# Section 7.2: Code Understanding

## Annotated Code Walkthrough:

In this section we will go over the most important scripts from this paper. Mainly the ML training script and the Atribution testing script

The first script to be used by the paper is the ML training script.

**Overview**

The script performs the following main tasks:

* **Environment Setup**: Configures environment variables and suppresses certain warnings.
* **Data Loading**: Loads the training and testing datasets.
* **Data Splitting**: Splits the data into training and validation sets based on indices.
* **Model Training**: Trains the specified model using the provided data.
* **Model Saving**: Saves the trained model to disk for future use.

**Imports and Initial Setup**

# Generic python

import argparse

import pdb

import os

import sys

import json

import pickle

import time

# Ignore ugly futurewarnings from np vs tf.

import warnings

warnings.filterwarnings('ignore',category=FutureWarning)

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_curve, precision\_recall\_curve

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from tensorflow.keras.models import load\_model

import tensorflow as tf

* **Standard Libraries:** Imports standard Python libraries for argument parsing, file handling, and more.
* **Machine Learning Libraries:** Imports modules from scikit-learn and TensorFlow for model training and evaluation.
* **Custom Modules:** Imports custom modules lstm, cnn, gru, load\_train\_data, load\_test\_data, and utils from the local project.
* **Warnings Suppression:** Suppresses future warnings to keep the output clean.

**Function Definitions**

1. train\_forecast\_model

def train\_forecast\_model(model\_type, config, Xtrain, Xval, Ytrain, Yval):

* **Purpose:** Trains a specified machine learning model using the provided training and validation datasets.
* **Parameters**
  + model\_type: Type of the model to train ('GRU', 'LSTM', 'CNN').
  + config: Configuration dictionary containing training and model parameters.
  + Xtrain, Ytrain: Training data and labels.
  + Xval, Yval: Validation data and labels.
* **Process**
  + Extracts training and model parameters from the config.
  + Sets the input size parameter nI based on the shape of Xtrain.
  + Initializes the appropriate model class based on model\_type.
  + Calls create\_model to build the model architecture.
  + Trains the model using the train method with the provided data and parameters.
* **Returns:** An instance of the trained event\_detector model.

1. train\_forecast\_model\_by\_idxs

def train\_forecast\_model\_by\_idxs(model\_type, config, Xfull, train\_idxs, val\_idxs):

* **Purpose:** Trains a model using index-based subsets of the dataset for training and validation.
* **Parameters**
  + model\_type: Type of the model to train ('GRU', 'LSTM', 'CNN').
  + config: Configuration dictionary containing training and model parameters.
  + Xfull: Full dataset from which to extract training and validation data.
  + train\_idxs, val\_idxs: Indices for training and validation data.
* **Process**
  + Similar to train\_forecast\_model, but uses indices to select subsets from Xfull.
  + Initializes and trains the model using the train\_by\_idx method.
* **Returns:** An instance of the trained event\_detector model.

1. save\_model

def save\_model(event\_detector, config, run\_name='results'):

* **Purpose:** Saves the trained model to disk.
* **Parameters**
  + event\_detector: The trained model instance.
  + config: Configuration dictionary containing the model name.
  + run\_name: Directory under which the model will be saved.
* **Process**
  + Attempts to save the model in models/{run\_name}/{model\_name}.
  + Handles FileNotFoundError by saving in a default directory and prompts the user to create the intended directory.

1. Load\_saved\_model

def load\_saved\_model(model\_type, params\_filename, model\_filename):

* **Purpose:** Loads a previously saved model along with its parameters.
* **Parameters**
  + model\_type: Type of the model ('GRU', 'LSTM', 'CNN').
  + params\_filename: Path to the JSON file containing model parameters.
  + model\_filename: Path to the saved model file.
* **Process**
  + loads model parameters from the JSON file.
  + Initializes the appropriate model class.
  + Loads the saved Keras model into event\_detector.inner.
* **Returns:** An instance of the trained event\_detector model.

1. Parse\_arguments

def parse\_arguments():

* **Purpose:** Parses command-line arguments for configuring the script.
* **Process**
  + Uses argparse to define and parse arguments such as training parameters, model type, dataset name, etc.
  + Adds specific arguments for training parameters like epochs and batch size.
* **Returns:** An argparse.Namespace object containing the parsed arguments.

**Main execution**

if \_\_name\_\_ == "\_\_main\_\_":

* **Argument Parsing:** Calls parse\_arguments() to get user-specified configurations.
* **Environment Variables:**
  + Sets the visible CUDA devices for GPU computation.
  + Minimizes TensorFlow log levels to reduce verbosity.
* **Training Parameters Setup:**
  + Constructs the train\_params dictionary from parsed arguments.
  + Initializes the config dictionary to hold model configurations.
* **Run Name and Model Configuration:**
  + Sets run\_name from arguments.
  + Updates the config dictionary with model details using utils.update\_config\_model().
* **Data Loading:**
  + Calls load\_train\_data(dataset\_name) to load the full training dataset Xfull and sensor columns.
  + Calls load\_test\_data(dataset\_name) to load test data (although it's not used in training here).
* **Data Splitting:**
  + Determines the history length from config.
  + Uses utils.train\_val\_history\_idx\_split() to split indices for training and validation based on the dataset and history length.
* **Training Steps Calculation:**
  + Calculates steps\_per\_epoch and validation\_steps based on the length of indices and batch size.
* **Model Training:**
  + Calls train\_forecast\_model\_by\_idxs() to train the model using the full dataset and index splits.
* **Model Saving:**
  + Calls save\_model() to save the trained model to disk.
* **Completion Message:**
  + Prints "Finished!" to indicate the end of the script execution.

The second script we will go over is the attribution testing script

**Overview**

The script performs the following main tasks:

* **Argument Parsing**: Parses command-line arguments to configure the training process.
* **Environment Setup**: Configures environment variables and suppresses certain warnings.
* **Data Loading**: Loads the training and testing datasets.
* **Data Splitting**: Splits the data into training and validation sets based on indices.
* **Model Training**: Trains the specified model using the provided data.
* **Model Saving**: Saves the trained model to disk for future use.

**Imports and Initial Setup**

import numpy as np

import pdb

import pickle

import lime

import shap

import os

import sys

sys.path.append('..')

# Ignore ugly futurewarnings from np vs tf.

import warnings

warnings.filterwarnings('ignore',category=FutureWarning)

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from data\_loader import load\_train\_data, load\_test\_data

from main\_train import load\_saved\_model

from live\_bbox\_explainer.score\_generator import lime\_score\_generator, shap\_score\_generator, lemna\_score\_generator

from utils import attack\_utils, utils

* **Standard Libraries:** Imports standard Python libraries for argument parsing, file handling, and more.
* **Machine Learning Libraries:** Imports modules from scikit-learn and TensorFlow for model training and evaluation.
* **Custom Modules:** Imports custom modules lstm, cnn, gru, load\_train\_data, load\_test\_data, and utils from the local project.
* **Warnings Suppression:** Suppresses future warnings to keep the output clean.

**Function Definitions**

1. Explain\_true\_position

def explain\_true\_position(event\_detector, lookup\_name, attack\_idx, attacks, Xtest, method='MSE', expl=None, num\_samples=1):

* **Purpose:** Generates explanations for a specific attack based on its true position in the dataset.
* **Parameters**
  + event\_detector: The pre-trained model used for anomaly detection.
  + lookup\_name: A unique identifier for the model and dataset.
  + attack\_idx: The index of the attack being explained.
  + attacks: List of all attack start indices.
  + Xtest: The test dataset.
  + method: Explainability method ('MSE', 'LIME', 'SHAP', or 'LEMNA').
  + expl: Pre-initialized explainer object (optional, for SHAP or LIME).
  + num\_samples: Number of samples to generate explanations for.
* **Process**
  + Identifies the starting position of the attack (att\_start).
  + Iterates over num\_samples to create input-output pairs from Xtest.
  + Generates explanations using the selected method:
    - LEMNA, SHAP, LIME: Calls corresponding score generator functions.
    - MSE: Computes the squared error between the model prediction and the true label.
* **Returns:** Saves generated explanations in a .pkl file for later use.

1. Explain\_detect

def explain\_detect(event\_detector, lookup\_name, attack\_idx, attacks, Xtest, detection\_points, method='MSE', expl=None, num\_samples=1):

* **Purpose:** Explains the detection of an attack at the detection point determined by the model.
* **Parameters:** Same as explain\_true\_position, with an additional detection\_points parameter that maps attack indices to detection indices.
* **Process**
  + Similar to explain\_true\_position, but explanations are generated starting at the detection point.
  + Handles cases where the attack was not detected (attack\_idx not in detection\_points).
* **Returns:** Saves explanations to a .pkl file.

1. Parse\_argument

def parse\_arguments():

* **Purpose:** Parses command-line arguments to configure the script.
* **Parameters:** Supported arguments that can be parsed
  + attack: The attack index to explain.
  + --explain\_params\_methods: The explanation method ('MSE', 'LIME', 'SHAP', 'LEMNA').
  + --num\_samples: Number of samples to use for explanation.
* **Returns:** Parsed arguments as a namespace object.

**Main Execution Flow**

**Steps in Main Execution**

* Parse Arguments:
  + Reads user-specified configurations for attack index, explanation method, and number of samples.
* Environment Configuration:
  + Sets GPU visibility and suppresses TensorFlow logs to reduce verbosity.
* Model Loading:
  + Loads a pre-trained model using load\_saved\_model.
* Attack Metadata:
  + Loads attack indices using attack\_utils.get\_attack\_indices.
  + Retrieves detection points from a stored pickle file.
* Explainability Setup:
  + Initializes SHAP or LIME explainers if required by the selected method.
* Explain Attacks:
  + Calls explain\_detect to explain detection behavior.
  + Calls explain\_true\_position to explain attack behavior based on the true position.
* Completion:
  + Prints a message indicating the script has finished.

## Extensions or Insights:

After testing the code can propose the following as improvements

* The code can update its use of only python libraries. The old will be deprecated and marginal performance improvement are available when updating to the new packages.
* The code is not optimized with GPU performance in mind. In our testing we found the processing time and performance identical whether we use our CPU or GPU. So refactoring the code and using newer tenser flow version will increase the GPU utilization and decrease processing time.
* We also noticed that the paper did not utilize new attribution methods that are more specific to network anomaly detection. The explainers are general and not IDS specific.
* We also found it cumbersome to use the code and test is since it is mainly command line based. We propose the development of a simple GUI to make the usage of the code easier
* Lastly, we recommend exporting the results in a standard format so that visualizer and dashboarding software can easily parse them for easier consumption.

# Section 7.3: Input and Output Demonstrations

## Input-Output Mapping:

**Objective**: Explain how input data maps to model outputs.

**Key Points**:

* **Input format:** ICS time-series data (e.g., SWaT, WADI, TEP datasets).
* Data preprocessing steps (normalization, batching).
* **Outputs:** The attribution methods scores in term of ranks.

## Test Cases:

The attribution methods were test under both practical and ideal condition which are referred to as practical rank, and best guess rank. Practical ranks is testing the attribution method from point of detecting the anomaly; where in best-guess rank the attribution method starts from the point of the anomaly itself which would clear any noise.

The following table shows the result of testing the attribution ranking from different methods on the TEP dataset. The test is done one attack using the CNN anomaly detection model. Later in the report we showcase a comparable finding to the paper.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Best guesss rank** | **Parctical rank** | **Best guess timning rank** | **Practical timning rank** |
| MSE | 1 | 1 | 8.44 | 7.62 |
| SM | 7 | 5 | 1.04 | 1.0 |
| SHAP | 6 | 4 | 6.526 | 3.586 |
| LEMNA | 2 | 4 | 3.5 | 4.593 |
| Ensemble |  |  | 1.026 | 1.0 |

Table 1: Results from attribution ranking from TEP dataset

## Comparative Analysis:

From the result’s in the paper in Table 2 it’s noticed that:

* MSE consistently outperforms other methods;
* SM performs well but lags behind MSE in some cases;
* SHAP and LEMNA are less effective in ranking manipulated features;
* SM excels in Practical Timing scenarios due to gradient-based sensitivity;
* Ensemble methods balance speed and accuracy effectively.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Best guesss rank** | **Parctical rank** | **Best guess timning rank** | **Practical timning rank** |
| MSE | 1 | 4 | 15.96 | 15.52 |
| SM | 11 | 14 | 1.56 | 1.22 |
| SHAP | 9 | 13 | 15.04 | 16.08 |
| LEMNA | 7 | 9 | 12.77 | 13.79 |
| Ensamble |  |  | 2.85 | 1.88 |

Table 2: The paperPaper results - Reproduced results – Attribution ranking for attack 1 on SWAT dataset using CNN

A diagram of a work flow

Description automatically generated

# Section 7.4: Validation

## Key metrics:

**Attribution ranking:**

The validation of attribution methods is done mainly by ranking the features. Each attribution method would rank the features from the top to least important, ideally the attribution method should rank the manipulated feature causing the anomaly at the top.

**The key metrics on the paper are as below:**

* Best-guess rank
* Practical rank
* Best-guess timing rank
* Practical timing rank

**Types of ranks:**

* **Best-guess rank**: the rank of the manipulated feature is done when the input window of the attenuations is starting directly at the anomaly point, this means as the time the anomaly happens, the attributions method starts.
* **Practical rank:** The rank of the manipulated features is done in a realistic manner, which means when the anomaly is detected, not when it’s happened. For example, if the anomaly happens at 4:30:20, and it got detected at 4:30:25; the attribution would start at 4:30:25 not at 4:30:20 (real-world timing).
* Timing ranks for best-guess timing and practical timing ranks compute the attributions during an optimal window align with the anomaly start or detection and maximize the relevancy.

**Ranking interpretations:**

* **Lower rank** (e.g, Rank 1) it means that the manipulated feature is considered the most critical by the attribution method.
* **Higher rank** (e.g Rank 23) it means the manipulated feature was not detected as one of the most critical features, which means lower accuracy.

In the paper the validation metric rank is also denoted as AvgRank, which is the average rank of manipulated feature across multiple anomalies.

## Reproduced Results:

On Table 3 below is the reproduced results, where Table 2 above is the original results from the paper. It’s noticed that the results are within acceptable margin of error based on paper recommendations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Best guesss rank** | **Parctical rank** | **Best guess timning rank** | **Practical timning rank** |
| MSE | 1 | 4 | 15.96 | 15.52 |
| SM | 11 | 14 | 1.56 | 1.22 |
| SHAP | 9 | 13 | 15.04 | 16.08 |
| LEMNA | 7 | 9 | 12.77 | 13.79 |
| Ensamble |  |  | 2.85 | 1.88 |

Table 3: Reproduced results – Attribution ranking for attack 1 on SWAT dataset using CNN