Smart Alarm Clock

Alma Alftayeh - 2267172

May 2025

Abstract

This project introduces a smart alarm clock system powered by Deep Learning technologies, designed to wake users at an optimal time based on their sleep efficiency rather than an exact time. The system integrates data from the Apple Watch, which records sleep patterns including sleep duration, REM, deep, and light sleep, as well as other factors that may affect sleep quality. Using AI techniques, specifically an ensemble of neural networks and LSTM models, the system analyzes this data to predict the best time for the user to wake up from a desired wake-up window decided by the user, enhancing sleep quality and daily productivity.

Contents

1 Related Work

Most wearable devices utilize AI technologies to analyze sleep patterns, but they do not automatically detect the optimal wake-up time or set an alarm. On the other hand, SleepCycle analyzes the user's sleep patterns and habits and sets an alarm automatically at the optimal wake-up time, but it collects data through the user's phone rather than wearable devices.

While individual components such as wearable data analysis, deep learning for sleep prediction, and ensemble modeling have been explored, no existing system fully integrates these elements into a single smart alarm clock solution. Specifically, there is a gap in systems that combine wearable data, use ensemble deep learning models (including LSTM networks), and optimize wake-up times based on comprehensive sleep analysis.

Integrating wearable data with advanced deep learning techniques, especially ensemble models incorporating LSTM networks, holds significant potential for developing smart alarm clocks that can optimize wake-up times based on individual sleep patterns. However, current research has yet to fully realize this integration, presenting a valuable opportunity for innovation in this area.

2 Methods

2.1 System Overview

The smart alarm clock system integrates several components to analyze and optimize the user's wake-up time. It collects data from the Apple Watch and cleans it, processes it through a deep learning model, and uses the model's output to recommend the best time to wake up the user from his desired wake-up time window. The user will be able to set a desired wake-up time window (e.g. 07:00 to 09:00), the model will calculate sleep efficiency for each 5 minute interval of this window, and determine time with highest sleep efficiency as the optimal time to wake the user up. This system optimizes wake-up times based on real-time sleep efficiency, ensuring that users wake up during their light sleep phase.

For this project, the data was collected from a Kaggle dataset, but during future developments, the Apple Watch API could be integrated to directly fetch data from the wearables rather than training on the Kaggle dataset only.

2.2 Data Preprocessing

- Data is loaded from csv file into Pandas dataframe.
- Categorical data is encoded for better model analysis. Both Gender and Smoking Status are encoded into 0's and 1's and converted to numerical features instead.
- Missing data in numerical columns are filled using the mean of the column.

- New features such as Wakeup hour and Wakeup minute are added and simulated using random values for training
- The data is split into training and testing dataset 80/20, and finally the features are scaled.

2.3 Sleep Efficiency Prediction using Deep Learning Model

Since the target prediction value is sleep efficiency which is a numerical value, the task of the project is a regression task. All activation functions and performance metrics are set according to that. The architecture of the deep learning model goes as the following:

- **Fully Connected Neural Network**: A sequential neural network is created:

  1. Input Layer that takes in input features.
  2. First Dense Layer with 64 neurons and ReLU activation function.
  3. Dropout Layer (0.2) for regularization
  4. Second Dense Layer with 32 neurons and ReLU activation function.
  5. Dropout Layer (0.2) for regularization
  6. Third Dense Layer with 16 neurons and ReLU activation function.
  7. Output Layer with a single neuron and Sigmoid activation function to predict sleep efficiency.

  The Adam optimizer is then used, MSE and MAE are calculated, and the model is complied

- **LSTM Model**: A sequential LSTM model is created:

  1. Input Layer that takes in input features.
  2. Reshape Layer that reshapes the input data to fit the LSTM layers.
  3. First LSTM Layer with 64 units and returns sequences.
  4. Dropout Layer (0.2) for regularization
  5. Second LSTM Layer with 32 units.
  6. Dropout Layer (0.2) for regularization
  7. A Dense Layer with 16 neurons and ReLU activation function.
  8. Output Layer with a single neuron and Sigmoid activation function to predict sleep efficiency.

  The Adam optimizer is then used, MSE and MAE are calculated, and the model is complied

- **Ensemble Model**: Takes in the input features and has two branches:

    1. Neural Network Branch
    2. LSTM Branch

  The outputs of both branches are then averaged and combined to produce the final prediction of sleep efficiency.

- **Model Training and Evaluation**: Trains each of the three models individually.

    1. Each model is trained for 50 epochs.
    2. Each model is trained with a batch size of 32.
    3. A validation split of 0.2 is created to monitor the model's performance on unseen data
    4. Early Stopping is implemented to prevent overfitting. The model training stops if the validation loss doesn't improve for 10 epochs, restoring the best weights to prevent overfitting.

  After training, each of the models is evaluated using the test data, and the MSE and R^2 scores are calculated.

3 Data Description

The dataset used for training the model was sourced from Kaggle https://www.kaggle.com/datasets/equilibrium /sleep-efficiency. It contains information about a group of test subjects and their various sleep-related parameters. The dataset has 15 columns and 454 rows. The features of the dataset are:

1. ID: a unique identifier for each test subject
2. Age: age of the test subject
3. Gender: male or female
4. Bedtime: the time the test subject goes to bed each night
5. Wakeup time: the time the test subject wakes up each morning
6. Sleep duration: the total amount of time the test subject slept (in hours)
7. Sleep efficiency: a measure of the proportion of time in bed spent asleep
8. REM sleep percentage: the percentage of total sleep time spent in REM sleep
9. Deep sleep percentage: the percentage of total sleep time spent in deep sleep
10. Light sleep percentage: the percentage of total sleep time spent in light sleep
11. Awakenings: the number of times the test subject wakes up during the night
12. Caffeine consumption: the number of times the test subject wakes up during the night
13. Alcohol consumption: the amount of alcohol consumed in the 24 hours prior to bedtime (in oz)
14. Smoking status: whether or not the test subject smokes
15. Exercise frequency: the number of times the test subject exercises each week

More detailed analysis about each of the features (including distribution, mean, standard deviation, and count) are available in the Kaggle link easing the data analysis part.

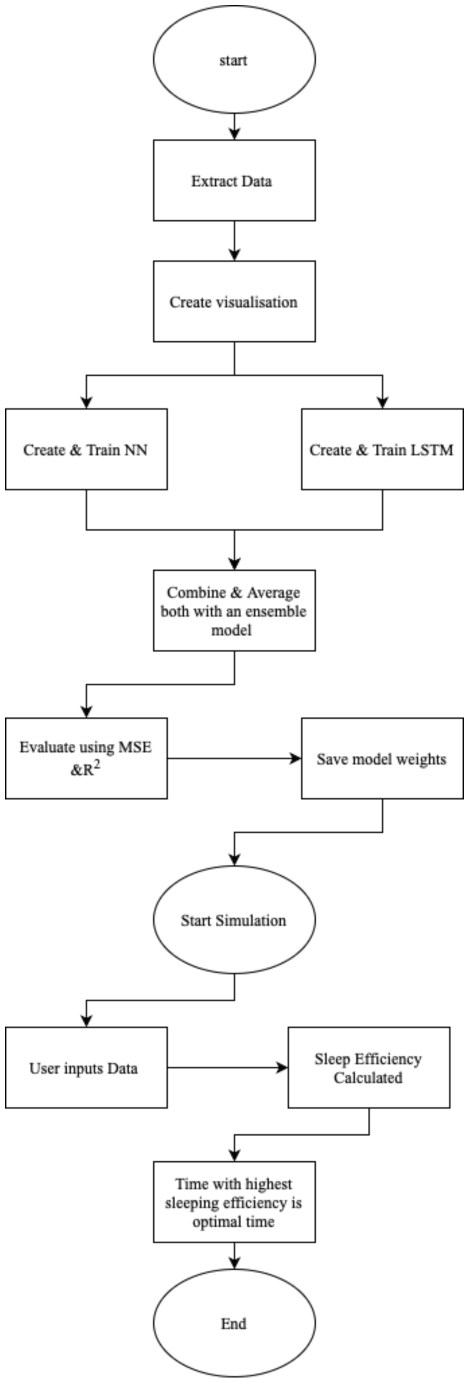
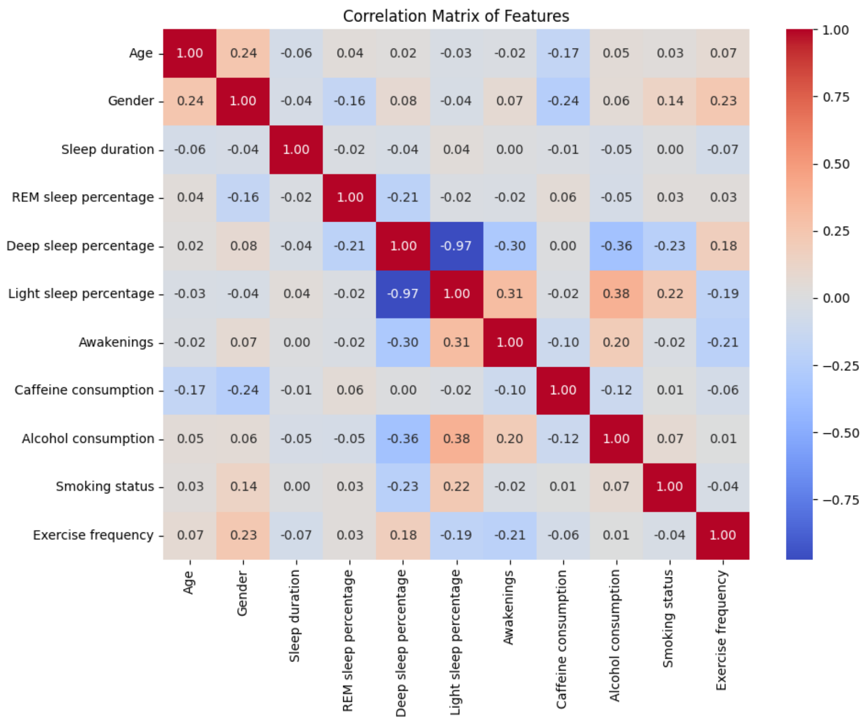

start
Extract Data
Create visualisation
Create & Train NN
Create & Train LSTM
Combine & Average both with an ensemble model
Evaluate using MSE &R^2
Save model weights
Start Simulation
User inputs Data
Sleep Efficiency Calculated
Time with highest sleeping efficiency is optimal time
End

Figure 1: System Flowchart



	Age	Gender	Sleep duration	REM sleep percentage	Deep sleep percentage	Light sleep percentage	Awakenings	Caffeine consumption	Alcohol consumption	Smoking status	Exercise frequency
Age	1.00	0.24	-0.06	0.04	0.02	-0.03	-0.02	-0.17	0.05	0.03	0.07
Gender	0.24	1.00	-0.04	-0.16	0.08	-0.04	0.07	-0.24	0.06	0.14	0.23
Sleep duration	-0.06	-0.04	1.00	-0.02	-0.04	0.04	0.00	-0.01	-0.05	0.00	-0.07
REM sleep percentage	0.04	-0.16	-0.02	1.00	-0.21	-0.02	-0.02	0.06	-0.05	0.03	0.03
Deep sleep percentage	0.02	0.08	-0.04	-0.21	1.00	-0.97	-0.30	0.00	-0.36	-0.23	0.18
Light sleep percentage	-0.03	-0.04	0.04	-0.02	-0.97	1.00	0.31	-0.02	0.38	0.22	-0.19
Awakenings	-0.02	0.07	0.00	-0.02	-0.30	0.31	1.00	-0.10	0.20	-0.02	-0.21
Caffeine consumption	-0.17	-0.24	-0.01	0.06	0.00	-0.02	-0.10	1.00	-0.12	0.01	-0.06
Alcohol consumption	0.05	0.06	-0.05	-0.05	-0.36	0.38	0.20	-0.12	1.00	0.07	0.01
Smoking status	0.03	0.14	0.00	0.03	-0.23	0.22	-0.02	0.01	0.07	1.00	-0.04
Exercise frequency	0.07	0.23	-0.07	0.03	0.18	-0.19	-0.21	-0.06	0.01	-0.04	1.00

Figure 2: Features Correlation Matrix

4 Experiments and Results

4.1 Experimental Setup

The model was trained using the following specifications:

- **Libraries**: Python, TensorFlow for deep learning, Pandas for data manipulation.
- **Training Data**: The dataset includes multiple weeks of sleep data from various users.
- **Training Time**: The model was trained on a local machine for 50 epochs and took a relatively short time to train since the data is numerical,
- **Model Configuration**: LSTM and fully connected layers for sequence learning and general prediction tasks.

4.2 Results

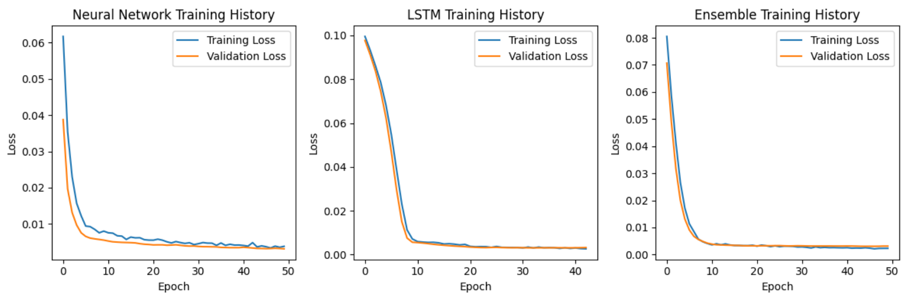


Figure 3: Training History

Model	MSE	R^2
NN	0.0028	0.85
LSTM	0.0028	0.85
Ensemble	0.0028	0.85

Table 1: Model Performance

The model's performance was evaluated using metrics such as Mean Squared Error (MSE) and R^2 score. The results demonstrate that the model accurately predicts optimal wake-up times based on sleep efficiency. The performance metrics are as follows:

We can see that all models performed the same, however the ensemble model was used for the final prediction as in the future when further development is done on this project and more data is added, the ensemble will combine the strengths of both NN and LSTM models.

4.3 Simulation and Further Work

The ultimate goal of this project is to create a smart alarm clock that will analyze the user's data from wearable devices and wake the user up at an optimal time. The user will enter a desired wake-up window before going to sleep and then the model will analyze and calculate sleep efficiency each 5 minutes, and then set an alarm for the wake-up time with the highest sleep efficiency as it would be the optimal one, and an alarm would be set automatically by the system.

A python file was created to simulate a simplified version of the project in the terminal. The user was asked to input the data since the model isn't connected to wearable's APIs yet, the sleep efficiency was calculated using the ensemble model, and the one with the highest efficiency was decided as the optimal wake-up time.

5 Conclusion

The AI-powered smart alarm clock system successfully predicts optimal wake-up time based on sleep efficiency, integrating data from the Apple Watch dataset. The system ensures that users are awakened during the optimal. Future work will aim to enhance the model's accuracy and extend integration with wearble's APIs, as well as creating a backend and frontend for the system and function to set an alarm automatically.

6 Supplementary Material

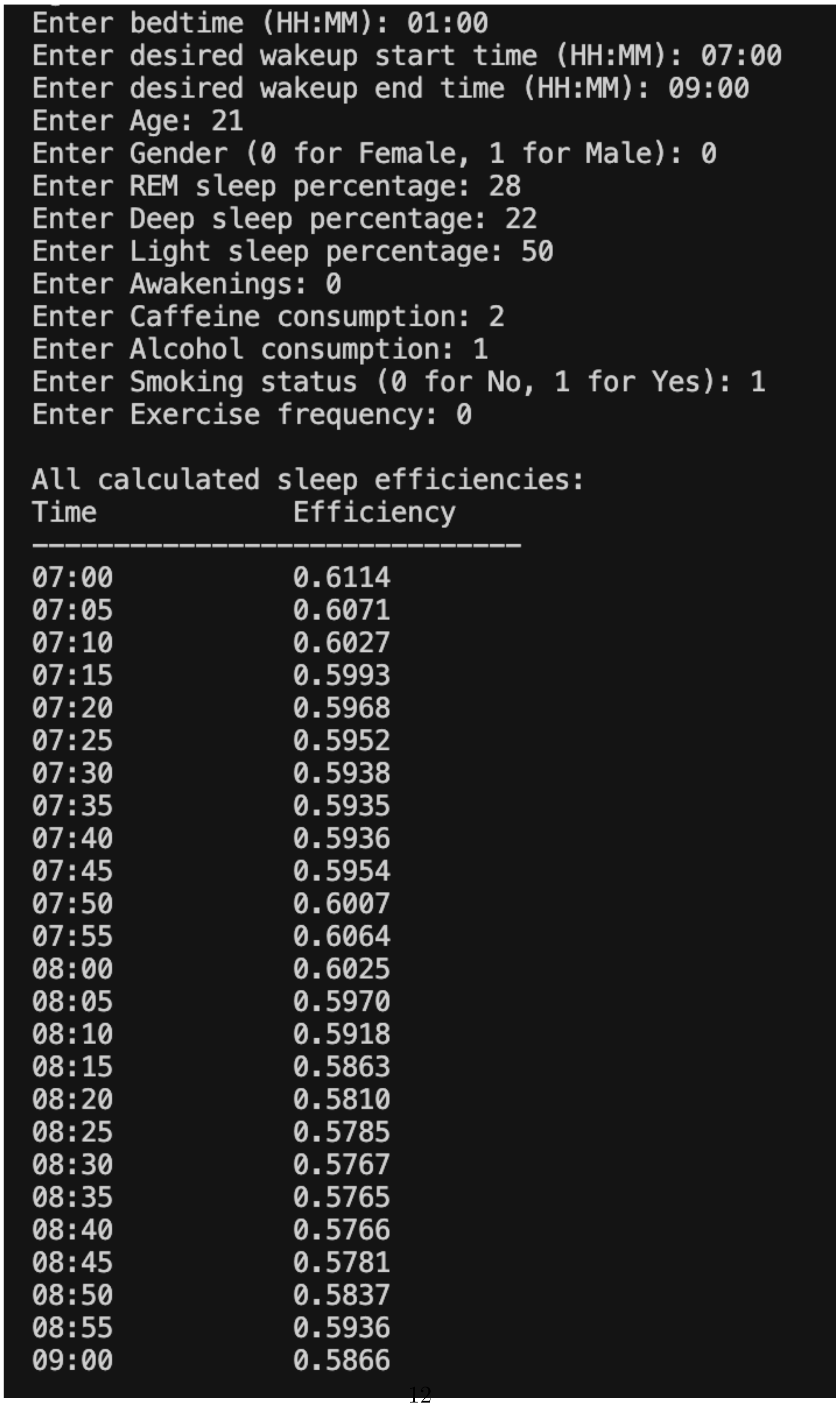
6.1 Code for Training the Model

The following Python code snippet demonstrates how the model was trained:

```
# Create Neural Network model
def create_nn_model(input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_dim=input_dim),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

# Create LSTM model
def create_lstm_model(input_dim):
    model = Sequential([
        LSTM(64, return_sequences=True, input_shape=(1, input_dim)),
        Dropout(0.2),
        LSTM(32),
        Dropout(0.2),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

# Create ensemble model
def create_ensemble_model(input_dim):
    # Input layer
    inputs = Input(shape=(input_dim,))
```



```
Enter bedtime (HH:MM): 01:00
Enter desired wakeup start time (HH:MM): 07:00
Enter desired wakeup end time (HH:MM): 09:00
Enter Age: 21
Enter Gender (0 for Female, 1 for Male): 0
Enter REM sleep percentage: 28
Enter Deep sleep percentage: 22
Enter Light sleep percentage: 50
Enter Awakenings: 0
Enter Caffeine consumption: 2
Enter Alcohol consumption: 1
Enter Smoking status (0 for No, 1 for Yes): 1
Enter Exercise frequency: 0

All calculated sleep efficiencies:
Time            Efficiency
------------------------------
07:00           0.6114
07:05           0.6071
07:10           0.6027
07:15           0.5993
07:20           0.5968
07:25           0.5952
07:30           0.5938
07:35           0.5935
07:40           0.5936
07:45           0.5954
07:50           0.6007
07:55           0.6064
08:00           0.6025
08:05           0.5970
08:10           0.5918
08:15           0.5863
08:20           0.5810
08:25           0.5785
08:30           0.5767
08:35           0.5765
08:40           0.5766
08:45           0.5781
08:50           0.5837
08:55           0.5936
09:00           0.5866
```

Figure 4: Example of Simulation

```
    # Neural Network branch
    nn_branch = Dense(64, activation='relu')(inputs)
    nn_branch = Dropout(0.2)(nn_branch)
    nn_branch = Dense(32, activation='relu')(nn_branch)
    nn_branch = Dropout(0.2)(nn_branch)
    nn_branch = Dense(16, activation='relu')(nn_branch)
    nn_branch = Dense(1, activation='sigmoid')(nn_branch)

    # LSTM branch
    lstm_branch = Reshape((1, input_dim))(inputs)
    lstm_branch = LSTM(64, return_sequences=True)(lstm_branch)
    lstm_branch = Dropout(0.2)(lstm_branch)
    lstm_branch = LSTM(32)(lstm_branch)
    lstm_branch = Dropout(0.2)(lstm_branch)
    lstm_branch = Dense(16, activation='relu')(lstm_branch)
    lstm_branch = Dense(1, activation='sigmoid')(lstm_branch)

    # Combine branches
    combined = Average()([nn_branch, lstm_branch])

    # Create model
    model = Model(inputs=inputs, outputs=combined)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

def train_and_evaluate_models():
    # Load and preprocess data
    X_train, X_test, y_train, y_test = load_and_preprocess_data()

    # Create early stopping callback
    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    )

    # Train individual models
    nn_model = create_nn_model(X_train.shape[1])
    lstm_model = create_lstm_model(X_train.shape[1])
    ensemble_model = create_ensemble_model(X_train.shape[1])

    # Train models with early stopping
    print("Training Neural Network model...")
    nn_history = nn_model.fit(
        X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)

print("\nTraining LSTM model...")
lstm_history = lstm_model.fit(
    X_train.reshape(-1, 1, X_train.shape[1]),
    y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)

print("\nTraining Ensemble model...")
ensemble_history = ensemble_model.fit(
    X_train,
    y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)

# Save ensemble model weights
ensemble_model.save_weights('ensemble_model_weights.h5')

# Evaluate models
nn_pred = nn_model.predict(X_test)
lstm_pred = lstm_model.predict(X_test.reshape(-1, 1, X_test.shape[1]))
ensemble_pred = ensemble_model.predict(X_test)

# Calculate metrics
models = {
    'Neural Network': nn_pred,
    'LSTM': lstm_pred,
    'Ensemble': ensemble_pred
}
```

6.2 Instructions for Setup and Running the System

To run the system, follow these steps:

1. Download the dataset: Visit(https://www.kaggle.com/datasets/laavanya/human-stress-detection-in-and-through-sleep). Download dataset then place it in the project root directory
2. Install Requirements `pip install -r requirements.txt`
3. Run and train the models `python3 sleep`$_e fficiency_m odel.py$
4. Run the simulation

   `python3 sleep`$_e fficiency_m odel.py$