National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Arun Edathil Veedu
Student ID: x22197931

School of Computing
National College of Ireland

Supervisor: Eugene Mclaughlin

National College of Ireland

Configuration Manual Submission Sheet

Student Name: ……Arun Edathil Veedu…….

Student ID: ……x22197931……

Programme: ……Cybersecurity…… **Year:** …2023-2024……

Module: ……MSc Research Practicum……

Lecturer: ……Eugene Mclaughlin……

Submission Due Date:
……12/08/2024……

Project Title: Integrating Tensor-Based Data Representation and CNN Model to Detect and Mitigate Noise-Based Data Manipulation Attacks on EEG Signals in BCI Systems

Word Count: ……1098……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: ……Arun Edathil Veedu……

Date: ……12/08/2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

Cybersecurity

Integrating Tensor-Based Data Representation with CNN Model to Detect and Mitigate Noise-Based Data Manipulation Attacks on EE Signals in BCI Systems

Your Name/Student Number	Course	Date
Arun Edathil Veedu	MSc Cybersecurity	12/08/2024

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilised in the process of completing this assignment.

Tool Name	Brief Description	Link to tool
NIL	NIL	NIL
NIL	NIL	NIL

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used**.

NIL	
NIL	
NIL	NIL

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

NIL

Additional Evidence:

NIL

Configuration Manual

Arun Edathil Veedu
Student ID: x22197931

1 Introduction

This configuration manual includes technical details of setup and requirements of the proposed model. These details include programming codes used for the implementation of models, technical specifications of software installed for this research purpose.

2 System Specification and Configuration

Mainly two types of machine learning techniques were used for this research. They require intense computational power with high-end personal computer. The process includes from dataset processing to generating classification reports such as accuracy, precision, and recall. For this purpose, a high-end gaming laptop with a powerful GPU is used. Additionally, a Brain Computer Interface Device is also used to collect real-time EEG data to implement the model in real-time.

Hardware Specification:

- **Laptop:** Asus TUF A15 Gaming.
- **CPU:** AMD Ryzen 7 4800H with Radeaon Graphics.
- **External GPU:** Nvidea RTX 3050 with 4GB RAM.
- **Hard drive:** 1TB SSD storage.

BCI Device Specifications:

- **Brand and model name:** Macrotellect Brainlink Lite v2.0.
- **Abilities:** They can be used to record basic brain signal spikes, such as the spikes during eye blink, eye movements, and eyebrow movements.
- **Other specifications:** This BCI device consists of three electrodes to capture the EEG signals from the brain. It then transferred to an external devices, such as a computer or a mobile phone via Bluetooth technology.

The proposed model is dependent on various software, tools, and libraries., which can be helpful in configuring the model solution. They are given below.

Software Specifications:

- **Operating System:** Microsoft Windows 11 x64 Home Edition
- **Python version:** Python 3.12 64-Bit
- **IDE:** Anaconda Navigator with Jupyter Notebook.

- **Software for EEG recording:** EEG recorder (Android)

Libraries used for the proposed model:

- **Numpy and Panda:** Used for extracting and preprocessing data.
- **Keras Framework:** This framework is used in neural network algorithms to anlyse data and its implementation.
- **Scikit-learn (Sklearn):** This library is used for classification and modelling of datasets.
- **Matplotlib:** This library is used for data visualization.
- **TensorFlow:** This library is used for building and training deep learning models.

3 Implementation of the Proposed Model

After installing necessary software and tools in the system, all important libraries were imported to the jupyter notebbok.

```
[37]: import numpy as np
      import pandas as pd
      from sklearn.preprocessing import MinMaxScaler, StandardScaler
      from sklearn.model_selection import train_test_split
      from imblearn.over_sampling import RandomOverSampler
      import tensorflow as tf
      from tensorflow.keras import layers, models
      from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
      import matplotlib.pyplot as plt
```

Figure 1 Libraries

After importing these libraries, the EEG datasets are fetched from the system. There are two types of datasets for the proposed model they are: the dataset recorded using a BCI device, and a complex dataset collected from Kaggle. These two datasets will be called in the python environment for training and testing. The real-time dataset contains limited features with fewer values, but the second dataset is complex and raw.

```
import pandas as pd

df = pd.read_csv('bci sample 2.csv')

print(df.head())
```

```
   timestamp  signal  attention  meditation    delta   theta  lowAlpha  \
0          1       0         47          54  1598284   90588     26306
1          2       0         44          47    61321   14801      1613
2          3       0         44          50   214042   97682     14663
3          4       0         48          69   301327   92323     63508
4          5       0         41          63   279796  111691     33574

   highAlpha  lowBeta  highBeta  lowGamma  highGamma  blinkValue
0     116472    81872     34126      9133         -1          -1
1       3105     4725      2311      1632         -1          -1
2      14225     2677      5674      1484         -1          -1
3      18250    19815     19419      1643         -1          -1
4      27550    26465      9492      3975         -1          -1
```

Figure 2 Recorded dataset

```
df = pd.read_csv('features_raw.csv')

print(df.head())
```

```
        Fp1       AF3        F3        F7       FC5       FC1        C3  \
0  0.057813 -1.335266  4.640480  0.219573  7.473817  2.314842  1.918097
1  1.367408 10.259654  3.345409  7.897852 -2.446051 -1.655035 -6.301423
2 -1.783132  4.133553 -0.951680 -1.624803 -1.827309 -2.280364 -2.279225
3 -3.690217 -0.814000  2.295469  0.901445  8.323679  1.127906  6.356886
4  2.137114  6.420466  6.122230 10.015321  3.106394  3.183129  3.658535

         T7       CP5       CP1  ...        Cz        C4         T8  \
0 -9.257533  9.089943 -7.104519  ... -2.241480  1.415335   2.406646
1 -7.290317 -3.546453 -5.705187  ... -2.568397 -5.651418  -0.096730
2  9.151344 -0.239575 -0.057604  ... -2.132823 -0.521117   8.605298
3 11.642082  9.354154 -1.662478  ... -0.506117 -1.154866  -3.940251
4  4.571793  4.917712 -2.325940  ...  1.813907 -6.444635 -27.680880

         CP6       CP2       P4        P8       PO4        O2  Unnamed: 32
0 12.864059  4.021099 -2.828598 -2.588735  2.637905 -5.226618          NaN
1 -4.930759 -1.722504 -6.111309  0.094893 -3.521353  1.887093          NaN
2 -4.499946 -3.232839 -4.249645 -3.687167 -7.383004 -4.489537          NaN
3  7.390881  2.129897 -0.794675 -1.959021  2.774530 -6.323060          NaN
4  0.641364  1.996658 -0.445779  2.614021  6.161845  3.308816          NaN
```

Figure 3 Collected dataset

After fetching the datasets, the next step is to apply various preprocessing techniques to remove unwanted values or attributes.

```
In [52]:
import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('features_raw.csv')

# Inspect the DataFrame Columns
print("Columns in the DataFrame:")
print(df.columns)

# Check for missing values
print("Missing values before processing:")
print(df.isnull().sum())

# Handle missing values
# Drop columns with any missing values
df = df.dropna(axis=1, how='any')


# Check again to ensure there are no missing values
print("Missing values after processing:")
print(df.isnull().sum())

# Normalize/Standardize the Data
# Initialize a StandardScaler
scaler = StandardScaler()

# Identify numerical columns
num_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Standardize numerical columns
df[num_cols] = scaler.fit_transform(df[num_cols])
```

Figure 4 Data pre-processing

Before preprocessing, missing values were checked to understand the completeness of the dataset. The missing values were then removed to ensure the dataset is ready and complete for analysis. Applied standardization and normalization techniques, which is used to transform features to have a mean of 0 and standard deviation of 1. Numerical columns were identified and applied the standardization.

```
8]: # Generate synthetic data
np.random.seed(42)
n_synthetic = 1000
synthetic_data = np.random.normal(loc=0, scale=1, size=(n_synthetic, df.shape[1]))

# Introduce anomalies by adding some extreme values
anomalies = np.random.choice([0, 1], size=n_synthetic, p=[0.95, 0.05])
for i in range(n_synthetic):
    if anomalies[i] == 1:
        anomaly_feature = np.random.randint(0, df.shape[1])
        synthetic_data[i, anomaly_feature] += np.random.normal(loc=10, scale=5)

# Create DataFrame
synthetic_df = pd.DataFrame(synthetic_data, columns=df.columns)
synthetic_df['anomaly'] = anomalies

synthetic_df.to_csv('bci_sample_2_preprocessed_att1.csv', index=False)
```

Figure 5 Noise-based attack simulation

After pre-processing the data, synthetic data were generated by introducing anomalies. Random spikes or values were generated using the spike attack method. Then the generated data frame is mixed with existing dataset.

For feature extraction, two types of feature engineering techniques were used, manual feature engineering and automated feature extraction using PCA.

```
## Manual feature extraction
# Create new features
df['total'] = df.sum(axis=1)
df['mean'] = df.mean(axis=1)
df['std'] = df.std(axis=1)

## Automated feature extraction using PCA
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Perform PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Create new PCA feature columns
for i in range(X_pca.shape[1]):
    df[f'pca_{i+1}'] = X_pca[:, i]

# Explained variance ratio of each principal component
print("Explained Variance Ratio:", pca.explained_variance_ratio_)

#Selecting top 5 features based on variance
top_features = df.var().nlargest(5).index.tolist()
df_top_features = df[top_features]

# Display the DataFrame with top features
print("Top Features DataFrame:")
print(df_top_features.head())
```

Figure 6 Feature extraction

Using this method, three features including sum, mean, and standard deviation were created. Additionally, by applying automated techniques, PCA is used to reduce the dimensionality of data.

After selecting appropriate features, these features were separated from target variable from the dataset.

```
# Separate features and target
X = df.drop('anomaly', axis=1).values
y = df['anomaly'].values

ros = RandomOverSampler(random_state=42, sampling_strategy=0.5)
X_res, y_res = ros.fit_resample(X, y)
```

Figure 7 Feature seperation from target variable

Additionally, class imbalance was handled with Random Oversampling by initializing the instances of the **'RandomOverSampler'.**

The dataset is then divided into training and test sets.

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)
```

Figure 8 Training and testing dataset

The training set ('X_train', 'y_train) is used to train the model. Model evaluation set ('X_test', 'y_test') is used to evalute the performance of the model.

After training the datasets, the normalziation is applied to normalize features of training and test datasets using 'StandardScaler'.

```
# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 9 Normalization of training and testing datasets

At first the scaler is fitted on the training data and then transformed both training and testing datasets.

Later, the Isolation Forest algorithm was implemented with these datasets for detecting the anomalies.

```
# Initialize the Isolation Forest model
model = IsolationForest(contamination=0.05, random_state=42)

# Train the model
model.fit(X_train)
```

```
IsolationForest(contamination=0.05, random_state=42)
```

```
# Predict anomalies
y_pred = model.predict(X_test)

# Convert predictions to 0 (normal) and 1 (anomalous)
y_pred = np.where(y_pred == -1, 1, 0)
```

Figure 10 Isolation Forest algorithm

At first, an instance of the Isolation Forest model is created with specified contamination and random state parameters. The model is then trained using the training dataset to learn characteristics of normal and anomalous data.

After implementation of the Isolation Forest Model, Convolutional Neural Network Model is tested with both datasets.

```
# Reshape for CNN
X_train_reshaped = X_train_scaled.reshape(-1, 1, X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(-1, 1, X_test_scaled.shape[1], 1)
```

```
# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (1, 3), activation='relu', input_shape=(1, X_train_reshaped.shape[2], 1)),
    layers.MaxPooling2D((1, 2)),
    layers.Conv2D(64, (1, 3), activation='relu'),
    layers.MaxPooling2D((1, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Figure 11 CNN Model

At first, the dataset is reshaped into 2D arrays to perform the algorithm. Then a sequential model is defined with convolutional and pooling layers and compiling those layers with appropriate parameters for training. The Rectified Linear Unit (ReLU) activation function introduces non-linearity into the model. It allows the model to learn complex patterns.

The model is then trained using the KerasAPI.

```
# Train the model
history = model.fit(X_train_reshaped, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

Figure 12 Training CNN model

This model specifies the number of epochs, batch size, and validation split to ensure that the model learns effectively while being evaluated for its ability to find hidden pattern and data.

After training the dataset, the model is evaluated and generated predictions for the dataset.

```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Test Accuracy: {test_accuracy:.2f}')
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9802 - loss: 0.0875
Test Accuracy: 0.99
```

```
# Predict anomalies on the test set
y_pred = (model.predict(X_test_reshaped) > 0.5).astype("int32")
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Figure 13 Evaluation and Prediction

The first method is to evaluate the performance of the trained model on the test dataset. The prediction method finds probabilities for each sample in class 0 and class 1 are classified as normal or anomaly.

Finally, the results are evaluated using classification report and confusion matrix.

```
# Compare predictions with actual values
print(classification_report(y_test, y_pred, target_names=['Normal', 'Anomaly']))

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal', 'Anomaly'])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

Figure 14 Classification Report and Confusion Matrix

After successfully training the datasets with CNN model, Tesor-Based Data Representation is implemented.

```
# Import TensorFlow
import tensorflow as tf

# Create a 3D tensor using TensorFlow
tensor_tf = tf.random.uniform((2, 3, 4))

# Print the shape of the tensor
print("TensorFlow tensor shape:", tensor_tf.shape)

# Access a specific element in the tensor
element = tensor_tf[1, 2, 3]
print("Element at position [1, 2, 3]:", element.numpy())

# Perform a basic operation
tensor_tf_add = tensor_tf + 1

# Print the modified tensor
print("Modified tensor after addition:\n", tensor_tf_add.numpy())
```

Figure 15 Tensor Creation

The first step of the implementation is to create a 3D tensor using TensorFlow.

After creating the 3D tensor, a function is applied to check if there are any NaN values in a TesnorFlow tensor.

```
# Check if the tensor contains any NaN values
has_nan = tf.reduce_any(tf.math.is_nan(tensor_tf))
print("Does the tensor contain NaN values?", has_nan.numpy())

# Check the data type of the tensor
tensor_dtype = tensor_tf.dtype
print("Data type of the tensor:", tensor_dtype)
```

Figure 16 Checking for NaN values in the tensor

The next function is to retrieve and display the data type of the tensor. It is important for understanding the nature of the data and to ensure the compatibility with TensorFlow operations.

The converted tensor data is then encrypted using AES encryption.

```
# Function to encrypt data
def encrypt_data(data, key):
    # Initialize cipher and mode
    iv = os.urandom(16)  # Generate a random initialization vector
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()

    # Pad the data to be a multiple of block size (16 bytes for AES)
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(data) + padder.finalize()

    # Encrypt the data
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

    return iv, encrypted_data

# Convert the TensorFlow tensor to a byte array
tensor_bytes_tf = tensor_tf_add.numpy().tobytes()

# Generate a random encryption key
key = os.urandom(32)  # 256-bit key for AES-256

# Encrypt the tensor data
iv_tf, encrypted_tensor_data_tf = encrypt_data(tensor_bytes_tf, key)
```

Figure 17 Data encryption

After encrypting the tensor, three types of cipher attacks were employed to verify the effectiveness of the model.

```
def ciphertext_only_attack(encrypted_data):
    try:
        cipher = Cipher(algorithms.AES(key), modes.CBC(iv_tf), backend=default_backend())
        decryptor = cipher.decryptor()

        # Attempt decryption
        decrypted_data = decryptor.update(encrypted_data) + decryptor.finalize()

        # Remove padding
        unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
        unpadded_data = unpadder.update(decrypted_data) + unpadder.finalize()

        return unpadded_data
    except Exception as e:
        return str(e)

# Simulate ciphertext-only attack
result = ciphertext_only_attack(encrypted_tensor_data_tf)
print("Result of ciphertext-only attack:", result)
```

Figure 18 Ciphertext-only attack

```python
# Simulate known-plaintext attack
def simulate_known_plaintext_attack(known_plaintext, encrypted_data, key, iv):
    try:
        # Decrypt the ciphertext using the known key and IV
        decrypted_data = decrypt_data(encrypted_data, key, iv)

        # Convert decrypted bytes back to tensor
        tensor_shape = (2, 3, 4)  # Known shape of tensor
        decrypted_tensor = tf.convert_to_tensor(np.frombuffer(decrypted_data, dtype=np.float32).reshape(tensor_shape))

        # Check if decrypted tensor matches the known plaintext tensor
        return decrypted_tensor, tf.reduce_all(tf.equal(decrypted_tensor, known_plaintext))
    except Exception as e:
        return str(e), False

def main():
    # Create and encrypt tensor
    known_tensor, encrypted_tensor, key, iv = create_and_encrypt_tensor()

    # Convert the known tensor to bytes for the attack
    known_tensor_bytes = known_tensor.numpy().tobytes()

    # Simulate known-plaintext attack
    decrypted_tensor, success = simulate_known_plaintext_attack(known_tensor.numpy().tobytes(), encrypted_tensor, key, iv)

    # Output results
    if success:
        print("Attack Successful: Decrypted tensor matches known plaintext tensor.")
        print(f"Decrypted Tensor:\n{decrypted_tensor.numpy()}")
    else:
        print("Attack Failed: Decrypted tensor does not match known plaintext tensor.")
        print(f"Error or Decrypted Tensor:\n{decrypted_tensor}")

if __name__ == "__main__":
    main()
```

Figure 19 Known-plaintext attack

```python
def chosen_plaintext_attack(chosen_plaintext):
    # Encrypt the chosen plaintext
    iv, encrypted_chosen_plaintext = encrypt_data(chosen_plaintext, key)

    # Show how changing plaintext changes ciphertext
    return iv, encrypted_chosen_plaintext

# Create a new chosen plaintext tensor
chosen_tensor = tf.random.uniform((2, 3, 4))
chosen_tensor_bytes = chosen_tensor.numpy().tobytes()

# Simulate chosen-plaintext attack
iv_chosen, encrypted_chosen_data = chosen_plaintext_attack(chosen_tensor_bytes)
print("Encrypted chosen plaintext length:", len(encrypted_chosen_data))
```

Figure 20 Chosen-plaintext attack