Overview of the Facebook Oculus AI Model Motion Level Prediction Process:

My code implements a motion level prediction system using an ONNX (Open Neural Network Exchange) model and a series of features extracted from eye-tracking data. The system predicts three potential motion levels: Low, Moderate (Mod), and High. The predictions are based on a real-time analysis of incoming data, and the thresholds for the transitions between these levels are dynamically adjusted using a **CyberSicknessMemory model using Self Attention and reinforcement learning**. The self-attention layer to allow the model to focus on key parts of the sequence. We also add **BERT LLM Transformer Feature Extraction** Integrate BERT embeddings to capture contextual information from the input sequence. This step creates new input features from the raw data, enabling better motion prediction.

Workflow for Training the Model

1. Data Preparation with BERT LLM Transformer

- **Load and Clean Data**: The input dataset (`mydf_cleaned`) contains features related to eye movement, head orientation, and other metrics. This data is used to predict `time_to_next_transition`—the time until the next transition between motion levels (e.g., from Low to Mod).
- **Feature Selection**: A set of selected features (`selected_features`) is extracted from the data.
- **Normalize Features**: The features are normalized using `StandardScaler` to standardize them, ensuring that the model training is not biased due to different feature scales.
- **Handle Imbalance with SMOTE**: To address data imbalance, **SMOTE** (Synthetic Minority Oversampling Technique) is applied to oversample the minority class, ensuring a balanced dataset.
- **BERT Feature Extraction** Integrate BERT embeddings to capture contextual information from your input sequence. This step creates new input features from the raw data, enabling better motion prediction.
- **Use Pre-trained BERT**: Extract embeddings using BERT for features like `eye_openess` and `gaze_direction` to encode contextual information about the motion:

2. Create Sequences for LSTM Input

- **Time-Series Data**: Since the LSTM model expects sequential data, the input data is transformed into sequences of a specific length (`sequence_length = 20`). This means that for each sample, a sequence of 20 time steps is used as input to predict the target (time to next transition).

.

3. LSTM Self Attention Model Architecture

- **LSTM with Residuals and Layer Normalization**: The `TransitionTimeLSTM` class defines a custom LSTM architecture with:
    - **Bidirectional LSTM**: Processes the input sequence in both forward and reverse directions, capturing dependencies from both ends of the sequence.
    - **Layer Normalization**: Improves model stability by normalizing the output of the LSTM layer.
    - **Fully Connected Layer**: Maps the LSTM output to the final prediction, which is the time to transition.
- **Dropout**: Used in the LSTM layer to prevent overfitting.
- **Self-Attention Mechanism**
- Add a self-attention layer to allow the model to focus on key parts of the sequence.

4. Bayesian Optimization for Hyperparameter Tuning

- **Optimize Model Hyperparameters**: Before training the model, **Bayesian Optimization** is used to find the best hyperparameters:
    - **Hidden Size**: The number of hidden units in the LSTM.
    - **Dropout Rate**: The dropout percentage to prevent overfitting.
    - **Learning Rate**: The learning rate for the optimizer.
- **Objective**: The Bayesian optimizer aims to minimize the loss during training by searching through different hyperparameter configurations.

5. Few-Shot Learning Training

- **Few-Shot Training**: The model is trained using a few-shot learning approach, which focuses on training the model with limited data to generalize well to new situations. The training process occurs over a number of **episodes** rather than epochs. In each episode, the model updates based on a small subset of data, improving its ability to adapt to new data patterns.

6. Training the LSTM Model

- **Training Process**:
    - The model is trained using the Adam optimizer and **Huber Loss** function, which is less sensitive to outliers than MSE (Mean Squared Error).
    - **Epochs**: The model is trained over multiple epochs (up to 50), during which the weights are updated based on the loss function. At each epoch, the model learns to improve its predictions.
    - **Scheduler**: A learning rate scheduler (`ReduceLROnPlateau`) is used to reduce the learning rate when the model's performance (RMSE) plateaus, which helps in fine-tuning the learning process.
- **Evaluation**:
    - At the end of each epoch, the model is evaluated on the test set using RMSE (Root Mean Squared Error). RMSE measures how well the model predicts the time to transition compared to the actual time.
    - **Best Model**: The model with the best RMSE is saved for later use.

7. Model Saving and Export to ONNX

- **Save the Model**: The model is saved in both PyTorch (`.pth`) and ONNX formats (`.onnx`). ONNX export allows the model to be used in other frameworks or deployed efficiently in production environments.
- **Dynamic Axes in ONNX**: During ONNX export, dynamic axes are defined for batch size and sequence length to allow the model to handle variable-length input sequences during inference.
    - ent.

This process ensures that the LSTM model is well-optimized for predicting motion level transitions, can handle sequential data, and generalizes well across different data patterns.

Here are the steps involved and an explanation of how it works:

1. Feature Collection and Preprocessing

- **Input Data**: The input data is loaded from a CSV file, which contains several features related to eye movements, gaze direction, and head orientation (e.g., `Right_Eye_Openness`, `GazeDirectionLclSpc_X`, `SPACE_STORM_saccade_num.b`).
- **Feature Selection**: From each row in the CSV, a dictionary of 28 selected features is constructed. This feature set is critical for feeding into the ONNX model to make predictions.

2. Loading the ONNX Model

- **ONNX Model**: The function `load_onnx_model` loads the trained ONNX model for motion level prediction. This model is designed to accept a sequence of input features and return a prediction for the probability of the motion being in the **Low**, **Mod**, or **High** state.

3. Running Predictions with the ONNX Model

- **Feature Conversion**: The dictionary of features from each row is converted into a NumPy array and reshaped to fit the input dimensions expected by the ONNX model.
- **Model Inference**: The reshaped array is passed to the ONNX model, and the model produces a prediction vector. This vector contains the probabilities for each motion level (Low, Mod, and High). These probabilities are computed using the softmax function, which normalizes the output into probabilities.

4. Tracking Probabilities with CyberSicknessMemory

- **Tracking Probabilities**: The **CyberSicknessMemory** class is responsible for tracking the probabilities of the Low, Mod, and High motion levels across time. The class maintains rolling averages and overall means of these probabilities.
- **Adaptive Threshold Adjustment**: Based on percentage differences between rolling averages and overall means, thresholds for classifying motion levels are adjusted dynamically. This ensures that the system can adapt to changing conditions in the input data.

    - **Thresholds**: These thresholds dictate when the system decides that the motion has shifted from one level to another (e.g., from Low to Mod or Mod to High). If the rolling average deviates significantly from the overall mean, the thresholds are adjusted accordingly.

5. Motion Level Classification

- **Classification Logic**: The system calculates percentage differences between rolling averages (over a sliding window) and overall means for each motion level. These differences help determine the likelihood of transitioning from one motion level to another.
- **Threshold Comparison**: If the difference in the High probability exceeds the threshold, the system predicts a transition to the High level. Similarly, it compares the Low and Mod probabilities to their respective thresholds to decide if the motion level should be classified as Low or Mod.
- **Final Decision**: After threshold comparison, the system selects the motion level with the highest probability and stores this prediction.

6. Smoothing Window

- **Smoothing Transitions**: The system uses a smoothing window to avoid abrupt changes in predicted motion levels. If the motion level is predicted to remain the same within a small number of rows (less than the smoothing window), it retains the previous motion level prediction. This helps prevent noisy fluctuations in the predictions.

7. CSV Output and Visualization

- **Saving Predictions**: Once the system has processed all rows in the CSV, it saves the actual and predicted motion levels into a new CSV file (`actual_vs_predicted.csv`).
- **Visualization**: The code generates a plot that compares the actual motion levels from the data with the predicted levels from the model. This visualization helps evaluate how closely the model predictions match the ground truth data.

8. Main Function

- **Entry Point**: The `main` function ties everything together by loading the CSV file, running the model row by row, and generating predictions. It also saves the results and plots the actual vs. predicted motion levels.

This process ensures that the model can adaptively predict and track motion levels over time, adjusting thresholds and smoothing transitions to maintain stable predictions.

Training the Time-to-Transition Model

The **time-to-transition model** predicts the time in seconds for transitioning between motion levels (Low to Mod or Mod to High) using a custom **LSTM model** with **Bayesian Optimization** for hyperparameter tuning.

Steps in Training the Model:

1. **Data Preparation**:
    - Data is cleaned, features are extracted, and **SMOTE** is used to balance the dataset.
    - Features are normalized using `StandardScaler`, and sequences are generated for LSTM input (e.g., sequences of 20 time steps).
2. **LSTM Model Definition**:
    - The `TransitionTimeLSTM` class defines a bidirectional LSTM with layer normalization and a fully connected layer for predicting the time to the next transition.
    - **Dropout** is applied to prevent overfitting.
3. **Bayesian Optimization for Hyperparameter Tuning**:
    - **Hidden size**, **dropout rate**, and **learning rate** are optimized using **Bayesian Optimization**.
    - The goal is to minimize training loss by searching for the best hyperparameters.
4. **Few-Shot Learning Training**:
    - The model is trained using a **few-shot learning** approach, with updates occurring over small episodes of data. This helps the model generalize well with limited data.
5. **Training Process**:
    - The model is trained over multiple epochs (up to 50), with **Huber Loss** used to reduce sensitivity to outliers.
    - A learning rate scheduler is used to reduce the learning rate when performance plateaus, improving the model's fine-tuning.
6. **Model Evaluation and Saving**:
    - The model is evaluated using **RMSE** (Root Mean Squared Error), and the best model is saved in both **PyTorch (.pth)** and **ONNX (.onnx)** formats.
    - ONNX allows the model to be efficiently deployed in production.

By following these steps, the **Time to Transition Model** dynamically predicts the time in seconds until a user transitions between motion levels (Low to Mod, Mod to High), ensuring real-time predictions and adaptability to changing conditions.

1. Motion Sickness Level prediction model Low Mod High

    Model Wrapper Run:

    https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/al_run_level_oct4_6.py

    Model File:

    https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/best_cyber_sickness_lstm_model.onnx

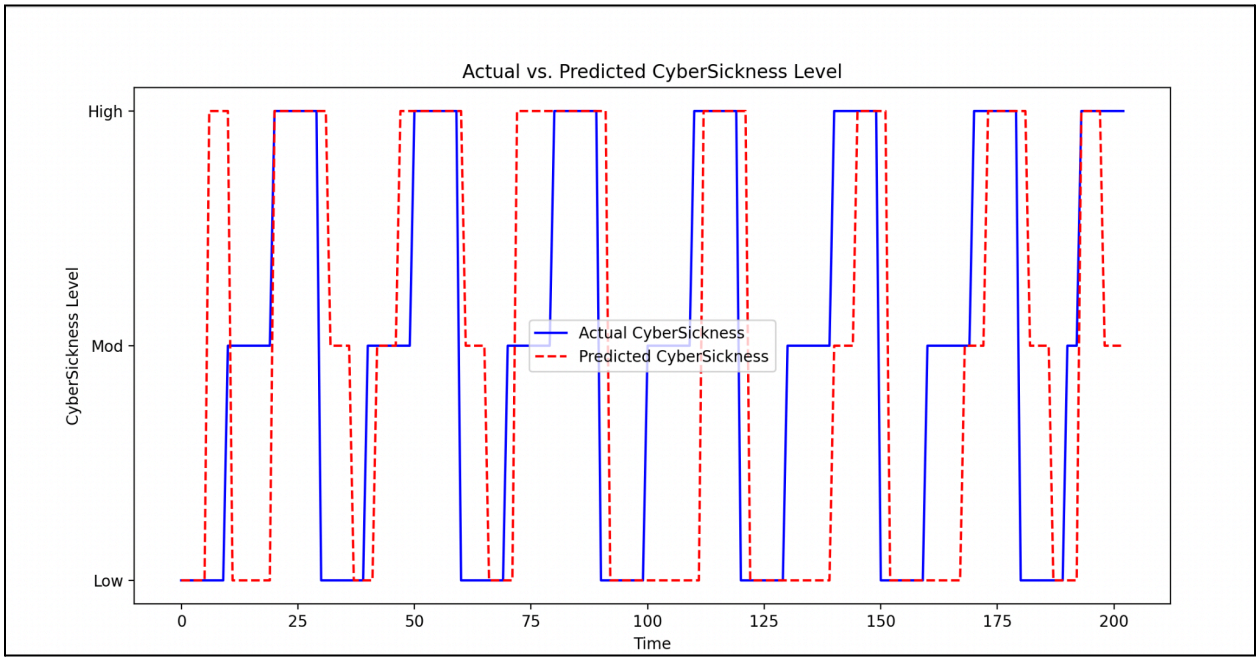
    Test Data In:

    https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/steady_state_low_mod_high.csv

Time Transition Low to Mod Predicted vs Actual Out:

https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted.csv

Plot:



2. Time to Transition Low to Mod Model predicts the NEXT time in seconds from Now that user with transition from Low to Mod

   Model Wrapper Run:

   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/al_run_tt_lm_oct6_3.py

   Model File:

   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/best_transition_time_lstm_model_low_mod.onnx

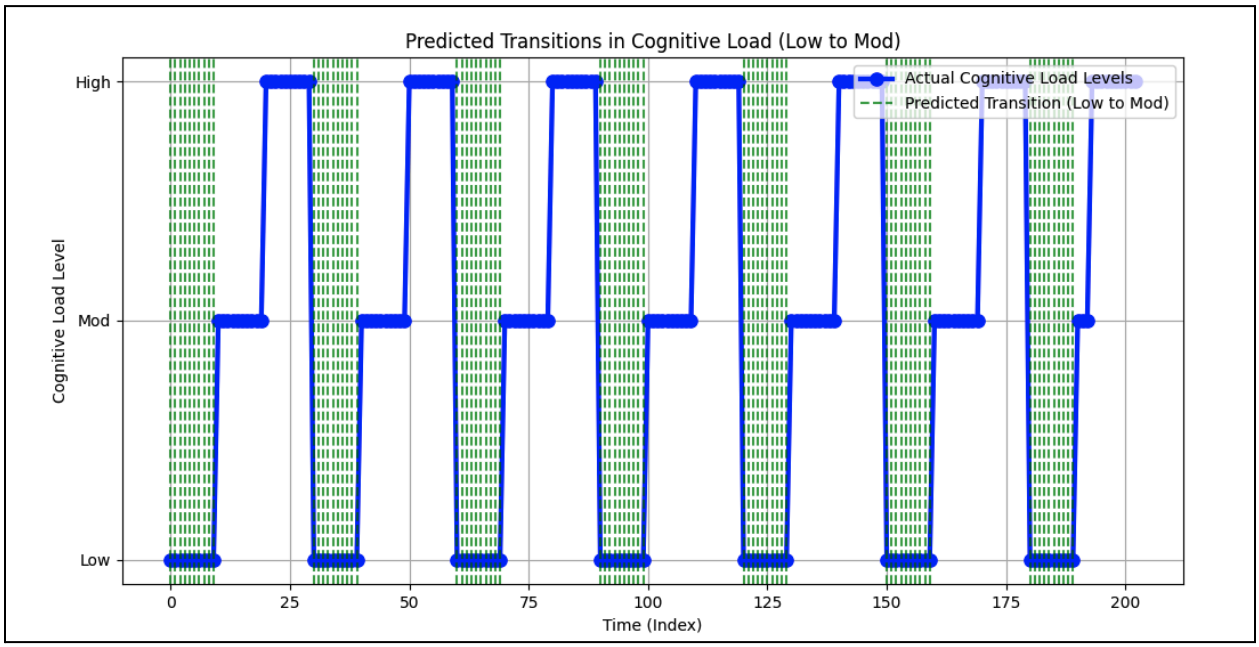
   Test Data In:

   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/steady_state_low_mod_high.csv

   Time Transition Low to Mod Predicted vs Actual Out:

https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_transition_time_low_mod.csv

Plot:



3. Time to Transition Mod to High Model predicts the NEXT time in seconds from Now that user with transition from Mod to High

   Model Wrapper Run:

   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/al_run_tt_mh_oct6_7.py

   Model File:

   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/best_transition_time_lstm_model_mod_high.onnx

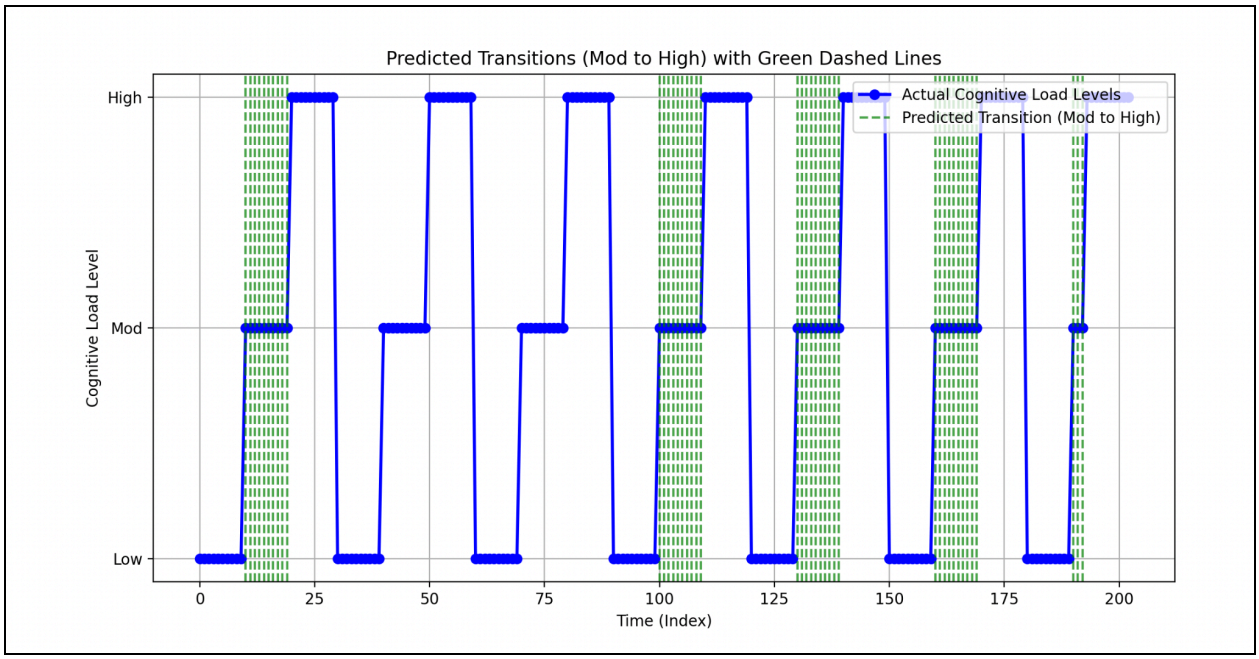
   Test Data In:

   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/steady_state_low_mod_high.csv

   Time Transition Mod to High Predicted vs Actual Out:

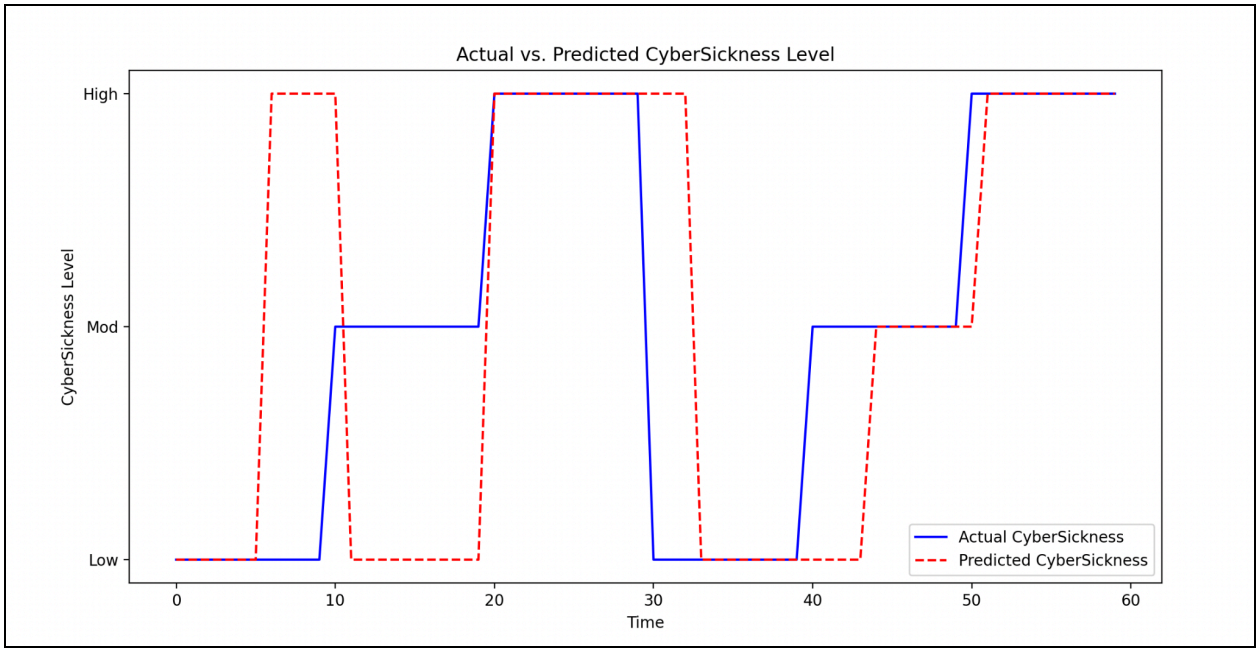
   https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_transition_time_mod_high.csv

   Plot:



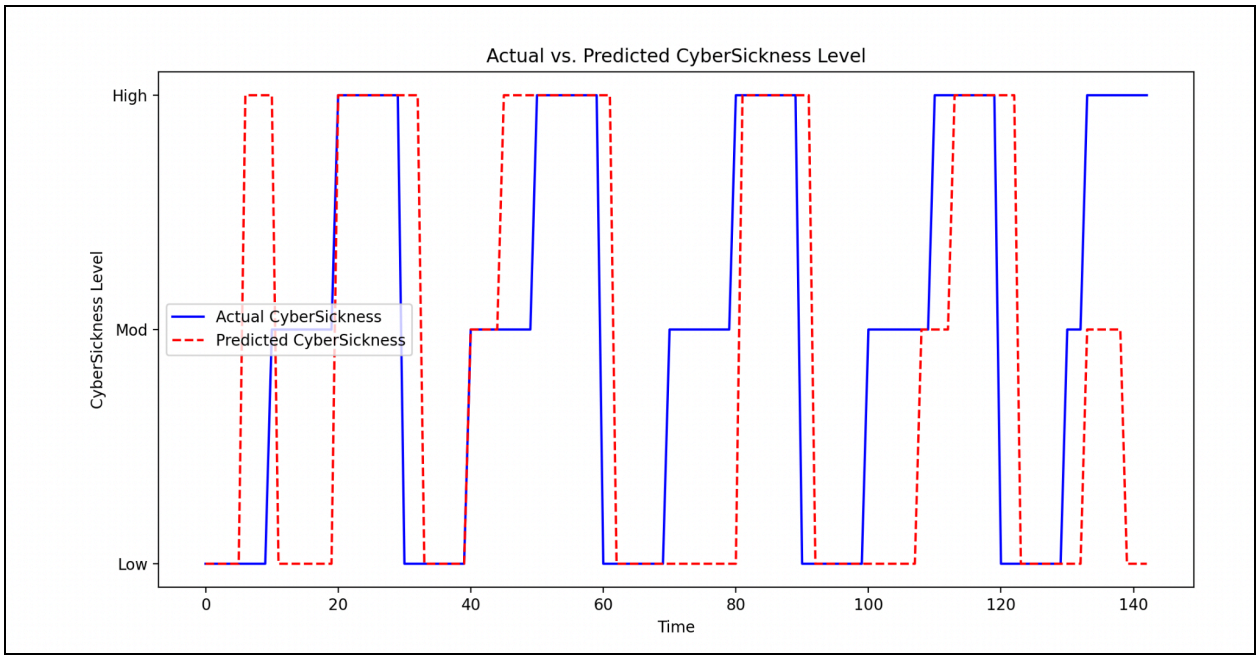
TEST 1

1. DATA IN
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/low_45_mod_45_high_45.csv
2. Prediction Out:
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_low_45_mod_45_high_45.csv



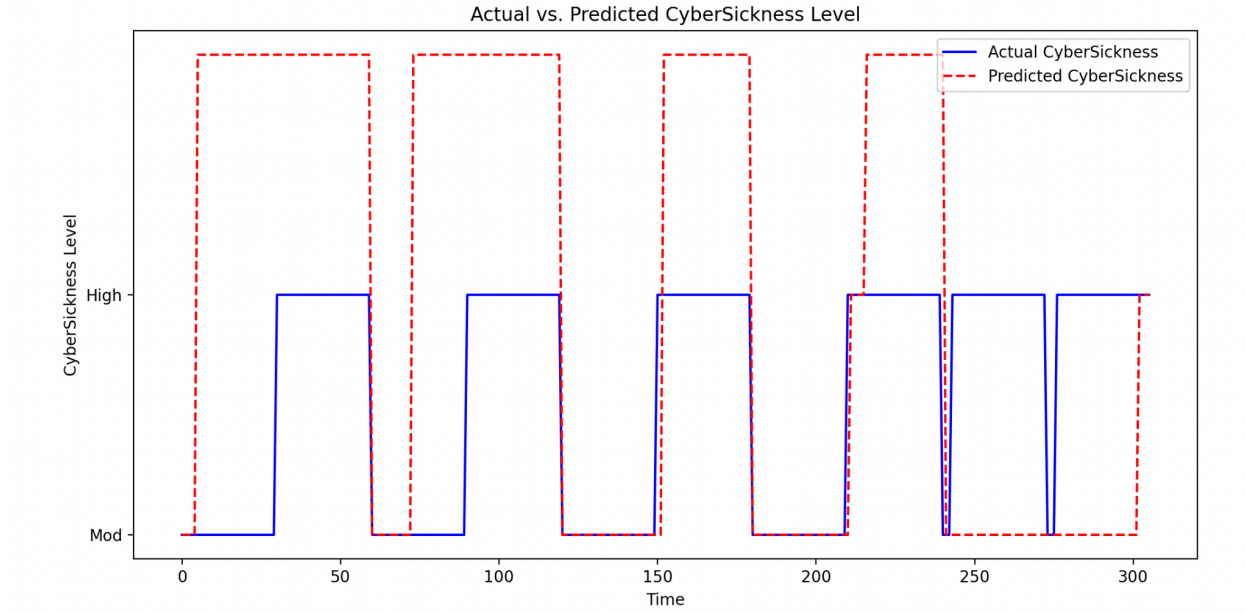
TEST 2

3. DATA IN
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/make_low_mod_high_15_sec.csv
4. Prediction Out:
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_make_low_mod_high_15_sec.csv



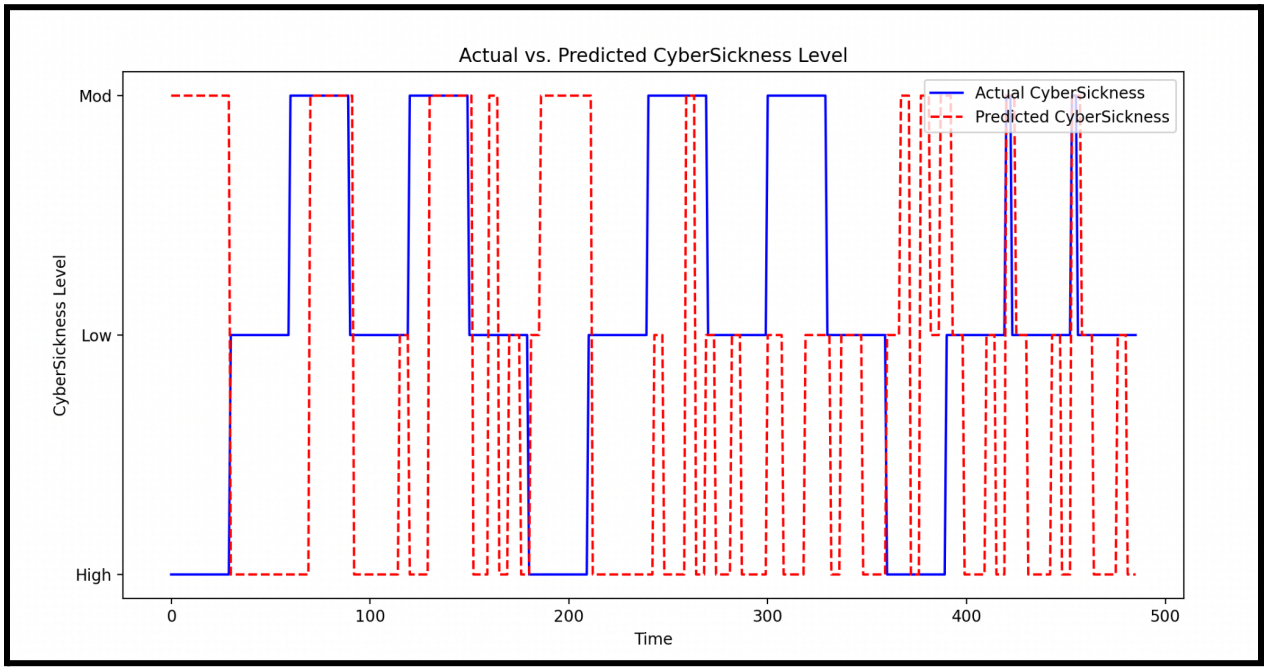
Test 3

5. DATA IN
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/make_low_mod_high_15_sec.csv
6. Prediction Out:
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_make_low_mod_high_15_sec.csv



Test 4

7. DATA IN
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/make_low_mod_high_15_sec.csv
8. Prediction Out:
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_make_low_mod_high_15_sec.csv



Test 5

9. DATA IN
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/make_low_mod_high_15_sec.csv
10. Prediction Out:
    a. https://github.com/RightEyeLLC/cyber/blob/main/AL_Release_1/data/actual_vs_predicted_make_low_mod_high_15_sec.csv