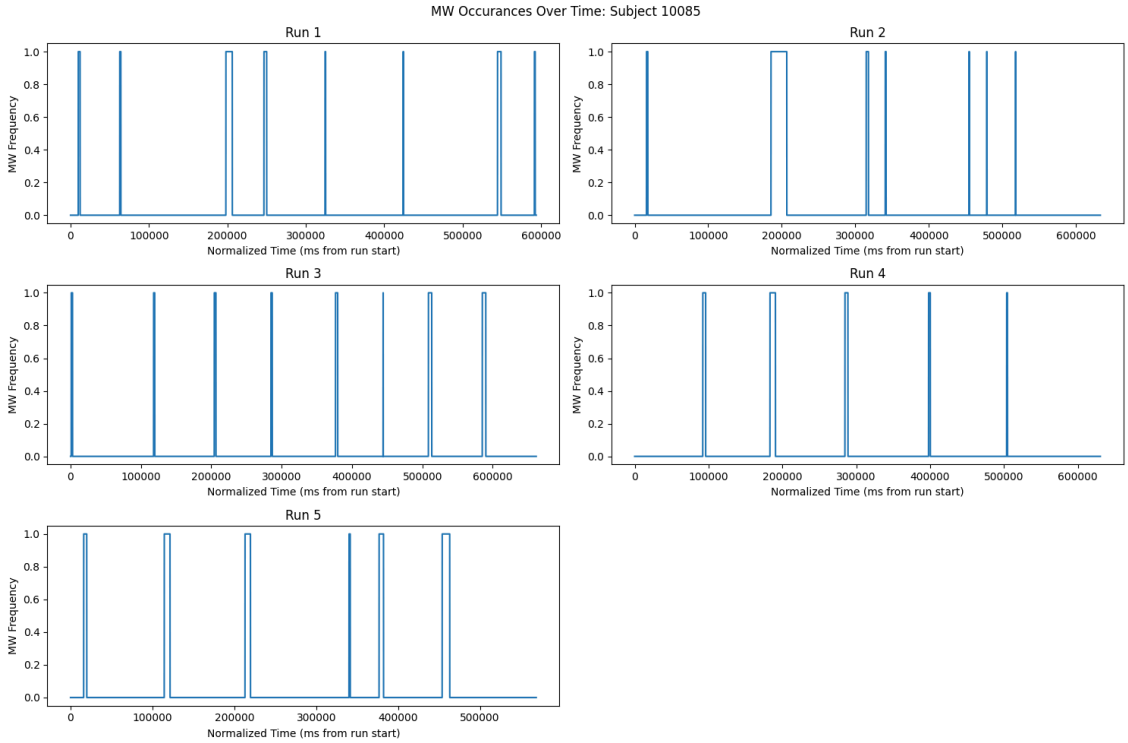

Figure 1. Occurrences of MW over time prior to selective downsampling. For this plot and the other plots of MW occurrences over time that follow, the x-axis shows normalized time (ms from run start) while the y-axis shows MW frequency such that a 1 means MW occurred for a given timestep and a 0 means that MW did not occur.

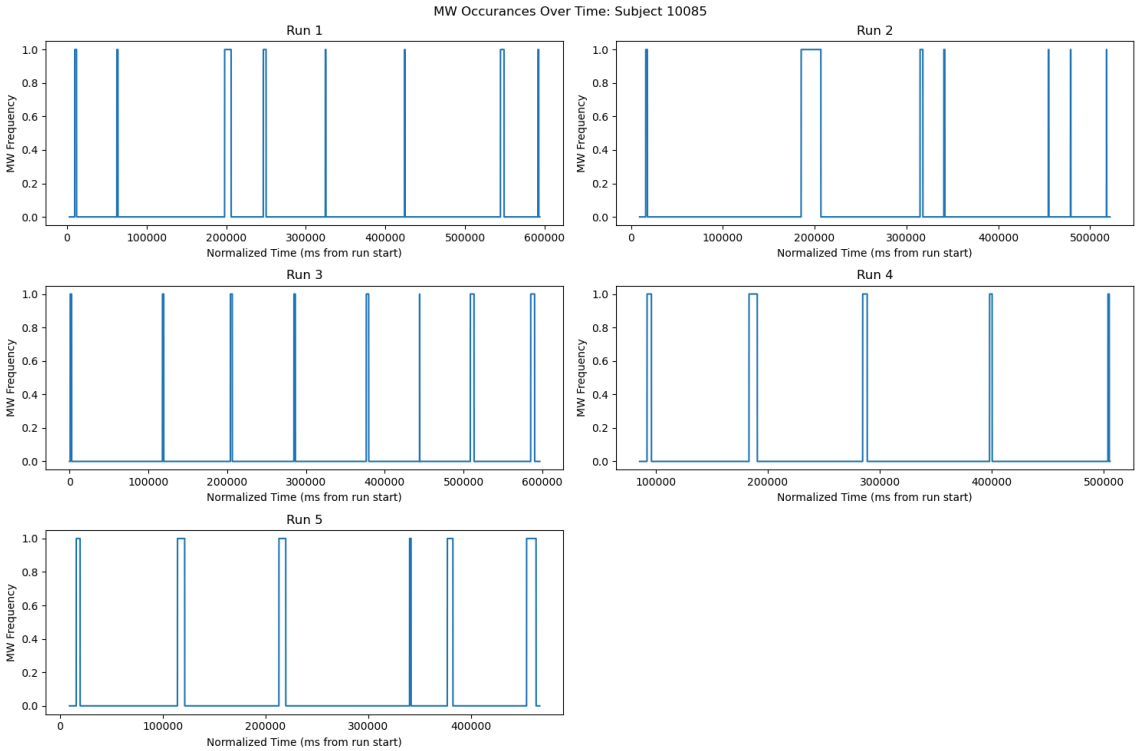

Figure 2. Occurrences of MW over time after downsampling with a 6,600 ms buffer.

2.2.2 EDA

To get a sense of patterns in the data and to ensure that importing and pre-processing were successful, various figures were produced to visualize the data. Please note that the pupil dilation and eye coordinate plots (figures 6 and 7) are only shown for a single subject, but were produced for all subjects. All plots are available upon request.

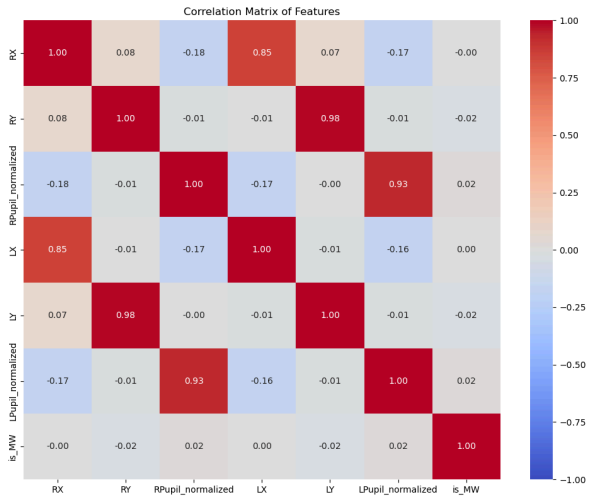

Figure 3: Heatmap of Pearson Correlation Coefficients for Features

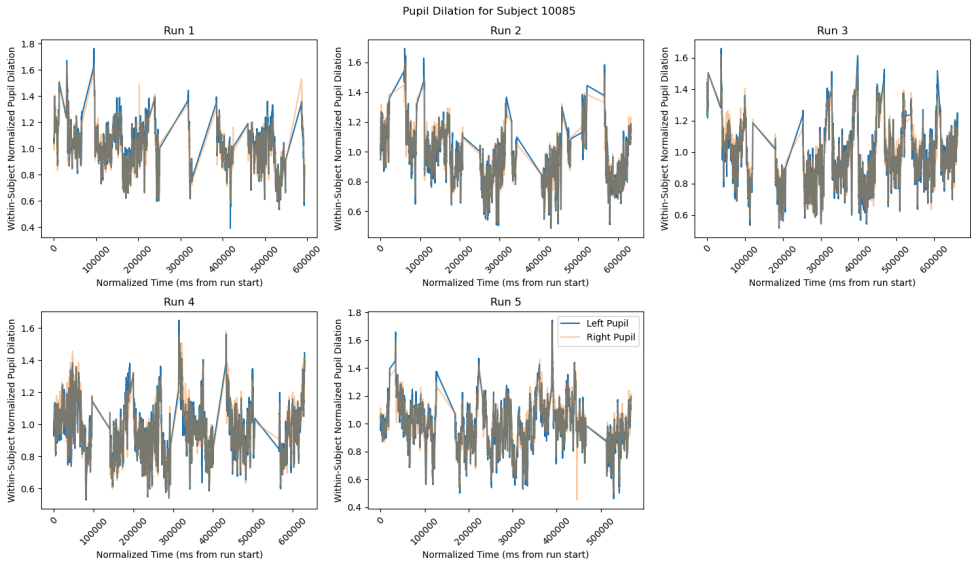

Figure 4: Within-Subject Normalized Pupil Dilation Over Time for All Runs for One Subject

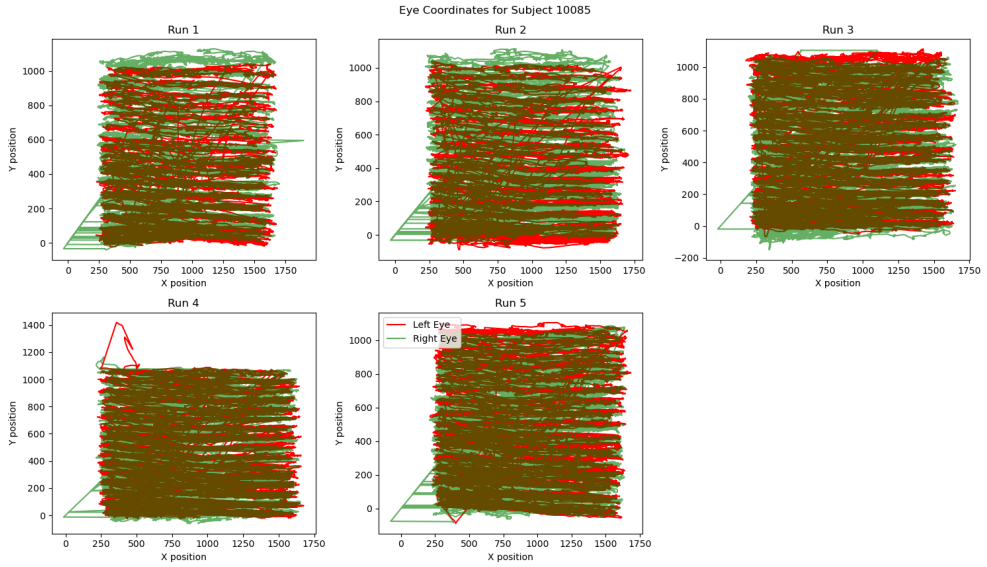

Figure 5: Left and Right Eye Coordinates For Each Run for One Subject.

2.2.3 Deep Learning System

As mentioned prior, we designed and experimented with a CNN and an LSTM individually before combining these models into a CNN-LSTM. We also extended the model to include two attention mechanisms. Attention mechanisms mimic focus on specific portions of the data most important for our model's inference performance. Such attention mechanisms are often utilized by transformers and large language models. Our CNN-LSTM consists of two 1D convolutions to act as a feature extractor that extracts spatial relationships with ReLU activation and extended to utilize a scaled dot-product attention mechanism to enhance the model with focusing on the important spatial information within the features. This feature extractor component is followed by a bidirectional LSTM with two hidden layers extended to include a Multi-Headed attention mechanism for focus on the temporal and spatial important information of our features. Lastly, there are three fully connected layers with GeLU activation to capture temporal dependencies and produce classifications. The details of our architecture are as follows, however we tweak the architecture and hyperparameters in various experiments that are described within the "Methodology" subsection of the "Experimental Evaluation" section of this report.

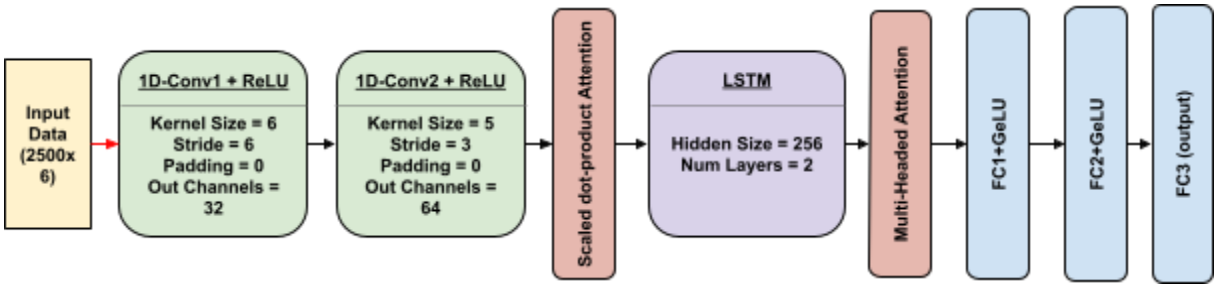

Figure 6. CNN-LSTM With Attention Architecture

We used AdamW optimizer with a learning rate of 0.00005 and a batch size of 32. BCEWithLogitsLoss was used, which combines a sigmoid layer with BCELoss. Additionally, we used automatic gradient clipping [7] with a clip percentile of 95 and a dropout mechanism with a value of 0.3, both are used to mitigate any issues with exploding or vanishing gradients that may stem from the use of LSTM on relatively long sequences. Automatic gradient clipping dynamically adjusts the clipping value using the defined clip percentile of observed gradient norms.

The input data size for the CNN-LSTM is 2500 x 6 as we want to include 2500 timesteps in the input and we have 6 features. The number of timesteps to include was informed by prior experiments with the LSTM alone. The LSTM was trained with sequence lengths ranging from 500 to 4,000, then evaluated to compare performance.

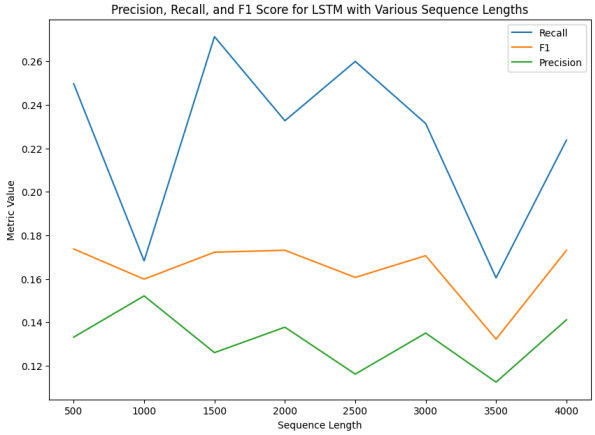

Figure 7. Precision, recall, and F1 score for LSTM with various sequence lengths ranging from 500 to 4000.

Through this experiment, it was made clear that the sequence lengths ranging from achieve similar performance with a spike in recall exhibited for sequence length 2500. Although both precision and recall are important for this task as we do not want to miss instances of MW or misclassify focused attention as MW, the spike in recall for sequence length 2500 led us to continue experimentation with this input size as the F1 score was only marginally lower than the F1 score for the sequence length 2000. Additionally, due to domain knowledge we expected that a longer sequence length would be necessary and 2500 was the longest sequence length tested before a large drop in recall was observed.

The CNN-LSTM utilizes a sliding window approach to avoid potential scenarios where sequences are grouped in such a way that important transition periods from not MW to MW or vice versa are missed. Through using this sliding window approach, we can ensure the model processes overlapping windows of data so that all relevant temporal dynamics are captured. A custom dataset class is created to produce sequences of our desired length in overlapping windows where the start index is incremented by our desired step size. A step size of 250 was chosen as it would be unusual to have multiple saccades within a 250 ms period and saccades are one of the main behaviors we are trying to capture through incorporating eye gaze coordinates as model features. Anything less would be unlikely to be valuable enough to justify the additional computational complexity required by utilizing a smaller step size.

Additionally, windows are limited so that they can only contain a single subject, run, and page so that sequences never contain a transition from one subject, run, or page to the next as that could lead the model to interpret transitions between subjects, passages, or pages as being related to MW. A standard scaler is trained on and applied to the train set, then applied to the test set during evaluation. Scaling is applied to all features to ensure they are on a similar scale. Additionally, padding is added to sequences that are under the set sequence length, which can happen when the number of rows of data are not divisible by the sequence length. To preserve temporal context, we do not shuffle within the DataLoader for this model.

The CNN-LSTM produces classifications for each timestep within the sequence input to the LSTM. Because the CNN reduces sequence length through the convolution layers, the sequence that is ultimately fed to the LSTM after passing through the CNN is reduced from a length of 2500 to 206. However, due to the sliding window approach, we are still able to generate a classification for each timestep in the original input data. Upon evaluation, the classifications for the same subject, run, and timestep across overlapping windows are aggregated by averaging the predicted probabilities for a given timestep then applying a .5 threshold to determine the final classification. This process ensures a robust prediction for each timestep while eliminating the redundancy that stems from the sliding window approach.

3. Experimental Evaluation

3.1 Methodology

The problem of mind wandering is formulated as a binary classification task, in which the goal is to classify a series of events representing eye movement behavior as either mind wandering (MW) or not mind wandering (focused attention - not MW). We employ a standard set of evaluation metrics commonly used for deep learning classification models.

- Precision
    - Also known as the Positive Predictive Value (PPV), measures the accuracy of positive predictions made by the model. High precision indicates that the model rarely mislabels a negative instance as positive. Precision is especially important in cases where false positives are costly.
- Recall
    - Also known as Sensitivity or True Positive Rate (TPR), measures the ability of the model to identify all positive instances. High recall indicates that the model correctly identifies most of the positive instances. Recall is particularly important in cases where false negatives are costly, such as in disease detection or fraud detection.
- F1-score
    - The harmonic mean of precision and recall. It provides a single metric that balances both concerns, making it useful when one wants to ensure both low false positives (high precision) and low false negatives (high recall). It is often used when one has an imbalanced dataset, as we do. The F1-score is helpful in situations where both precision and recall are important, and a balance between them is important. This is the case for our scenario, as we do not want to inaccurately notify users that they are experiencing mind

wandering when they are not as that in itself would be distracting and we do not want to miss instances of mind wandering.

- Log loss
    - A metric for binary classification tasks. Measures the distance of the predicted probability compared to the ground truth values. The lower the log loss value, the better the model is performing.
- Confusion Matrix
    - Provides a more detailed breakdown of the model's predictions, showing how often the model correctly or incorrectly predicts positive and negative classes. It is particularly useful for evaluating model performance on imbalanced datasets such as ours.
- Area Under the Curve, AUC- ROC
    - Receiver Operating Characteristic (ROC) Curve is a graphical plot that illustrates the performance of a binary classifier as its discrimination threshold is varied. The curve is generated by plotting the True Positive Rate (Recall) against the False Positive Rate (FPR) at different threshold settings.
    - Area Under the Curve (AUC) represents the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. AUC is a single value that summarizes the performance of the classifier across all threshold values.
- Matthews Correlation Coefficient (MCC)
    - Accounts for all data from each of the confusion matrix's four quadrants. This is an excellent metric for imbalanced datasets such as ours as it reflects tradeoffs between true positives, true negatives, false positives, and false negatives.

Our study hypothesized that a clear distinction between focused attention (not MW) and mind wandering (MW) during reading sessions can be made by analyzing temporal patterns in eye behavior without first performing significant feature engineering. To test this hypothesis, we performed various experiments with the CNN-LSTM, CNN, and LSTM. CNN-LSTM experiments fall into four categories: downsampling experiments, batch manipulation experiments, loss function modifications, and CNN architecture modification, we also experimented with extending our models with different attention mechanisms.

We present the results of our hypothesis testing through a series of graphs and tables that can be found beneath the heading "Results" and subheadings corresponding to the experiment categories.

3.2 Results

3.2.1 CNN-LSTM Experiments

In the following table, we present results for a series of experiments performed with the CNN-LSTM. These experiments pertain to an architecture similar to that described in the section "Deep Learning System", but without attention mechanisms and ReLU is used rather than GeLU. Results for four types of experiments are presented: downsampling, batch manipulation, loss function modifications, and architecture modifications. In all cases, the model was trained for 60 epochs with the same batch size,

optimizer, learning rate, and loss function described in the section “Deep Learning System” unless otherwise specified. We did experiment with training for more epochs, but it was consistently found that overfitting increased when the model was trained for longer. Additionally, all experiments utilize automatic gradient clipping as described in “Deep Learning System”. Descriptions of experiment types are as follows.

Downsampling experiments present results for the CNN-LSTM when trained on the data downsampled with a 10,000 ms buffer and when trained on the data downsampled with a 6,600 ms buffer, respectively.

For batch manipulation experiments, batches were manipulated so that they contained a balanced number of samples from each class (MW and not MW). Results are presented for batch manipulation on data downsampled with a 10,000 ms buffer and with a 6,600 ms buffer, respectively. Balanced batches were achieved in the following way:

1. Group sequences by class. Calculate the majority class in each sequence, then separate sequences into two lists (MW and not MW) corresponding to their majority class.
2. Create batches by combining equal numbers of sequences from each class, then shuffling sequences around within the batch. The hope is that the sequences capture enough of the transition periods that this will not skew temporal context.
3. Utilize a custom PyTorch Sampler on the DataLoader that takes the list of batches created in step 2 and yields batches during training.

For loss function modification experiments, we present results for the CNN-LSTM when trained on the full train dataset without downsampling. Two different methods of loss function modification for imbalance mitigation were used in separate experiments: focal loss [8] and PyTorch’s pos weight. Hyperparameters alpha and gamma for focal loss were determined through a grid search on the LSTM alone. The following combinations of values were tested:

Alpha	1 - p_mw			.7 * (1 - p_mw)			.3 * (1 - p_mw)		
Gamma	1	1.5	2	1	1.5	2	1	1.5	2

Table 3. Focal Loss Parameters Assessed Through Grid Search.

P_mw = the number of samples where MW occurred divided by the total number of samples. It was found that using 1-p_mw for alpha and 1.5 for gamma was optimal as it resulted in the most consistently decreasing training loss. These were the hyperparameters used to generate the results with focal loss displayed below. For pos weight, we simply divide the number of samples where MW did not occur by the number of samples where MW did occur. This penalizes the loss function more heavily for incorrect predictions in the positive class proportional to the positive class’s prevalence in the dataset.

For architecture modification experiments, a modified CNN architecture was created. While they do not utilize eye tracking data without feature engineering, various other studies have found that the best results in classifying MW vs. not MW were produced when models were trained on sequences or windows containing 8 seconds of data [9,10]. This suggests that meaningful changes related to mind wandering could occur over longer temporal windows than are captured through our existing feature extractor. For this reason, we experimented with using larger kernels and dilations to expand the area captured through the feature extractor. Specifically, the changes to the CNN architecture were as follows:

Layer	Out Channels	Kernel Size	Stride	Padding	Dilation
Conv1	32	15	2	0	1
Conv2	64	15	2	0	275

Table 4. Convolutional Layers Configuration.

The exceptionally large dilation in the second convolutional layer is motivated by the temporal dynamics of saccades. Multiple saccades are unlikely to occur within 250 ms, so the hope is that this dilation can help to capture more broad temporal patterns without skipping too much meaningful information. This CNN was then used as a feature extractor just as seen in the original CNN-LSTM, but with input of size 8,000 x 6 instead of 2,500 x 6. The modified CNN-LSTM was trained on the data downsampled with the 6,600 ms buffer for 60 epochs with a learning rate of 0.001. We chose to use the 6,600 buffer data to ensure that class imbalance wasn't negatively affecting the model. Results for all experiment types are presented in the following table.

Experiment Description	ROC-AUC	Log Loss	F1	MCC	Precision	Recall
Downsampling: 10,000 ms buffer	0.5932	0.6850	0.1450	0.0499	0.0812	0.6712
Downsampling: 6,600 ms buffer	0.5	0.7759	0.1189	-0.0012	0.0632	1
Batch Manipulation: 10,000 ms buffer	0.6069	0.6569	0.1584	0.0761	0.09215	0.5638
Batch Manipulation: 6,600 ms buffer	0.5946	0.6087	0.1654	0.0822	0.1034	0.4123
Loss Function Modification: Focal Loss	0.6164	0.6979	0.1430	0.0576	0.07814	0.8412
Loss Function Modification: Pos Weight	0.5806	0.2734	0	0.0013	0	0
Architecture Modification	0.5	0.7039	0.0853	-0.0030	0.0445	1

Table 5. Results of CNN-LSTM Experiments

Results of the downsampling experiments show that neither technique results in sufficient performance; ROC-AUC is never much higher than .5, while an ROC-AUC of .5 would represent a score that is no better than random guessing. Additionally, log loss is quite high for each and precision is low. The 10,000 ms buffer did result in slightly better performance, as evidenced through the slightly lower log loss and slightly higher F1 score and precision. The 6,600 ms buffer appears to have led the model to overfit towards MW as indicated by the high recall of 1, implying no instances of MW are missed, with the low precision, indicating many false positives.

Results of the batch manipulation experiments tell a similar story; the technique does not result in sufficient performance with either downsampled train dataset. Although ROC-AUC is slightly higher than seen without batch manipulation, it is still only slightly better than .5. Both values for log loss are relatively high and once again, precision is low. Better results were achieved with the batch manipulation on the data downsampled with the 10,000 ms buffer. In fact, these were the best results achieved across all CNN-LSTM experiments without the attention mechanism in terms of achieving higher ROC-AUC and F1 score and lower log loss. Still, this performance was subpar; the F1 score was still far from our goal.

Results of the loss function modification experiments did not reveal any significant progress. The experiment with pos weight yielded all not MW predictions, implying that the class imbalance was still negatively impacting the model. The experiment with focal loss produced the highest ROC-AUC out of all experiments, but high log loss and low F1. Due to the low value for precision and high value for recall, we can infer that the model over classified examples as MW.

Results of the experiment with the modified CNN architecture were also subpar; log loss was high, F1 was low, and the values for precision and recall suggest that the model learned to generate many false positives, reflecting its inability to form an accurate decision boundary.

For the batch manipulation experiment with downsampled data (10,000 ms buffer), training loss did not reach a plateau over 60 epochs, so we trained this model for 200 epochs to see if an improvement in performance was observed. Still, loss did not plateau, however metrics on the test set revealed that performance only decreased as shown through the following table; ROC-AUC, F1 score, and precision decreased while log loss increased. Recall increased slightly, which combined with the lower precision implies that the model learned to generate more false positives.

ROC-AUC	**Log Loss**	**F1**	**Precision**	**Recall**
0.5242	0.7930	0.1438	0.0821	0.575

Table 6. Results for the Batch Manipulation Experiment with Downsampled Data: 10,000 ms buffer after training for 200 epochs.

3.2.2 Bidirectional LSTM Experiments

We present results for experiments on the bidirectional LSTM without the CNN as a feature extractor. This LSTM simply utilizes the same architecture described in “Deep Learning System”, but without the CNN feature extractor. The LSTM utilizes a sequence length of 2500 and was trained with the same optimizer (AdamW), and learning rate (0.00005) as described for the CNN-LSTM, but with a larger batch size of 128 to facilitate faster training. Experiments with pos weight and focal loss utilize the same hyperparameters as used in the CNN-LSTM experiments with pos weight and focal loss: the number of samples where MW did not occur divided by the number of samples where MW did occur for pos weight and alpha = 1-p_mw, gamma = 1.5 for focal loss. Batch manipulation is the same as described in “3.2.2 Batch Manipulation Experiments”.

Experiment Description	**ROC-AUC**	**Log Loss**	**F1**	**MCC**	**Precision**	**Recall**
Downsampling: 10,000 ms buffer.	0.5752	0.6353	0.1605	0.0602	0.0960	0.4879
Downsampling: 6,600 ms buffer.	0.5844	0.7892	0.1480	0.0377	0.0812	0.8346

Batch Manipulation: 10,000 ms buffer.	0.5833	0.6821	0.1593	0.0599	0.0915	0.6116
Batch Manipulation: 6,600 ms buffer.	0.5770	0.6657	0.1568	0.0535	0.0931	0.4944
Loss Function Modification: Focal Loss.	0.5885	0.6315	0.1666	0.0705	0.1058	0.3908
Loss Function Modification: Pos Weight.	0.5954	0.6110	0.1888	0.1024	0.1274	0.3640

Table 7. Results of Bidirectional LSTM Experiments

Results of downsampling experiments show that neither technique results in acceptable performance; with both the data downsampled with the 10,000 ms buffer and the data downsampled with the 6,600 ms buffer, ROC-AUC is only slightly higher than .5, log loss is high, and the F1 score is low. The experiment performed on the data downsampled with the 6,600 ms buffer had much higher recall, but lower precision and higher log loss, representing that the model learned to over classify samples as MW.

Similarly, results of batch manipulation experiments do not show promise; with both the data downsampled with the 10,000 ms buffer and the data downsampled with the 6,600 ms buffer, similar metrics were achieved. The only notably different metric is recall; the model trained on the data downsampled with the 10,000 ms buffer yielded a higher score of 0.6116 compared to the recall of 0.4944 achieved with the data downsampled with the 6,600 ms buffer.

Lastly, experiments with loss function modification also yielded subpar results. For both the experiment with pos weight and focal loss, ROC-AUC was between .5 and .6 and log loss was just over .6. These experiments were slightly interesting in that using pos weight with the CNN-LSTM yielded 0 classifications of the positive class, while using pos weight with the bidirectional LSTM alone did produce some classifications of the positive class, resulting in a slightly higher F1 score and MCC than using focal loss with the LSTM alone.

3.2.3 CNN- LSTM (Unidirectional) Experiments

To enhance our experiments, this subsection examines the integration of a unidirectional LSTM model within the combined CNN-LSTM architecture. The primary aim is to assess the performance of the unidirectional LSTM approach compared to its bidirectional counterpart. For the unidirectional configuration, input data underwent distinct pre-processing: instead of preserving individual time-step labels, entire time sequences were categorized as either MW or not_MW using a sliding window approach, with a threshold of 0.50 identified as optimal for labeling.

Additionally, we employed the Gaussian Error Linear Unit (GELU) activation function instead of ReLU for these experiments, leveraging its ability to combine the benefits of ReLU and sigmoid-like functions. GELU's smooth, non-linear transitions facilitate better gradient flow during backpropagation, enhancing performance in tasks requiring sophisticated representations, such as those encountered in

transformers and language models. Finally, we utilized the same CNN component as specified in the "Deep Learning System" section.

To refine the architecture further, various hyperparameters were systematically adjusted to identify the optimal configuration, with details of the modified parameters and corresponding results presented below.

Hyper-parameters	ROC-AUC	Log Loss	F1	MMC	Precision	Recall
• 1 (1D-conv) • 5 FC	0	153.6297	0.561	0.0139	1	0.3898
• 2 (1D-conv) • 4 FC	0	149.983	0.5613	-0.0043	1	0.3902

Table 8. Results of Unidirectional CNN-LSTM Experiments

The results of the CNN-LSTM (unidirectional) model demonstrated poor performance across both configurations, with zero AUROC, a high log loss, the precision indicates the models predict only one class. This observation is further supported by analysis using the confusion matrix. While recall and F1 scores show very slight improvements as we increased the number of convolutional layers, they are still inadequate for effective classification. A reduction in the number of fully connected layers resulted in marginal gains in log loss, suggesting potential overfitting in the deeper configuration. These findings underscore significant challenges, likely due to class imbalance. A comparison with the bidirectional LSTM results confirms that the latter is better suited as a model for this task. Differences in performance could also be attributed to the different pre-processing strategy; labelling full sequences as mind wandering or not mind wandering likely eliminates some necessarily granularity within the data.

3.2.4 CNN Experiments

A Convolutional Neural Network (CNN) utilizing 1-D convolution was designed to capture spatial dependencies within sequential data, enabling the identification of patterns and relationships among proximate feature values. While CNNs do not inherently capture the temporal dynamics of time series data, they excel in constructing spatial feature correlations and often outperform RNNs and their variants in this regard. The baseline CNN architecture was designed for binary classification in a many-to-one model structure, providing a classification for each sequence rather than individual timesteps.

Just as described for the CNN-LSTM in the "Deep Learning System" section, the CNN model employed a sliding window approach with sequence length of 2,500 and a stride of 250 time steps and 1D

convolutional layers. This ensures spatial-temporal features are preserved while reflecting meaningful progression. A sliding window approach was used where the entire window of time sequences was labeled as either MW or not_MW based on a threshold of 0.50 as it was identified as optimal for labeling.

This setup reduced the number of parameters and computational cost while maintaining the spatial attributes of the data. The resulting feature maps enabled the CNN to effectively learn patterns in the dataset, distinguishing it as a suitable model for many-to-one classification of sequential time-series data. In the below table we cover the various hyperparameters that were systematically adjusted to identify the optimal configuration.

Hyper-parameters	**ROC-AUC**	**Log Loss**	**F1**	**MMC**	**Precision**	**Recall**
• 1 (1D-conv) • 5 FC • FC layer size=64	0.75	0.8526	0.0008	0.0139	0.0004	1
• 1 (1D-conv) • 5 FC • FC layer size=256	0.7501	0.6933	0.0016	0.0197	0.0008	1
• 2 (1D-conv) • 4 FC • FC layer size=512	0.7149	0.6922	0.0101	0.0448	0.0051	0.9286

Table 9. Results of CNN Experiments

The CNN model results show consistent performance across configurations, with slight improvements as the model depth increases. The first two configurations, with a single convolutional layer and multiple fully connected layers, demonstrated strong recall but struggled with precision, resulting in low F1-scores and limited ability to balance class predictions. In the third configuration, which incorporated an additional convolutional layer, there was a notable improvement in performance metrics, particularly in balancing recall and precision, along with more confident predictions. These improvements highlight the impact of increasing convolutional layers on the model's ability to capture features, though challenges with data imbalance and class discrimination remain evident.

3.3 Discussion

Overall, none of our experiments yielded acceptable results, leading us to conclude that our hypothesis is unsupported. Additional experiments that were not described in this report were performed as well, such as using different batch sizes, different learning rates, and different combinations of techniques communicated through these experiments such as using loss mitigation strategies with the data downsampled with a 10,000 ms buffer. The results presented within this report are the highlights of our

experiments that efficiently communicate our story in an organized manner. Detailed results are available upon request, but all experiments resulted in similar performance where the model failed to form an accurate decision boundary.

Across all experiments, the best results for the CNN-LSTM were achieved through the batch manipulation experiment with the data downsampled with a 10,000 ms buffer with two attention mechanisms alongside the use of mixed activations. The balanced batches likely helped the model to learn slightly more efficiently while using a 10,000 ms buffer instead of a 6,600 ms buffer reduced the data enough to somewhat mitigate the class imbalance without reducing the data as much as the 6,600 ms buffer. The performance metrics alongside confusion matrix, training loss over epochs are shown below.

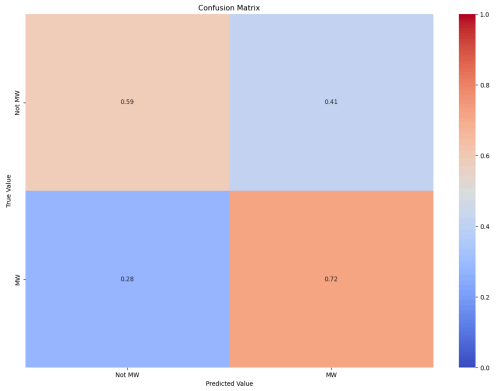

Figure 8. Confusion Matrix for CNN-LSTM with Attention

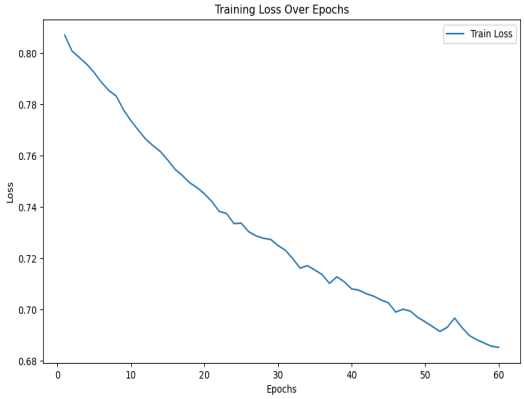

Figure 9. Training Loss over Epochs for CNN-LSTM with Attention

ROC-AUC	**Log Loss**	**F1**	**Precision**	**Recall**
0.7415	0.7224	0.2114	0.123	0.721

Table 10. Results for the CNN-LSTM with Attention

Related Work

The following papers present the highlight of the related research that we found most relevant. An interesting point to raise is the mutual agreement that eye behavior is a valuable resource of study for the phenomena of mind wondering:

- *The Eyes Have It: Gaze-Based Detection of Mind Wandering during Learning with an Intelligent Tutoring System* [3]. This study addresses the phenomena of mind wandering. The novelty of this study stems from addressing mind wandering in interactive learning tasks, dubbed as "intelligent tutoring systems". The researchers relied on self-reporting for the ground truth labels for mind wandering. They presented the study's best result using an evolutionary approach "NeuroEvolution of Augmenting Topologies algorithm" for the neural network classifier's topologies. This approach has proven in the evaluation that neural networks do outperform standard classifiers.
- *Mind Wandering in a Multimodal Reading Setting: Behavior Analysis & Automatic Detection Using Eye-Tracking and an EDA Sensor* [2]. In this study, researchers were able to achieve a good F-1 score of 0.78 using electrodermal features in random forest classifiers. When limiting the features to eye movement, the F-1 score increased to 0.80. The highest F1 Score, 0.83, was observed when including both features. The results of the random forest classifier can be attributed to the use of EDA and progress in the field over the past years. They found that a strong relationship exists between mind wandering episodes and eye movement patterns, in particular fixations, blink rate, pupil diameter, eye vergence and saccades. The study also showed that internally directed cognition is associated with longer yet less frequent fixations and a higher variability in pupil size diameter and eye vergence, and the angle of eye vergence. The study also found that an increase in blinking rate is associated with mind wandering periods.
- *Automatic Gaze-based User-independent Detection of Mind Wandering During Computerized Reading* [4]. This study shifted focus to shorter time frames within the tasks instead of labeling the entire event as focused or mind wandering. The aim was to build a supervised machine learning model from a short window of 4-10 seconds. The study included the experimentation on various models such as bagging, bayes net, naive bayes, logistic regression, simple logistic regression, SMO and SPegasos SVM models.
- *Window to the Wandering Mind: Pupillometry of Spontaneous Thought While Reading* [5]. This analysis included 13 participants whose PD data passed the quality control measures and who also had reported both on- and off-task episodes. A one-way repeated measures ANOVA revealed a significant difference in PD, $F(1, 12) = 6.45$, $p = .03$; overall, participants had higher PD when off-task than when on-task. They found that higher PD is associated with mindless (vs. mindful) reading when it occurs in a spontaneous manner.
- *The Eye–Mind Wandering Link: Identifying Gaze Indices of Mind Wandering Across Tasks* [1]. This study found that mind wandering was associated with fewer, more dispersed fixations as well as shorter saccades in the reading task. Fixation durations associated with mind wandering were also longer. In this research the scientists wanted to verify and correlate the different attributes of eye behavior with mind wandering. They focused on a number of features for this study such as duration of fixation and blinking rate. The researchers were able to confirm that

there is a clear connection between the eye/gaze behavior and mind wandering and highlighted the following: longer fixations are associated with mind wandering, fewer eye movement do indicate mind wandering, and higher blinking rate indicates mind wandering.

- *Eye Movements During Mindless Reading* [6]. In this study, researchers found that fixation-duration measures were longer during self-caught and probe-caught mindless reading than during normal reading, and that these differences were evident as early as 60s to 120s prior to when the mind wandering was caught. However, this pattern was more pronounced for self-caught episodes and participants were less likely to make first-pass fixations, word fixations, or interword regressions (all of which are indicative of normal text processing) in the 2.5-s interval immediately preceding self-caught mind wandering than in the 2.5-s interval preceding either probe-caught mind wandering or normal reading. This suggests disengagement from the reading process just before self-caught instances of mind wandering.

5. Code and Dataset

As our dataset holds biometric human subject data, it comes under the strict IRB protocol privacy standards. In compliance with the protocol's policies and procedures, it is not possible to share the data publicly.

The project code files can be found on our GitHub repository. The repository holds multiple files, each for its own distinct purpose. The files used for this model and their functions are listed below (please note that the GitHub repository contains various other files that were used in the development process, but are less relevant to readers in the context of this report):

1. Load_Preprocess_EDA.py: A script for loading and preprocessing raw data as well as performing exploratory data analysis on data after preprocessing.
2. Reading_analysis.py: The original code from which the data interpolation functions were derived from.
3. parse_eyeLink_asc.py: Combines raw, unprocessed eye-tracking data with metadata about the experiment (such as whether or not MW was reported to be occurring during each timepoint) to result in labeled samples for each millisecond of recorded time. This is referenced in the Load_Preprocess_EDA script.
4. Balance_data.py: The script used to perform selective downsampling as described in this report.
5. prep_data_LSTM.py: The script used to perform the train test split as described in this report.
6. train_CNN-LSTM.py: The training script for the CNN-LSTM as described in the section "Deep Learning System".
7. test_CNNLSTM.py: The testing script for the CNN-LSTM as described in the section "Deep Learning System".
8. train_CNN-LSTM_experiment.py: The training script for the CNN-LSTM as described in the section "CNN Architecture Modification".
9. test_CNN-LSTM_experiment.py: The testing script for the CNN-LSTM as described in the section "CNN Architecture Modification".
10. train_LSTM_new_windows.py: The training script for the LSTM without the CNN feature extractor.

11. test_LSTM_with_analysis_eff.py: The testing script for the LSTM without the CNN feature extractor. Contains functionality to aggregate classifications for timesteps corresponding to the same subject, run, and page, repeated in multiple overlapping windows, plot errors over time for each test subject, and plot a heatmap of the confusion matrix.
12. CNN.py: the program for training and evaluating the CNN model, sliding window mechanism for sequence level labeling.
13. CNN_ULSTM.py: the program for training and evaluating the unidirectional CNN-LSTM model, sliding window mechanism for sequence level labeling.
14. train_CNN_LSTM_Att.py: the program for training the directional CNN-LSTM model extended with the use of attention mechanisms on both the CNN and LSTM models.
15. test_CNN_LSTM_Att.py: the program for evaluating the directional CNN-LSTM model extended with the use of attention mechanisms on both the CNN and LSTM models.

6. Conclusion

In conclusion, our results do not support our hypothesis under the conditions tested; we believe that it is not possible for the CNN-LSTM to make a clear distinction between not MW and MW through use of the eye tracking data we currently have without feature engineering. Although proving this rigorously would require a timeframe that is not currently feasible, we feel confident in this conclusion due to the wide variety of experiments performed and consistently subpar results. It is possible that obtaining more data from a wider variety of subjects could help. Perhaps seeing more data from more subjects could allow the model to form an accurate decision boundary despite differences between subjects. Incorporating additional features may also be beneficial. For example, EEG data could be used instead of or in addition to eye tracking data. There is certainly promise to using EEG data for this purpose as Hosseini & Guo were able to achieve “91.78% accuracy, 92.84% sensitivity, and 90.73% specificity” in MW classification through the use of a CNN on EEG data with minimal feature engineering in *Deep Convolutional Neural Network for Automated Detection of Mind Wandering Using EEG Signals* [9].

Bibliography

[1] Faber, M., Krasich, K., Bixler, R. E., Brockmole, J. R., & D'Mello, S. K. (2020). The eye–mind wandering link: Identifying gaze indices of mind wandering across tasks. Journal of Experimental Psychology: Human Perception and Performance, 46(10), 1201–1221. https://doi.org/10.1037/xhp0000743

[2] Brishtel, I., Khan, A. A., Schmidt, T., Dingler, T., Ishimaru, S., & Dengel, A. (2020). Mind Wandering in a Multimodal Reading Setting: Behavior Analysis & Automatic Detection Using Eye-Tracking and an EDA Sensor. *Sensors*, *20*(9), 2546. https://doi.org/10.3390/s20092546

[3] Hutt, S., Mills, C., White, S., Donnelly, P. J., & D'Mello, S. K. (n.d.). The Eyes Have It: Gaze-Based Detection of Mind Wandering during Learning with an Intelligent Tutoring System. https://eric.ed.gov/?id=ED592729

[4] Bixler, R., & D'Mello, S. (2015). Automatic gaze-based user-independent detection of mind wandering during computerized reading. User Modeling and User-Adapted Interaction, 26(1), 33–68. https://doi.org/10.1007/s11257-015-9167-1

[5] Franklin, M. S., Broadway, J. M., Mrazek, M. D., Smallwood, J., & Schooler, J. W. (2013). Window to the Wandering Mind: Pupillometry of Spontaneous Thought While Reading. Quarterly Journal of Experimental Psychology, 66(12), 2289-2294. https://doi.org/10.1080/17470218.2013.858170

[6] Reichle, E. D., Reineberg, A. E., & Schooler, J. W. (2010). Eye Movements During Mindless Reading. Psychological Science, 21(9), 1300-1310. https://doi.org/10.1177/0956797610378686

[7] Seetharaman, P., Wichern, G., Pardo, B., & Le Roux, J. (2020, September). Autoclip: Adaptive gradient clipping for source separation networks. In 2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP) (pp. 1-6). IEEE.

[8] Lin, T. (2017). Focal Loss for Dense Object Detection. arXiv preprint arXiv:1708.02002.

[9] Hosseini, S., & Guo, X. (2019, September). Deep convolutional neural network for automated detection of mind wandering using EEG signals. In Proceedings of the 10th ACM international conference on bioinformatics, computational biology and health informatics (pp. 314-319).

[10] Sidney D'Mello, Kristopher Kopp, Robert Earl Bixler, and Nigel Bosch. 2016. Attending to Attention: Detecting and Combating Mind Wandering during Computerized Reading. In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16). Association for Computing Machinery, New York, NY, USA, 1661–1669. https://doi.org/10.1145/2851581.2892329