Configuration Manual

MSc Research Project
M.Sc. in Cybersecurity

Jai Allahabadi
Student ID: X23218193

School of Computing
National College of Ireland

Supervisor: Prof. Evgeniia Jayasekera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jai Allahabadi		
Student ID:	X23218193		
Programme:	M.Sc. in Cybersecurity	**Year:**	2024-25
Module:	M.Sc. Research Project		
Lecturer:	Prof. Evgeniia Jayasekera		
Submission Due Date:	12th December 2024		
Project Title:	Hybrid Anomaly Detection Framework for Kubernetes Environments		
Word Count:	1120		

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 bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Jai Allahabadi

Date: 12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

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

Configuration Manual

Jai Allahabadi
Student ID: x23218193

Hybrid Anomaly Detection Framework for Kubernetes Environments

1 Introduction

The configuration manual of this project elaborates the comprehensive outline of this research project coving the configurations on which this project builds upon along with implementations steps, where hybrid model which employs LSTM, custom attention layer and Transformer network have been trained using both traditional and meta learning (MAML) methods on the preprocess dimensionality reduced data done by PCA and Autoencoders. This exhaustive manual comprises of all datasets been used, with the corresponding code and respective functionalities.

2 Configurations

2.1 Hardware

This project have employed Google Colab notebook to develop anomaly detection model for K8s.

- System Architecture: TPU
- TPU version : 2.8
- TPU Cores : 8
- Processor: Intel(R) Xeon(R) CPU @ 2.00GHz
- RAM : 334.6 GB (This project used up to 15GB)
- Disk: 225.3 GB

2.2 Software/Libraries

Below libraries have been used of Python to perform this research:

- Pandas 2.2.2
- Numpy 1.26.4
- Matplotlib 3.8
- Seaborn 0.13.2
- TensorFlow 2.15
- Imblearn 0.12.4
- Sklearn 1.5.2

3 Implementation

In this section, implementation steps have been elaborated with respective code snippets. In this study, I have used 2 datasets : Kubernetes based dataset () and NSL KDD (). I have performed below table 1 on Kubernetes based datasets and table 2 for NSL KDD dataset.

Feature Reduction Techniques	**Training Methods**
PCA	MAML
PCA	Traditional
Autoencoders	MAML
Autoencoders	Traditional

Table 1 : Experiments on Kubernetes dataset

Feature Reduction Techniques	**Training Methods**
Autoencoders	MAML
Autoencoders	Traditional

Table 2: Experiments on NSL KDD dataset

Below are the steps that have been followed to conduct above experiments.

3.1 Using Kubernetes dataset

Step 1: Open Google Colab notebook and connect with TPU v2.8 runtime resource.

Step 2: Download the data from Kaggle (*Link*) and upload on the google drive.

Step 3: Connect the notebook with the google drive where dataset has been saved. Import the NumPy and Pandas libraries in Colab notebook to load the dataset.

```
[ ] from google.colab import drive
    drive.mount('/content/drive', force_remount=True)

    Mounted at /content/drive

[ ] import pandas as pd
    import numpy as np

    k8s_df = pd.read_csv('/content/drive/My Drive/Datasets/Kubernetes.csv')

    print("Kubernetes Dataset loaded successfully.")

    Kubernetes Dataset loaded successfully.
```

Step 4: Data Pre-processing has been performed on the dataset, starting with label encoding into binary.

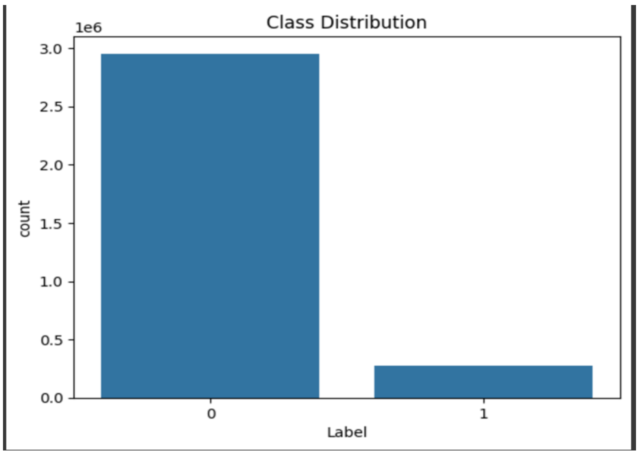
```
label_counts = k8s_df['Label'].value_counts()

print(label_counts)
```

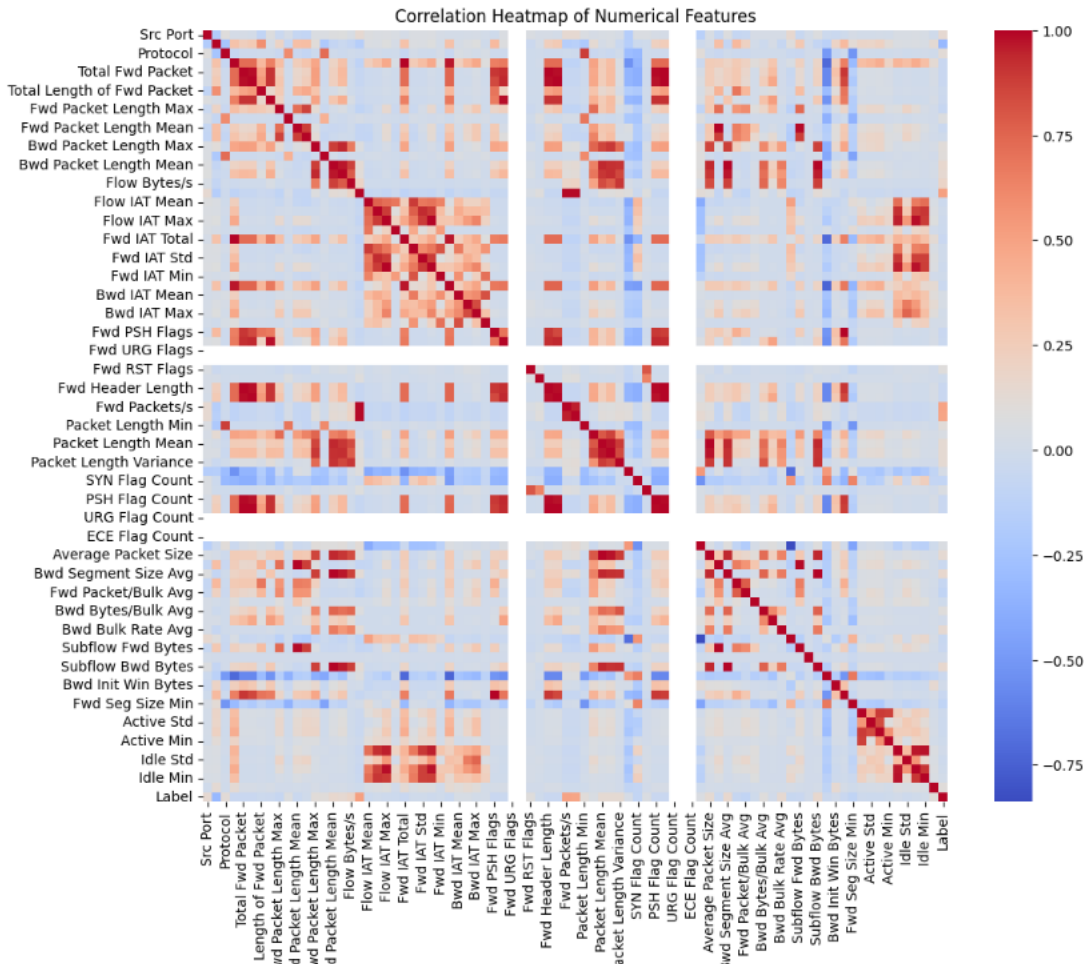
```
Label
0     2953291
2      156614
1      111251
11       8722
8         824
6         193
3         168
4         163
7         131
10         48
5          36
9          34
Name: count, dtype: int64
```

```
[ ]  k8s_df['Label'] = k8s_df['Label'].apply(lambda x: 1 if x != 0 else 0)
     print(k8s_df['Label'].unique())


     label_counts = k8s_df['Label'].value_counts()
     print(label_counts)
```

```
[0 1]
Label
0    2953291
1     278184
Name: count, dtype: int64
```

Step 5: Data have been explored with the help of visualization tools. Class distribution, correlation heatmap among others have been examined to understand the data.





Step 6: Handling the infinite values in the dataset to clean the dataset before normalizing the data using StandardScaler. The scaled dataset have been saved for further processing.

```
# Replace infinite values with NaN
k8s_df_cleaned.replace([np.inf, -np.inf], np.nan, inplace=True)

nan_counts = k8s_df_cleaned.isnull().sum()
print("Number of NaN values in each column:")
print(nan_counts[nan_counts > 0])

# Filling NaN values with the median of each column
k8s_df_cleaned.fillna(k8s_df_cleaned.median(), inplace=True)

# Feature scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(k8s_df_cleaned.drop('Label', axis=1))

# Create a DataFrame with scaled features
k8s_df_scaled = pd.DataFrame(scaled_features, columns=k8s_df_cleaned.columns[:-1])
k8s_df_scaled['Label'] = k8s_df_cleaned['Label']
```

```
Number of NaN values in each column:
Flow Bytes/s       41709
Flow Packets/s     41709
Flow IAT Mean      41709
Flow IAT Std       41709
Flow IAT Max       41709
Flow IAT Min       41709
dtype: int64
```

Step 7: After data has been pre-processed, high correlation features have been dropped to avoid redundancy, reduce computation overhead and improve PCA and Autoencoders results which were used to reduce the dimensionality of the dataset.

```python
corr_matrix = k8s_df_scaled.drop('Label', axis=1).corr().abs()

# Mask created to ignore self-correlations
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

threshold = 0.9
to_drop = [column for column in corr_matrix.columns if any(corr_matrix[column][corr_matrix[column] > threshold].index != column)]

print(f"Highly correlated features to drop (threshold > {threshold}):")
print(to_drop)

# Drop the highly correlated features
k8s_df_selected = k8s_df_scaled.drop(columns=to_drop)
```

```
Highly correlated features to drop (threshold > 0.9):
['Flow Duration', 'Total Fwd Packet', 'Total Bwd packets', 'Total Length of Bwd Packet', 'Fwd Packet Length Mean', 'Bwd Packet Length Max', 'Bwd Packe
```

```python
[ ]  from sklearn.decomposition import PCA


     X = k8s_df_selected.drop('Label', axis=1)
     y = k8s_df_selected['Label']


     # Apply PCA to retain 95% of the variance
     pca = PCA(n_components=0.95)
     X_pca = pca.fit_transform(X)


     # Create a DataFrame with PCA features
     pca_columns = [f'PC{i+1}' for i in range(X_pca.shape[1])]
     k8s_df_pca = pd.DataFrame(X_pca, columns=pca_columns)
     k8s_df_pca['Label'] = y.reset_index(drop=True)
```

```
[ ] import tensorflow as tf
    from tensorflow.keras.layers import Input, Dense
    from tensorflow.keras.models import Model
    from tensorflow.keras.callbacks import EarlyStopping

    # Define the size of the encoding dimension
    encoding_dim = 10


    input_data = Input(shape=(X.shape[1],))
    encoded = Dense(encoding_dim, activation='relu')(input_data)
    decoded = Dense(X.shape[1], activation='sigmoid')(encoded)


    autoencoder = Model(input_data, decoded)

    autoencoder.compile(optimizer='adam', loss='mean_squared_error')


    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

    # Train the model
    autoencoder.fit(X, X,
                    epochs=100,
                    batch_size=256,
                    shuffle=True,
                    callbacks=[early_stopping],
                    validation_split=0.2)

    # Extract the encoder to get reduced features
    encoder = Model(input_data, encoded)
    X_autoencoder = encoder.predict(X)

    # Create a DataFrame with autoencoder features
    ae_columns = [f'AE{i+1}' for i in range(X_autoencoder.shape[1])]
    k8s_df_ae = pd.DataFrame(X_autoencoder, columns=ae_columns)
    k8s_df_ae['Label'] = y.reset_index(drop=True)
```

Step 8: Once the features have been reduced using above techniques, data balancing have been done using SMOTE, to balance the benign and anomaly label data for enhancing the efficiency of training the hybrid model. Synthetic data has been generated for oversampling of 'Anomaly' label data. It is done after feature engineering so that it does have to generate synthetic data for all columns, which will make it more computationally exhaustive. It is done for both PCA based data as well as Autoencoder based data.

```
[ ] from imblearn.over_sampling import SMOTE

    # For PCA reduced data
    X_pca = k8s_df_pca.drop('Label', axis=1)
    y_pca = k8s_df_pca['Label']

    smote = SMOTE(random_state=42)
    X_pca_balanced, y_pca_balanced = smote.fit_resample(X_pca, y_pca)

    print("Balanced dataset shape after SMOTE (PCA):", np.bincount(y_pca_balanced))

    # For Autoencoder reduced data
    X_ae = k8s_df_ae.drop('Label', axis=1)
    y_ae = k8s_df_ae['Label']

    X_ae_balanced, y_ae_balanced = smote.fit_resample(X_ae, y_ae)

    print("Balanced dataset shape after SMOTE (Autoencoder):", np.bincount(y_ae_balanced))

    Balanced dataset shape after SMOTE (PCA): [2953291 2953291]
    Balanced dataset shape after SMOTE (Autoencoder): [2953291 2953291]
```

Step 9: Once all the pre-processing has been done and features have been reduced, which is now ready for training the model. All the data frames have been saved in google drive to avoid repeating the same process again and again.

```
import os

# Define the path
project_folder = '/content/drive/My Drive/k8s_security'

if not os.path.exists(project_folder):
    os.makedirs(project_folder)
    print(f"Created project folder at {project_folder}")
else:
    print(f"Project folder already exists at {project_folder}")

Created project folder at /content/drive/My Drive/k8s_security
```

```
[ ] # List of dataframes to save
    dataframes = {
        'k8s_df_cleaned.csv': k8s_df_cleaned,
        'k8s_df_scaled.csv': k8s_df_scaled,
        'k8s_df_selected.csv': k8s_df_selected,
        'k8s_df_pca.csv': k8s_df_pca,
        'k8s_df_ae.csv': k8s_df_ae
    }

    # Save dataframes to CSV
    for filename, df in dataframes.items():
        filepath = os.path.join(project_folder, filename)
        df.to_csv(filepath, index=False)
        print(f"Saved {filename} to {filepath}")
```

```
Saved k8s_df_cleaned.csv to /content/drive/My Drive/k8s_security/k8s_df_cleaned.csv
Saved k8s_df_scaled.csv to /content/drive/My Drive/k8s_security/k8s_df_scaled.csv
Saved k8s_df_selected.csv to /content/drive/My Drive/k8s_security/k8s_df_selected.csv
Saved k8s_df_pca.csv to /content/drive/My Drive/k8s_security/k8s_df_pca.csv
Saved k8s_df_ae.csv to /content/drive/My Drive/k8s_security/k8s_df_ae.csv
```

Step 10: Splitting both PCA and Autoencoder based data into train, validation and test data with 70:15:15.

```
[ ] from sklearn.model_selection import train_test_split

    # Split the PCA balanced dataset
    X_train_pca, X_temp_pca, y_train_pca, y_temp_pca = train_test_split(
        X_pca_balanced, y_pca_balanced, test_size=0.3, random_state=42, stratify=y_pca_balanced
    )
    X_val_pca, X_test_pca, y_val_pca, y_test_pca = train_test_split(
        X_temp_pca, y_temp_pca, test_size=0.5, random_state=42, stratify=y_temp_pca
    )

    print("PCA Training set size:", X_train_pca.shape)
    print("PCA Validation set size:", X_val_pca.shape)
    print("PCA Testing set size:", X_test_pca.shape)
```

```
PCA Training set size: (4134607, 12)
PCA Validation set size: (885987, 12)
PCA Testing set size: (885988, 12)
```

```
from sklearn.model_selection import train_test_split

# Split the Autoencoder balanced dataset
X_train_ae, X_temp_ae, y_train_ae, y_temp_ae = train_test_split(
    X_ae_balanced, y_ae_balanced, test_size=0.3, random_state=42, stratify=y_ae_balanced
)
X_val_ae, X_test_ae, y_val_ae, y_test_ae = train_test_split(
    X_temp_ae, y_temp_ae, test_size=0.5, random_state=42, stratify=y_temp_ae
)

print("Autoencoder Training set size:", X_train_ae.shape)
print("Autoencoder Validation set size:", X_val_ae.shape)
print("Autoencoder Testing set size:", X_test_ae.shape)
```

```
Autoencoder Training set size: (4134607, 10)
Autoencoder Validation set size: (885987, 10)
Autoencoder Testing set size: (885988, 10)
```

Step 11: Now hybrid model has been defined which uses LSTM, custom attention layer and transformer network, where LSTM used for capturing temporal dependency over series data and pass it over to custom attention layer to focus more important data points by using trainable weights and biases along with context vector. This will emphasize the important time steps before feeding it further to transformer model.

```
# Transformer encoder block
def transformer_encoder(inputs, num_heads, dff, d_model):
    attention_output = MultiHeadAttention(num_heads=num_heads, key_dim=d_model)(inputs, inputs)
    attention_output = Add()([inputs, attention_output])
    attention_output = LayerNormalization(epsilon=1e-6)(attention_output)

    ffn_output = Dense(dff, activation='relu')(attention_output)
    ffn_output = Dense(d_model)(ffn_output)
    ffn_output = Add()([attention_output, ffn_output])
    ffn_output = LayerNormalization(epsilon=1e-6)(ffn_output)

    return ffn_output


def create_anomaly_detection_model(input_shape):
    input_layer = Input(shape=input_shape)
    lstm_out = LSTM(64, return_sequences=True)(input_layer)
    attention_out = Attention()(lstm_out)
    dropout_out = Dropout(0.2)(attention_out)
    transformer_out = transformer_encoder(tf.expand_dims(dropout_out, axis=1), num_heads=4, dff=128, d_model=64)
    transformer_out = Flatten()(transformer_out)
    output_layer = Dense(1, activation='sigmoid')(transformer_out)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model

model = create_anomaly_detection_model((1, X_train_ae.shape[2]))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```


```
# Transformer encoder block
def transformer_encoder(inputs, num_heads, dff, d_model):
    attention_output = MultiHeadAttention(num_heads=num_heads, key_dim=d_model)(inputs, inputs)
    attention_output = Add()([inputs, attention_output])
    attention_output = LayerNormalization(epsilon=1e-6)(attention_output)

    ffn_output = Dense(dff, activation='relu')(attention_output)
    ffn_output = Dense(d_model)(ffn_output)
    ffn_output = Add()([attention_output, ffn_output])
    ffn_output = LayerNormalization(epsilon=1e-6)(ffn_output)

    return ffn_output


def create_anomaly_detection_model(input_shape):
    input_layer = Input(shape=input_shape)
    lstm_out = LSTM(64, return_sequences=True)(input_layer)
    attention_out = Attention()(lstm_out)
    dropout_out = Dropout(0.2)(attention_out)
    transformer_out = transformer_encoder(tf.expand_dims(dropout_out, axis=1), num_heads=4, dff=128, d_model=64)
    transformer_out = Flatten()(transformer_out)
    output_layer = Dense(1, activation='sigmoid')(transformer_out)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model

model = create_anomaly_detection_model((1, X_train_ae.shape[2]))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Below is the hybrid model summary which has shown each layer along with its parameters and it connection within the model.

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 1, 10)	0	-
lstm (LSTM)	(None, 1, 64)	19,200	input_layer[0][0]
attention (Attention)	(None, 64)	65	lstm[0][0]
expand_dims_layer (ExpandDimsLayer)	(None, 1, 64)	0	attention[0][0]
multi_head_attention (MultiHeadAttention)	(None, 1, 64)	66,368	expand_dims_layer[0][… expand_dims_layer[0][…
add (Add)	(None, 1, 64)	0	expand_dims_layer[0][… multi_head_attention[…
layer_normalization (LayerNormalization)	(None, 1, 64)	128	add[0][0]
dense (Dense)	(None, 1, 128)	8,320	layer_normalization[0…
dense_1 (Dense)	(None, 1, 64)	8,256	dense[0][0]
add_1 (Add)	(None, 1, 64)	0	layer_normalization[0… dense_1[0][0]
layer_normalization_1 (LayerNormalization)	(None, 1, 64)	128	add_1[0][0]
flatten (Flatten)	(None, 64)	0	layer_normalization_1…
dense_2 (Dense)	(None, 1)	65	flatten[0][0]

Step 12: Once hybrid model has been defined, now it has to be trained using traditional method and meta learning via MAML. Both training methods have been trained on both PCA and Autoencoders based extracted data.

Step 13: In the below code snippets the model has been trained with traditional model on both PCA and Autoencoders based data.

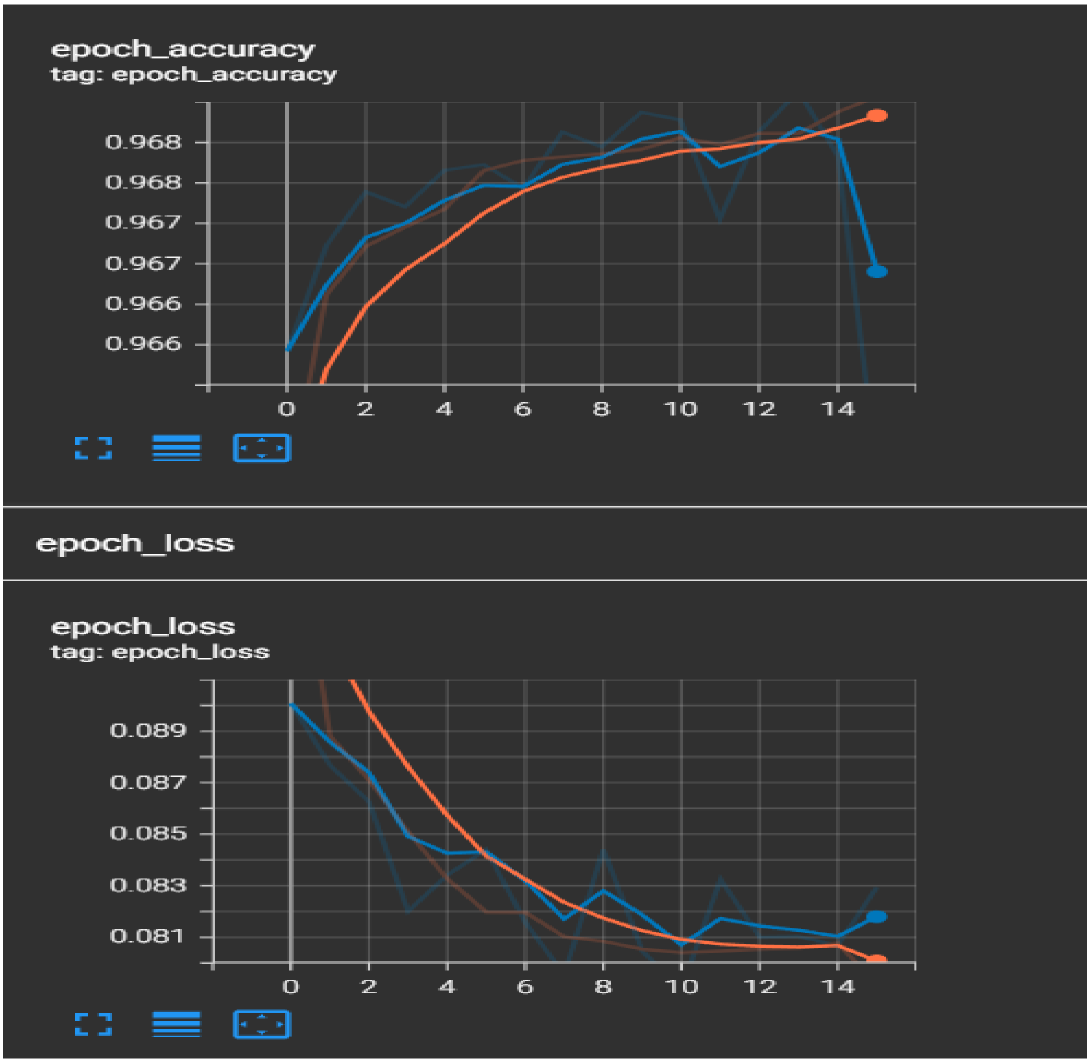
```python
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard


tensorboard_callback = TensorBoard(
    log_dir='logs/fit',
    histogram_freq=1,  # Log histogram of model weights every epoch
    write_graph=True,
    write_images=True
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

history = model.fit(
    X_train_ae, y_train_ae,
    epochs=20,
    batch_size=64,
    validation_data=(X_val_ae, y_val_ae),
    callbacks=[early_stopping, tensorboard_callback]
```

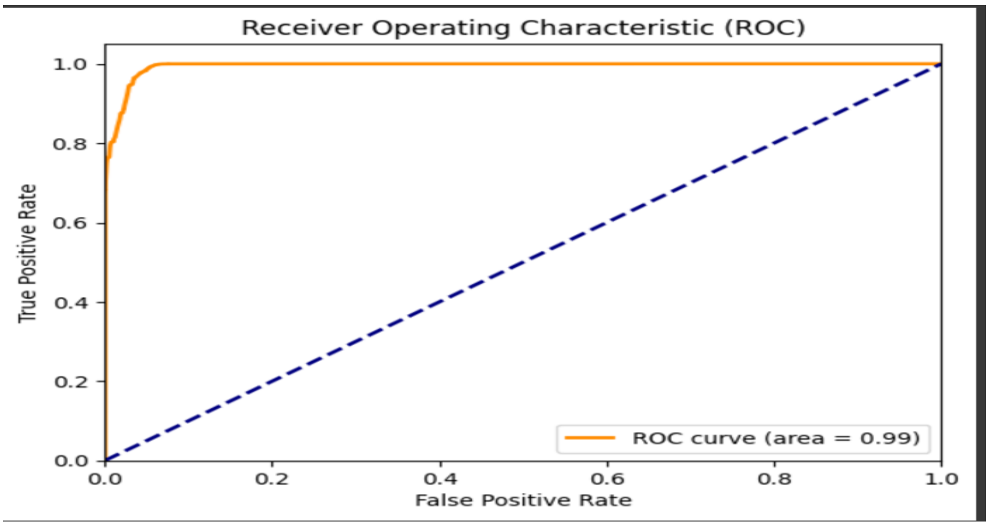
Evaluation

Tensor board has been to store the logs while training the model for further analysis on the hybrid model training. The hybrid model has been evaluated by showing test loss, test accuracy, classification report, along with epoch loss and epoch accuracy.

Below are code snippets for Autoencoder based data.

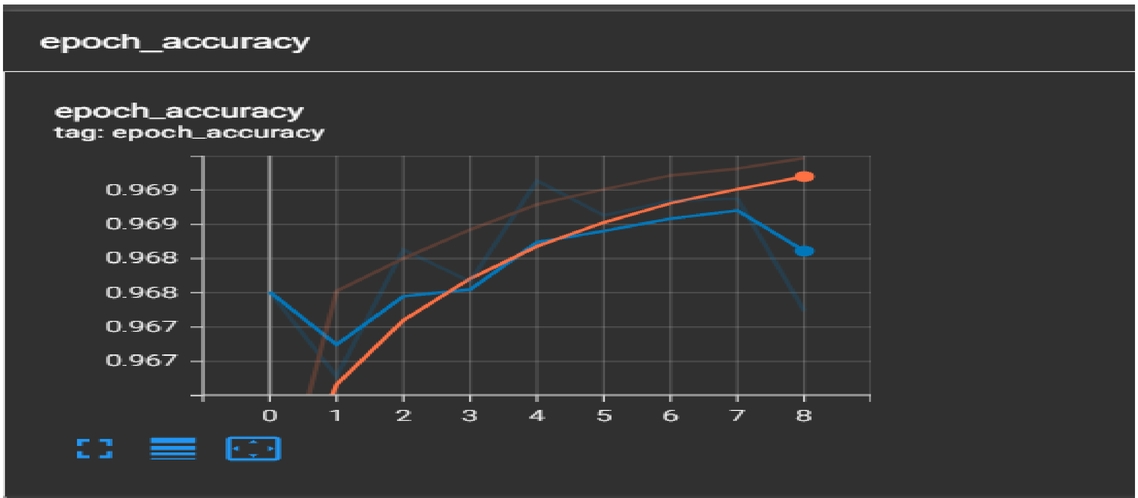
```
[ ] test_loss, test_accuracy = model.evaluate(X_test_ae, y_test_ae, verbose=1)
    print(f"Test Loss: {test_loss}")
    print(f"Test Accuracy: {test_accuracy}")

    27688/27688 [==============================] - 69s 2ms/step - loss: 0.0786 - accuracy: 0.9685
    Test Loss: 0.07857741415500641
    Test Accuracy: 0.9685255289077759

[ ] y_pred_probs = model.predict(X_test_ae)
    y_pred = (y_pred_probs > 0.5).astype(int)

    27688/27688 [==============================] - 67s 2ms/step

[ ] from sklearn.metrics import classification_report

    print(classification_report(y_test_ae, y_pred, target_names=["Class Benign", "Class Anomaly"]))

                  precision    recall  f1-score   support

    Class Benign       0.99      0.94      0.97    442994
   Class Anomaly       0.95      0.99      0.97    442994

        accuracy                           0.97    885988
       macro avg       0.97      0.97      0.97    885988
    weighted avg       0.97      0.97      0.97    885988
```

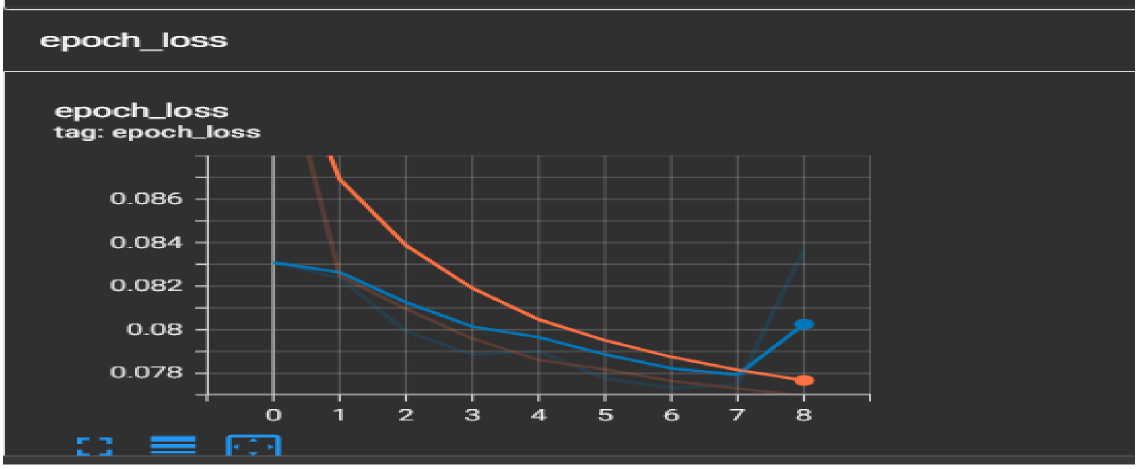

epoch_accuracy
tag: epoch_accuracy
0.968
0.968
0.967
0.967
0.966
0.966
0
2
4
6
8
10
12
14
epoch_loss
epoch_loss
tag: epoch_loss
0.089
0.087
0.085
0.083
0.081
0
2
4
6
8
10
12
14


Receiver Operating Characteristic (ROC)
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.99)

Below are code snippets for PCA based data.

```
[ ] test_loss, test_accuracy = model.evaluate(X_test_pca, y_test_pca, verbose=1)
    print(f"Test Loss: {test_loss}")
    print(f"Test Accuracy: {test_accuracy}")

27688/27688 [==============================] - 61s 2ms/step - loss: 0.1293 - accuracy: 0.9502
Test Loss : 0.12930119037628174
Test Accuracy : 0.9502476453781128

[ ] y_pred_probs = model.predict(X_test_pca)
    y_pred = (y_pred_probs > 0.5).astype(int)

27688/27688 [==============================] - 59s 2ms/step

[ ] from sklearn.metrics import classification_report

    print(classification_report(y_test_pca, y_pred, target_names=["Class Benign", "Class Anomaly"]))

                precision    recall  f1-score   support

  Class Benign       0.96      0.94      0.95    442994
 Class Anomaly       0.94      0.96      0.95    442994

      accuracy                           0.95    885988
     macro avg       0.95      0.95      0.95    885988
  weighted avg       0.95      0.95      0.95    885988
```





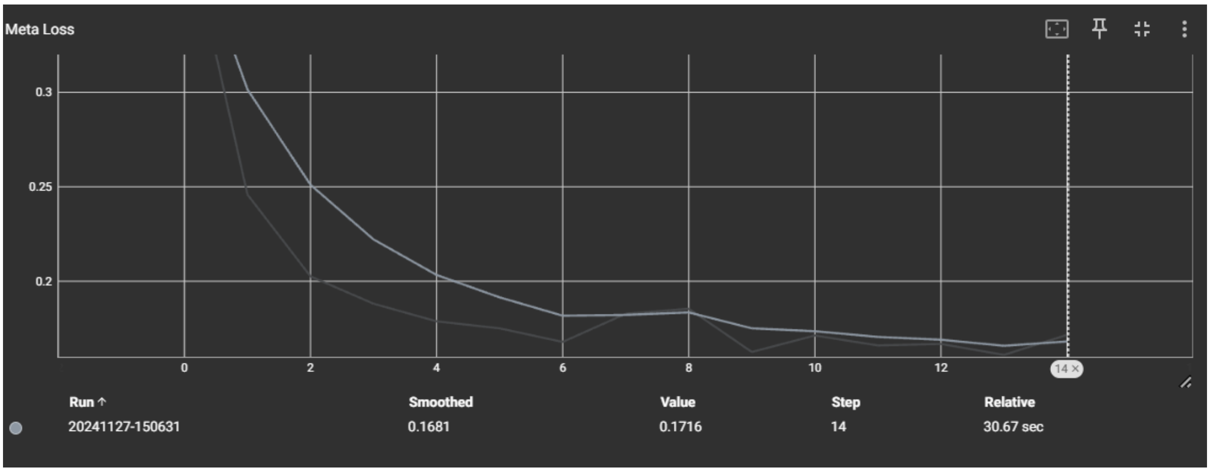
Step 14: Now, hybrid model will trained using meta learning, by creating tasks for normal behaviour and trained using inner and outer loop, aiming to achieve minimal meta loss.

```python
# Define a function to create tasks
def create_normal_behavior_tasks(X, y, num_tasks, k_shot_normal, k_query_anomalous):
    tasks = []
    normal_indices = np.where(y == 0)[0]  # For normal data
    anomalous_indices = np.where(y == 1)[0]  # For anomalous data

    for _ in range(num_tasks):
        # Samples of normal data for the support set
        normal_samples_support = np.random.choice(normal_indices, k_shot_normal, replace=False)
        # Samples of anomalous data for the query set
        anomalous_samples_query = np.random.choice(anomalous_indices, k_query_anomalous, replace=False)


        X_task = np.concatenate((X[normal_samples_support], X[anomalous_samples_query]), axis=0)
        y_task = np.concatenate((y[normal_samples_support], y[anomalous_samples_query]), axis=0)


        indices = np.arange(len(y_task))
        np.random.shuffle(indices)
        tasks.append((X_task[indices], y_task[indices]))
    return tasks

# Generate tasks for meta-training
tasks = create_normal_behavior_tasks(X_train_pca, y_train_pca, num_tasks=20, k_shot_normal=16, k_query_anomalous=8)
```

```python
@tf.function
def maml_inner_train(X_task, y_task, model, inner_lr):
    with tf.GradientTape() as tape:
        y_pred = model(X_task, training=True)
        y_task_reshaped = tf.expand_dims(y_task, axis=-1)
        task_loss = tf.keras.losses.binary_crossentropy(y_task_reshaped, y_pred)
    grads = tape.gradient(task_loss, model.trainable_variables)

    # Update the weights directly by applying gradient descent step-by-step
    for var, grad in zip(model.trainable_variables, grads):
        if grad is not None:
            var.assign_sub(inner_lr * grad)

    return task_loss
```

```python
# Meta-training loop with updated maml_inner_train function
meta_epochs = 15
inner_lr = 0.01  # Inner loop learning rate
meta_lr = 0.001  # Meta-learning rate
meta_optimizer = tf.keras.optimizers.Adam(learning_rate=meta_lr)

for epoch in range(meta_epochs):
    print(f'Meta Epoch {epoch + 1}/{meta_epochs}')
    meta_loss = 0
    for task_X, task_y in tasks:

        original_weights = [tf.identity(var) for var in model.trainable_variables]

        # Inner loop: Adapt model to the task
        maml_inner_train(task_X, task_y, model, inner_lr)

        # Compute the loss after adaptation
        y_pred_task = model(task_X, training=True)
        y_task_reshaped = tf.expand_dims(task_y, axis=-1)
        task_loss = tf.keras.losses.binary_crossentropy(y_task_reshaped, y_pred_task)
        task_loss = tf.reduce_mean(task_loss)

        # Reset weights to original before computing meta-gradient
        for var, original in zip(model.trainable_variables, original_weights):
            var.assign(original)

        # Compute meta-gradients with respect to the original weights
        with tf.GradientTape() as meta_tape:
            y_pred_task = model(task_X, training=True)
            task_loss = tf.keras.losses.binary_crossentropy(y_task_reshaped, y_pred_task)
            task_loss = tf.reduce_mean(task_loss)

        # Apply meta-update
        meta_grads = meta_tape.gradient(task_loss, model.trainable_variables)
        meta_optimizer.apply_gradients(zip(meta_grads, model.trainable_variables))

        meta_loss += task_loss.numpy()

    avg_meta_loss = meta_loss / len(tasks)
    print(f"Meta Epoch {epoch + 1} completed with Avg Meta Loss: {avg_meta_loss}")

    # Log to TensorBoard
    with tensorboard_callback.as_default():
        tf.summary.scalar('Meta Loss', avg_meta_loss, step=epoch)
```

The model has trained with MAML with above configurations, aiming to achieve minimal meta loss. This has been logged using tensor board as shown below:



The classification report for both PCA and Autoencoders are shown as below:

For Autoencoder:

```
27688/27688 [==============================] - 66s 2ms/step
Classification Report on Full Test Set:
              precision    recall  f1-score   support

      Normal       0.95      0.92      0.94    442994
   Anomalous       0.93      0.95      0.94    442994

    accuracy                           0.94    885988
   macro avg       0.94      0.94      0.94    885988
weighted avg       0.94      0.94      0.94    885988
```

For PCA:

```
27688/27688 [==============================] - 61s 2ms/step
Classification Report on Full Test Set:
              precision    recall  f1-score   support

      Normal       0.95      0.90      0.92    442994
   Anomalous       0.90      0.95      0.92    442994

    accuracy                           0.92    885988
   macro avg       0.92      0.92      0.92    885988
weighted avg       0.92      0.92      0.92    885988
```

3.2 For NSL KDD dataset

From all the experiments that have been conducted on Kubernetes dataset. It has been shown that Autoencoder techniques has shown better results than PCA. Hence, Autoencoder has been used on this dataset. Hence from steps 1 to 12 have been repeated with NSL KDD dataset. Hybrid have been defined with same parameters and layers of LSTM, attention layer and transformer network. Then both have been trained with both traditional method and MAML and evaluated upon.

Hybrid Model

```
import tensorflow as tf
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, Layer, MultiHeadAttention, LayerNormalization, Add, Flatten
from tensorflow.keras.models import Model
import tensorflow.keras.backend as K
from sklearn.metrics import classification_report, roc_curve, auc
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import TensorBoard
import datetime

# Splitting the data into train, test, and validation sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.2, random_state=42)

# model creation function with LSTM, Attention, and Transformer layers
class Attention(Layer):
    def __init__(self, **kwargs):
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name='att_weight', shape=(input_shape[-1], 1),
                                 initializer='glorot_uniform', trainable=True)
        self.b = self.add_weight(name='att_bias', shape=(input_shape[1], 1),
                                 initializer='zeros', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)
        e = K.squeeze(e, axis=-1)
        alpha = K.softmax(e)
        alpha = K.expand_dims(alpha, axis=-1)
        context_vector = x * alpha
        context_vector = K.sum(context_vector, axis=1)
        return context_vector
```

```
def transformer_encoder(inputs, num_heads, dff, d_model):
    attention_output = MultiHeadAttention(num_heads=num_heads, key_dim=d_model)(inputs, inputs)
    attention_output = Add()([inputs, attention_output])
    attention_output = LayerNormalization(epsilon=1e-6)(attention_output)

    ffn_output = Dense(dff, activation='relu')(attention_output)
    ffn_output = Dense(d_model)(ffn_output)
    ffn_output = Add()([attention_output, ffn_output])
    ffn_output = LayerNormalization(epsilon=1e-6)(ffn_output)

    return ffn_output

def create_anomaly_detection_model(input_shape):
    input_layer = Input(shape=input_shape)
    lstm_out = LSTM(64, return_sequences=True)(input_layer)
    attention_out = Attention()(lstm_out)
    dropout_out = Dropout(0.2)(attention_out)
    transformer_out = transformer_encoder(tf.expand_dims(dropout_out, axis=1), num_heads=4, dff=128, d_model=64)
    transformer_out = Flatten()(transformer_out)
    output_layer = Dense(1, activation='sigmoid')(transformer_out)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model


model = create_anomaly_detection_model((1, X_train.shape[2]))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Trained with Traditional Method

```
# Set up TensorBoard logging
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)


history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=20, batch_size=32, callbacks=[tensorboard_callback])


test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")


y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int).flatten()


print("Classification Report on NSL KDD Test Set:")
print(classification_report(y_test, y_pred_classes, target_names=['Normal', 'Anomalous']))
```

Trained with MAML learning

```
# Function to create tasks for MAML meta-learning
def create_normal_behavior_tasks(X, y, num_tasks, k_shot_normal, k_query_anomalous):
    tasks = []
    normal_indices = np.where(y == 0)[0]
    anomalous_indices = np.where(y == 1)[0]
    for _ in range(num_tasks):
        normal_samples_support = np.random.choice(normal_indices, k_shot_normal, replace=False)
        anomalous_samples_query = np.random.choice(anomalous_indices, k_query_anomalous, replace=False)
        X_task = np.concatenate((X[normal_samples_support], X[anomalous_samples_query]), axis=0)
        y_task = np.concatenate((y[normal_samples_support], y[anomalous_samples_query]), axis=0)
        indices = np.arange(len(y_task))
        np.random.shuffle(indices)
        tasks.append((X_task[indices], y_task[indices]))
    return tasks

# Generate tasks for MAML meta-training
tasks = create_normal_behavior_tasks(X_train, y_train, num_tasks=20, k_shot_normal=16, k_query_anomalous=8)

@tf.function
def maml_inner_train(X_task, y_task, model, inner_lr):
    with tf.GradientTape() as tape:
        y_pred = model(X_task, training=True)
        y_task_reshaped = tf.expand_dims(y_task, axis=-1)
        task_loss = tf.keras.losses.binary_crossentropy(y_task_reshaped, y_pred)
    grads = tape.gradient(task_loss, model.trainable_variables)
    for var, grad in zip(model.trainable_variables, grads):
        if grad is not None:
            var.assign_sub(inner_lr * grad)
    return task_loss
```

```
# Meta-training loop
meta_epochs = 15
inner_lr = 0.01
meta_lr = 0.001
meta_optimizer = tf.keras.optimizers.Adam(learning_rate=meta_lr)

for epoch in range(meta_epochs):
    print(f'Meta Epoch {epoch + 1}/{meta_epochs}')
    meta_loss = 0
    for task_X, task_y in tasks:
        original_weights = [tf.identity(var) for var in model.trainable_variables]
        maml_inner_train(task_X, task_y, model, inner_lr)
        y_pred_task = model(task_X, training=True)
        y_task_reshaped = tf.expand_dims(task_y, axis=-1)
        task_loss = tf.reduce_mean(tf.keras.losses.binary_crossentropy(y_task_reshaped, y_pred_task))
        for var, original in zip(model.trainable_variables, original_weights):
            var.assign(original)
        with tf.GradientTape() as meta_tape:
            y_pred_task = model(task_X, training=True)
            task_loss = tf.reduce_mean(tf.keras.losses.binary_crossentropy(y_task_reshaped, y_pred_task))
        meta_grads = meta_tape.gradient(task_loss, model.trainable_variables)
        meta_optimizer.apply_gradients(zip(meta_grads, model.trainable_variables))
        meta_loss += task_loss.numpy()
    avg_meta_loss = meta_loss / len(tasks)
    print(f"Meta Epoch {epoch + 1} completed with Avg Meta Loss: {avg_meta_loss}")
```

4 Conclusion

Researchers can generate same results by integrating same configuration for the framework as shown above and can further delve into future research and development.

References

Imbalanced-learn documentation—Version 0. 12. 4. (n.d.). Retrieved, from https://imbalanced-learn.org/stable/

Matplotlib—Visualization with python. Retrieved, from https://matplotlib.org/

Numpy . Retrieved December 10, 2024, from https://numpy.org/
Pandas—Python data analysis library.. Retrieved, from https://pandas.pydata.org/

Scikit-learn: Machine learning in python—Scikit-learn 1. 6. 0 documentation. (n.d.). Retrieved, from https://scikit-learn.org/stable/

Sever, Y., & Dogan, A. H. (2023). *A Kubernetes dataset for misuse detection*. https://doi.org/10.52953/fplr8631

Tensorflow. TensorFlow. Retrieved December 10, 2024, from https://www.tensorflow.org/

Waskom, M. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021