PROJECT REPORT

Subject: Intrusion Detection and Prevention System

NT204.O11.ATCL

Project name: (B16) A scalable distributed machine learning approach for attack detection

1. GENERAL INFORMATION:

Group: 09

Num	Full name	Student ID	Email
1	Tran Le Dai Nghia	20520252	20520252@gm.uit.edu.vn
2	Le Sy Cuong	20521149	20521149@gm.uit.edu.vn
3	Nguyen Dinh Kha	20520562	20520562@gm.uit.edu.vn
4	Chau Gia Khang	20521431	20521431@gm.uit.edu.vn

Task Assignment:

Num	Full name	Role/Responsibility
1	Tran Le Dai Nghia	*Team leader* Task allocation Assist with report writing and presentation preparation Execute programming and model implementation
2	Le Sy Cuong	*Member* Research and read articles. Assist with report writing and presentation preparation
3	Nguyen Dinh Kha	*Member* Research and read articles Assist with report writing and presentation preparation Execute programming and model implementation
4	Chau Gia Khang	*Member*

		Research and read articles. Execute programming Assist with report writing and presentation preparation

The section below in this report is the detailed documentation of the team's implementation.

DETAILED REPORT

I. Introduction

The Internet of Things (IoT) is integrating advanced technology into our everyday environment. It connects things like sensors and remote controls to the internet, enabling them to communicate. This brings significant benefits to fields such as health monitoring, traffic control, and emergency response.

Recently, there has been a trend of integrating these IoT devices with cloud technology, making devices like sensors available online. These devices, often simple and battery-operated, now need to handle changing demands for data processing and storage. To manage this, many are turning to 'edge computing,' placing some computing tasks closer to where data is collected, reducing the burden on central cloud systems. However, this new approach comes with its own challenges, particularly regarding security. Devices are now more exposed to risks such as network attacks and data breaches. To address this, we need advanced security systems right at the point of data collection. These systems must be smart enough to detect unusual activities that may indicate a network attack.

Developing specific detection systems for the edge faces challenges due to the limited processing and storage capabilities of edge devices. To overcome this, our work introduces a distributed detection scheme using Extreme Learning Machine (ELM) classifiers, leveraging the resources of High-Performance Computing (HPC) clusters in the cloud for training resource-intensive classifiers. Our experiments show that ELMs achieve superior overall performance and faster learning speed compared to other binary classification schemes. The design of the ELM classifier allows for efficient data analysis at the edge level, resulting in a scalable distributed attack detection architecture where features of traffic are processed rapidly. The effectiveness of this approach is further confirmed through extensive experiments on standard public network security datasets.

II. Related Tasks

Integrating IoT technology into various sectors demands a concentrated approach to network security, especially as traditional Network-Physical Systems evolve into industrial IoT applications. Key objectives in this realm include developing protective systems capable of rapidly detecting DoS attacks, ensuring resilience against attack propagation, and maintaining the integrity and reliability of all system components, including third-party components. Due to the diverse legal requirements across industries, establishing a common security framework for IoT solutions poses a challenge.

Numerous studies have been conducted to identify and mitigate security threats in IoT. Research has explored common security threats across the physical, network, and application layers of IoT architecture. High-level security measures such as device identification, authentication, and access control have been proposed to mitigate these threats. Methodological frameworks to identify and address security risks in IoT environments have been presented, focusing on specific security requirements and policies.

The challenge in designing security for cloud-based Network-Physical Systems, particularly the physical consequences of network attacks, has been a focal point of research. Proposals include cloud middleware with context-awareness to limit resource access based on contextual factors like location and time, aiming to mitigate sensor value manipulation risks. Other frameworks discussed include a security enforcement framework for monitoring energy consumption in smart grid infrastructures and application-layer protection solutions against HX-DoS attacks.

Additionally, distributed attack detection models have been designed to address fundamental issues in detecting and identifying attacks. Research has also been directed towards intrusion detection systems specifically tailored for IoT, identifying and detecting attacks related to wireless networks.

The use of machine learning in network attack detection is becoming prevalent due to the scale and complexity of modern systems. Proposed in the paper is the integration of machine learning technology with distributed computing capabilities in edge-cloud computing models, providing a new approach to effectively address security challenges in modern Network-Physical Systems.

III. Implementation Methodology

1. Attack Detection System based on ELM (Extreme Learning Machine)

The attack detection solution for the edge/cloud IoT environment utilizes ELM machine learning technology and edge/cloud infrastructure. The data processing process is divided into two stages. Firstly, the initial feature data is transformed into a random feature space. Subsequently, the ELM classifier is trained on this data to create an attack detection model. In the edge/cloud environment, this stage can be separated based on edge devices and the cloud. Data is transmitted from edge devices to the cloud for analysis, but before that, it is anonymized to protect sensitive information. The cloud is used to train the ELM classifier on anonymous data and build a detection model. This model is used by the classifier on edge devices to detect attacks and anomalies in the monitoring environment.

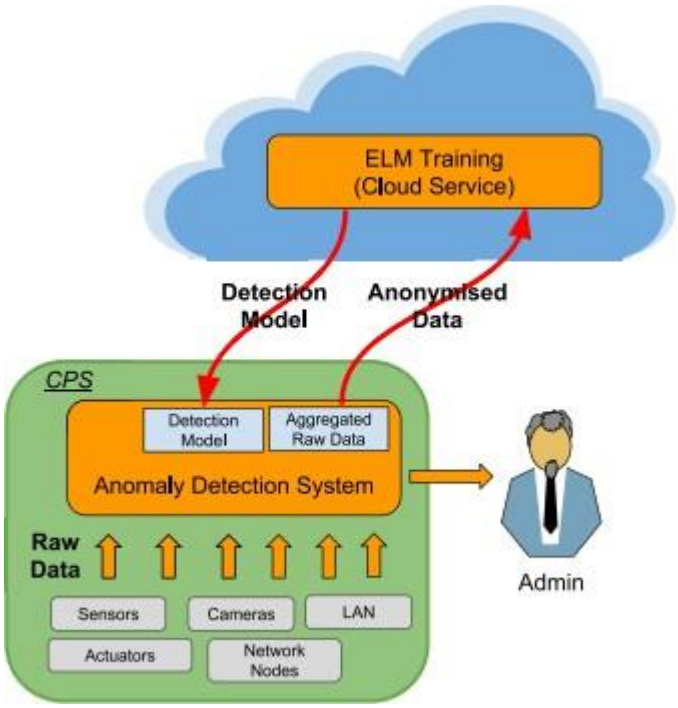

Figure 1. System Architecture Model

2. Network Traffic Collection for IDS

Data is sent by relevant network entities and structured as NetFlow collection items.

Attributes of NetFlow:

- **Node (or host) -** refers to a computer within an HPC cluster.

- ● **Driver (an application) -** is a module containing application code and communicates with the cluster manager.
- ● **Master node (cluster manager) -** communicates with the driver and is responsible for allocating resources to applications.
- ● **Worker node -** is a machine capable of executing application code and holding an instance.
- ● **Context (SparkContext) -** serves as the entry point for Spark functions, providing APIs for data operations (e.g., broadcasting variables, creating data, etc.).
- ● **Executor -** is a process on a worker machine that executes tasks.
- ● **Task -** is a unit of work sent to an executor.

3. Attribute Vector Extraction

For each TimeWindow, NetFlows are grouped by **WindowNumber** and **SourceAddress.** For each group, statistics on the following data are calculated:

- **Number of flows:** Count of NetFlow flows.
- **Sum of transferred bytes:** Total bytes transferred.
- **Average sum of bytes per NetFlow:** Average bytes per NetFlow flow.
- **Average communication time with unique IP addresses:** Average time spent communicating with unique IP addresses.
- **Number of unique IP addresses:** Count of unique IP addresses.
- **Number of unique destination ports:** Count of unique destination ports.
- **Number of unique protocols used by specific source IP addresses (e.g., TCP, UDP):** Count of unique protocols used by specific source IP addresses.

TimeWindow	SrcAddr	NumFlows	SumBytes	BytesPerF	mmunicati	umUniquel	mUniqueP	UniqueProtocols
2011-08-17 01:31:00	108.0.2.34	3	2158	719.3333	16.25026	1	2	2
2011-08-17 01:31:00	108.106.4	1	134	134	0.000528	1	1	1
2011-08-17 01:31:00	108.83.15	1	549	549	0.00057	1	1	1
2011-08-17 01:31:00	109.107.9	1	136	136	0.000334	1	1	1
2011-08-17 01:31:00	109.199.2	1	445	445	3450.963	1	1	1
2011-08-17 01:31:00	109.205.2	1	134	134	0.000479	1	1	1
2011-08-17 01:31:00	109.206.1	1	312	312	0.000492	1	1	1
2011-08-17 01:31:00	109.74.57.	1	270	270	3064.768	1	1	1

Multidimensional NetFlow items can be considered as time series. Furthermore, NetFlow items are divided into TimeWindows, and for each TimeWindow, traffic statistics are calculated.

For example, in the figure below, the model uses two scales, 1 minute and 2 minutes. The final attribute vector is obtained by concatenating the computed vectors for both TimeWindows.

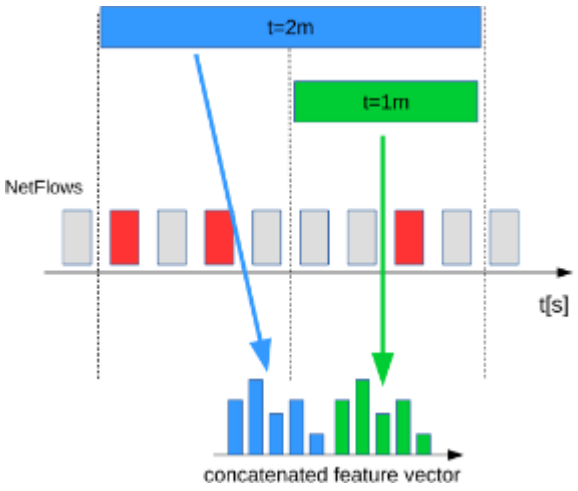

Figure 2. Illustration of attribute vector when divided into 2 time windows 1 minute and 2 minutes

4. Classification based on Extreme Learning Machine

ELM (Extreme Learning Machines) is a machine learning method used for data classification. It has a three-layer structure, including input layer, hidden layer, and output layer. The hidden layer uses nonlinear neurons to transform input data into a higher-dimensional space. The output layer uses linear neurons to generate prediction outputs.

ELM has several advantages. It can learn much faster than conventional neural networks and does not require complex iterative processes. It also addresses some common issues in gradient-based learning algorithms, such as local minima and overfitting.

In ELM, neurons in the hidden layer do not need to be adjusted during learning and can be randomly initialized. Output weights can be computed using the least squares regression method.

The ELM training process involves minimizing the approximation error. The best results can be achieved using the Moore-Penrose pseudo-inverse of the activation matrix.

In binary classification problems, classifying a new data sample is done by computing the activation of neurons in the hidden layer and applying output weights.

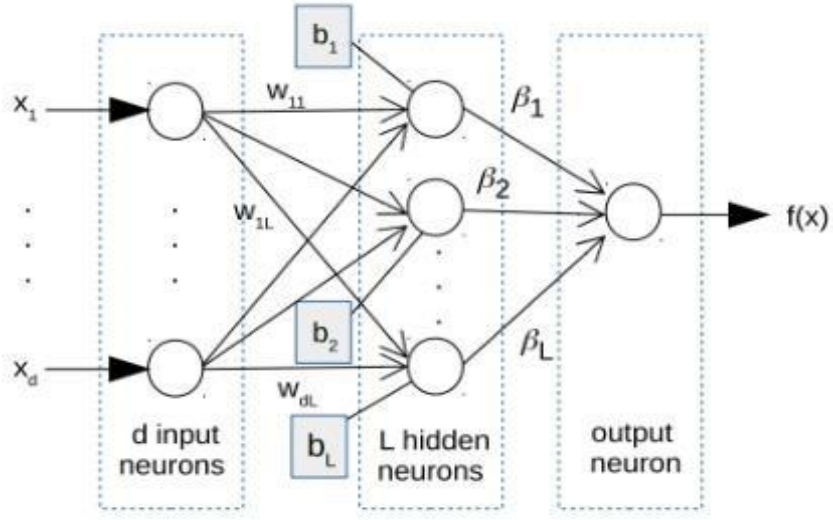

Fig. 3. Extreme learning machine.

5. Data Anonymization

The hidden layer of neurons used by ELM maps original attribute vectors to new attribute spaces.

The mapping process is only known to the data collection entity and therefore anonymization can be easily achieved on edge devices.

Here, we use the Sigmoid activation function. Each training sample x is transformed by a projection matrix a to produce an intermediate result. Then, bias is added. The final vector is obtained by passing the result from the previous operation through the sigmoid function.

$$f(x) = \frac{1}{(1 + e^{(-x)})}$$

6. Computation of Outer Layer Weights

In some machine learning applications, training samples need to be weighted. One reason for this may be data imbalance issues. In such cases, the number of training samples belonging to some classes may be larger than others, affecting machine learning algorithms and reducing classifier performance. In other words, most of the data contains clean traffic, while only a few data samples contain observations related to malware.

Cost-sensitive learning has been reported as the most effective solution for large imbalance issues. This can be applied in our specific environment based on ELM by assigning different weight coefficients to training errors. We can view the training process (estimating parameters βˆ) as a balancing problem and apply L2-regularized ridge regression in the following form:

$$\hat{\beta} = \left(\lambda I + H^T C H\right)^{-1} H^T C T.$$

Where, β is the model parameter to be optimized (which here is the output layer weight), λ is the regularization parameter, C is the sample weight, and H is the sample label.

To compute β, we use the following algorithm:

Algorithm 1 Distributed ELM training

Input: Cost matrix C, regularization factor λ, hidden layer response matrix H, and T vector of target labels.

Output: The output layer weights $\hat{\beta}$

1. Let Z be diagonal matrix of square roots of elements of C.
2. Let $A = ZH$ and $B = ZT$ (calculated using Map-Reduce).
3. Calculate (using Map-Reduce) the matrix $G = A^T A$.
4. From G matrix calculate right singular vectors V, singular values D, and left singular vectors U.
5. Calculate $\hat{\beta} = V(D^2 + \lambda)^{-1} D U^T B$

After computing the output layer weights, we will apply these parameters to the classifier to start training the model. Here, we use the sigmoid function to check whether a sample is malicious by examining the result of the sigmoid function.

```
# Function to apply sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Figure 5. Sigmoid function computation

- If the result of the sigmoid function > 0.5 then we classify it as malicious, and vice versa as benign.

IV. Implementation and Evaluation

1. Tools Used

Apache Spark framework: This framework provides tools for processing large amounts of data. It uses a data abstraction technique called RDD (resilient distributed dataset) to manage distributed data on an HPC cluster and to handle errors, all in a cloud environment. Here are some important concepts and components in the structure:

- **Node (or host)** - is a computer within an HPC cluster.
- **Driver (an application)** - is a module containing application code and communicates with the cluster manager.
- **Master node (cluster manager)** - communicates with the driver and is responsible for distributing resources to applications.
- **Worker node** - is a machine capable of executing application code and maintaining an instance.
- **Context (SparkContext)** - is the main entry point for Spark functions, providing an API for operations on data (e.g., broadcasting variables, creating data, etc.).
- **Executor** - is a process on a worker machine that executes tasks.
- **Task** - is a unit of work sent to an executor.

Additionally, this framework provides a library called MLib. This Apache Spark machine learning library is scalable, handling distributed matrices, allowing for distributed training of Extreme Learning Machine models on a large scale.

- Programming Language: Python.
- Programming Tool: Visual Studio 2022.

2. Sample Dataset:

- We used a dataset similar to the author's - the CTU-13 dataset (containing labeled traffic in attack scenarios), specifically using scenario 8 in this dataset:

Scen.	Total Flows	Botnet Flows	Normal Flows	C&C Flows	Background Flows
1	2,824,636	39,933(1.41%)	30,387(1.07%)	1,026(0.03%)	2,753,290(97.47%)
2	1,808,122	18,839(1.04%)	9,120(0.5%)	2,102(0.11%)	1,778,061(98.33%)
3	4,710,638	26,759(0.56%)	116,887(2.48%)	63(0.001%)	4,566,929(96.94%)
4	1,121,076	1,719(0.15%)	25,268(2.25%)	49(0.004%)	1,094,040(97.58%)
5	129,832	695(0.53%)	4,679(3.6%)	206(1.15%)	124,252(95.7%)
6	558,919	4,431(0.79%)	7,494(1.34%)	199(0.03%)	546,795(97.83%)
7	114,077	37(0.03%)	1,677(1.47%)	26(0.02%)	112,337(98.47%)
8	2,954,230	5,052(0.17%)	72,822(2.46%)	1,074(2.4%)	2,875,282(97.32%)
9	2,753,884	179,880(6.5%)	43,340(1.57%)	5,099(0.18%)	2,525,565(91.7%)
10	1,309,791	106,315(8.11%)	15,847(1.2%)	37(0.002%)	1,187,592(90.67%)
11	107,251	8,161(7.6%)	2,718(2.53%)	3(0.002%)	96,369(89.85%)
12	325,471	2,143(0.65%)	7,628(2.34%)	25(0.007%)	315,675(96.99%)
13	1,925,149	38,791(2.01%)	31,939(1.65%)	1,202(0.06%)	1,853,217(96.26%)

Figure 6. Scenarios in the dataset

- This file is originally in .pca file format and has been processed by our group into a spreadsheet for easy viewing and analysis. Our group also added a column named "Class" indicating the class to which the traffic belongs, where "1" indicates malicious traffic, and "0" indicates benign traffic.

StartTime	Dur	Proto	SrcAddr	Sport	Dir	DstAddr	Dport	State	sTos	dTos	TotPkts	TotBytes	SrcBytes	Label	Class
########	92.33551	tcp	147.32.84.	3058	->	222.189.2.	3389	FSPA_FSPA	0	0	12	1200	561	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	67.69266	tcp	147.32.84.	3059	->	222.189.2.	3389	FSPA_FSPA	0	0	11	1138	499	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	61.28217	tcp	147.32.84.	3060	->	222.189.2.	3389	FSPA_FSPA	0	0	10	1076	437	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	61.27032	tcp	147.32.84.	3061	->	222.189.2.	3389	FSPA_FSPA	0	0	10	1076	437	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	61.25626	tcp	147.32.84.	3062	->	222.189.2.	3389	FSPA_FSPA	0	0	9	1016	437	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	61.25645	tcp	147.32.84.	3063	->	222.189.2.	3389	FSPA_FSPA	0	0	10	1076	437	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	66.93957	tcp	147.32.84.	3064	->	222.189.2.	3389	FSPA_FSPA	0	0	12	1198	559	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1
########	64.20969	tcp	147.32.84.	3065	->	222.189.2.	3389	FSPA_FSPA	0	0	12	1271	572	flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Por	1

Figure 7. Spreadsheet of traffic for scenario 8

3. Method Used:

- First, our group processed the dataset data (originally a .pcap file containing information on netflows) and extracted the attributes of those netflows:

TimeWindowStart	SrcAddr	NumFlows	SumBytes	BytesPerF	mmunicati	umUniqueI	mUniqueP	UniqueProt	Class
2011-08-17 01:31:00	108.0.2.34	3	2158	719.3333	16.25026	1	2	2	0
2011-08-17 01:31:00	108.106.4	1	134	134	0.000528	1	1	1	0
2011-08-17 01:31:00	108.83.15	1	549	549	0.00057	1	1	1	0
2011-08-17 01:31:00	109.107.9	1	136	136	0.000334	1	1	1	0
2011-08-17 01:31:00	109.199.2	1	445	445	3450.963	1	1	1	0
2011-08-17 01:31:00	109.205.2	1	134	134	0.000479	1	1	1	0
2011-08-17 01:31:00	109.206.1	1	312	312	0.000492	1	1	1	0
2011-08-17 01:31:00	109.74.57.	1	270	270	3064.768	1	1	1	0
2011-08-17 01:31:00	109.93.19.	1	317	317	0.000522	1	1	1	0
2011-08-17 01:31:00	110.134.1	1	319	319	0.000553	1	1	1	0
2011-08-17 01:31:00	110.233.2	1	314	314	0.000575	1	1	1	0

Figure 8. Attribute extraction

- The next step is to anonymize the data using the sigmoid algorithm and based on the time distance between the data, which is 1 minute (our group relies on a time window of 1 minute and the next is:

0.754652	0.620128	0.648049	0.738604	0.661257	0.734008	0.579389	0.565083	0.653085	0.690359	0
0.702591	0.547845	0.603057	0.696814	0.645096	0.667923	0.553387	0.514287	0.629795	0.673092	0
0.702592	0.547846	0.603058	0.696815	0.645096	0.667923	0.553387	0.514287	0.629795	0.673093	0
0.702591	0.547845	0.603057	0.696814	0.645096	0.667923	0.553387	0.514287	0.629795	0.673092	0
0.77092	0.687963	0.726522	0.814222	0.7591	0.674653	0.741472	0.709317	0.801642	0.77229	0
0.702591	0.547845	0.603057	0.696814	0.645096	0.667923	0.553387	0.514287	0.629795	0.673092	0
0.702592	0.547845	0.603058	0.696815	0.645096	0.667923	0.553387	0.514287	0.629795	0.673093	0
0.763852	0.673403	0.713922	0.803046	0.747662	0.673903	0.723064	0.689683	0.785794	0.762319	0
0.702592	0.547845	0.603058	0.696815	0.645096	0.667923	0.553387	0.514287	0.629795	0.673093	0
0.702592	0.547845	0.603058	0.696815	0.645096	0.667923	0.553387	0.514287	0.629795	0.673093	0
0.702592	0.547845	0.603058	0.696815	0.645096	0.667923	0.553387	0.514287	0.629795	0.673093	0
0.702644	0.54795	0.603152	0.696911	0.645185	0.667928	0.553534	0.514434	0.629938	0.67317	0
0.702595	0.547849	0.60306	0.696816	0.645098	0.667925	0.553389	0.514289	0.629797	0.673094	0

Figure 9. Data anonymization

- After this step, the data has been anonymized and is secure even when stolen by attackers. The next step is to calculate the output layer weight. Our group closely adhered to and implemented the algorithm through the following code snippet:

```python
def calculate_output_weights_spark(C, lambda_, H, T):
    # Initialize Spark Session
    spark = SparkSession.builder.appName("calculate_output_weights").getOrCreate()

    # Convert numpy arrays to Spark DataFrames
    C_spark = spark.createDataFrame([(float(c),) for c in np.nditer(C)], ["C"])
    H_spark = spark.createDataFrame([(float(h),) for h in np.nditer(H)], ["H"])
    T_spark = spark.createDataFrame([(float(t),) for t in np.nditer(T)], ["T"])

    # Add a common column for joining
    C_spark = C_spark.withColumn("id", monotonically_increasing_id())
    H_spark = H_spark.withColumn("id", monotonically_increasing_id())
    T_spark = T_spark.withColumn("id", monotonically_increasing_id())

    # Join the dataframes
    Z_spark = C_spark.join(H_spark, "id").join(T_spark, "id")

    # Calculate Z, A, and B
    Z_spark = Z_spark.withColumn("Z", sqrt("C"))
    A_spark = Z_spark.withColumn("A", col("Z") * col("H"))
    B_spark = Z_spark.withColumn("B", col("Z") * col("T"))

    # Step 3: Calculate G
    G_pandas = A_spark.toPandas().transpose().dot(A_spark.toPandas())
    G_spark = spark.createDataFrame(G_pandas)

    # Convert DataFrame to IndexedRowMatrix for SVD
    G_indexed = IndexedRowMatrix(G_spark.rdd.zipWithIndex().map(lambda x: IndexedRow(x[1], list(x[0]))))

    # Step 4: Calculate SVD of G
    svd = G_indexed.computeSVD(G_indexed.numCols(), computeU=True)
    U = svd.U
    s = svd.s
    V = svd.V

    # Convert singular values to DenseMatrix for operations
    D = DenseMatrix(len(s), len(s), np.diag(s).tolist())

    # Convert DenseMatrix and IndexedRowMatrix to numpy arrays for multiplication
    V_np = np.array(V.toArray())
    D_np = np.array(D.toArray())
    U_np = np.array(U.toBlockMatrix().toLocalMatrix().toArray())
    B_spark_np = np.array(B_spark.collect())

    # Step 5: Calculate beta_hat
    beta_hat = V_np.dot((D_np**2 + lambda_)**(-1)).dot(D_np).dot(U_np.T).dot(B_spark_np.T)
```

Figure 10. Computation of output layer weight

- After this step, we will obtain a matrix β containing output layer weights. Next, we will use β (in the code snippet it is the variable beta_hat) to classify the samples:

```python
# Split the test data into features and labels
X_test = test_data.iloc[:, :num_features].values  # first 10 columns as features
y_test = test_data.iloc[:, -1].values  # last column as class label

# Initialize an empty list to store predicted labels
y_pred_labels = []

# Classification using ELM in batches
batch_size = 900  # Adjust the batch size as needed

for i in range(0, len(X_test), batch_size):
    batch_X_test = X_test[i:i+batch_size]  # Use slicing to get a batch
    batch_y_pred = sigmoid(np.dot(batch_X_test, beta_hat_padded*100))
    batch_y_pred_labels = (batch_y_pred > 0.5).astype(int)
    y_pred_labels.extend(batch_y_pred_labels)
```

Figure 11. Classification of samples

- To classify the samples, we compute the sigmoid index of the values in the sample vector and set a threshold to determine whether the sample is malicious or not (over 0.5 is malicious and vice versa).

```
for i in range(0, len(X_test), batch_size):
    batch_X_test = X_test[i:i+batch_size]  # Use slicing to get a batch
    batch_y_pred = sigmoid(np.dot(batch_X_test, beta_hat_padded*100))
    batch_y_pred_labels = (batch_y_pred > 0.5).astype(int)
    y_pred_labels.extend(batch_y_pred_labels)
```

Figure 12. Setting threshold and classifying samples

4. Experimental Results:

- Confusion Matrix:

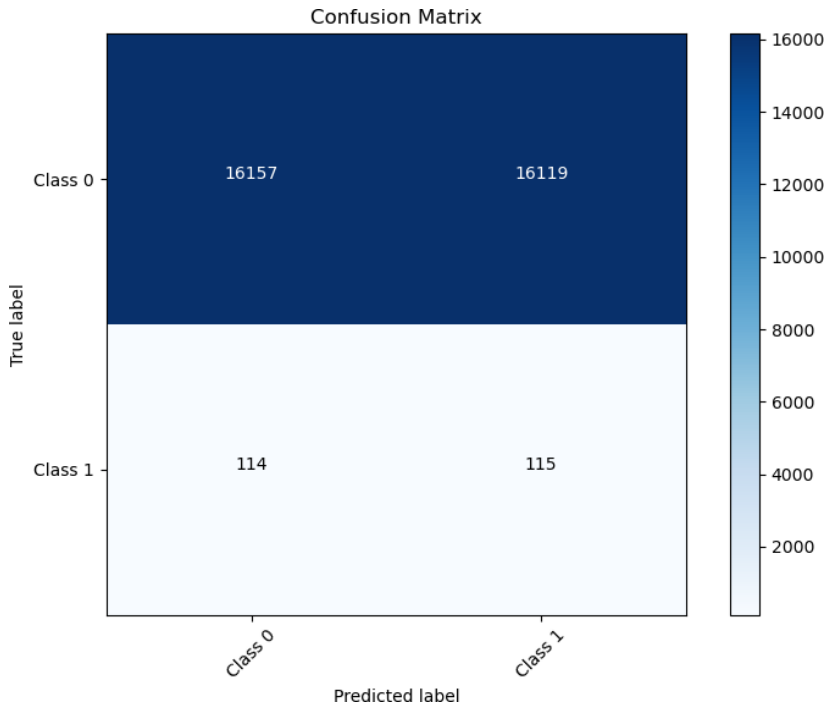

Figure 13. Classification matrix of ELM(0.2, 1) scenario

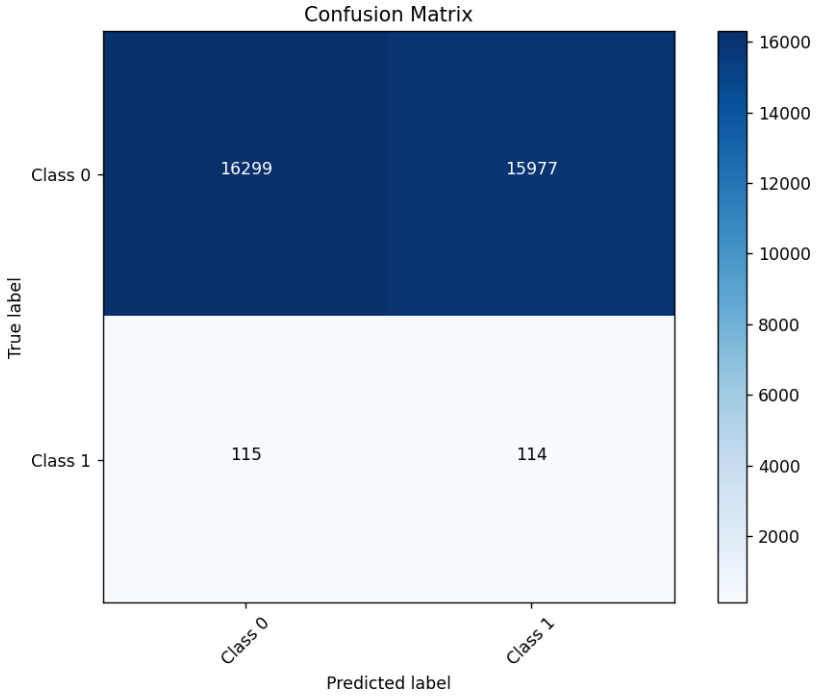

Figure 14. Classification matrix of ELM(0.2, 1) scenario

- Model Evaluation by Our Group:

Algorithm	Scanning Scenario (scenario implemented by our group)								
	TPR	FPR	TNR	FNR	P	ACC	ERR	FM	MC
ELM(0.2, 1)	0.99	0.99	0.01	0.01	0.50	0.67	0.50	0.67	0.00
ELM(0.3, 1)	0.99	0.99	0.01	0.01	0.50	0.67	0.50	0.67	0.00

- Comparison with the Author's Paper:

Table 2
Effectiveness of proposed method.

Algorithm	Scanning scenario								
	TPR	FPR	TNR	FNR	P	ACC	ERR	FM	MC
ELM(0.3,1)	1.00	0.02	0.08	0.00	0.92	0.86	0.02	0.96	0.82
ELM(0.2,1)	1.00	0.02	0.08	0.00	0.92	0.86	0.02	0.96	0.82
ELM(0.2,1+3)	1.00	0.01	0.07	0.00	0.98	0.99	0.01	0.99	0.91
Algorithm	C&C Communication								
	TPR	FPR	TNR	FNR	P	ACC	ERR	FM	MC
ELM(0.3,1)	0.21	0.01	0.99	0.79	0.92	0.76	0.24	0.35	0.32
ELM(0.2,1)	0.23	0.03	0.97	0.77	0.74	0.74	0.26	0.35	0.30
ELM(0.2,1+3)	0.21	0.03	0.97	0.79	0.76	0.74	0.26	0.33	0.28
Algorithm	Infected hosts								
	TPR	FPR	TNR	FNR	P	ACC	ERR	FM	MC
ELM(0.3,1)	0.62	0.01	0.99	0.38	0.97	0.90	0.09	0.76	0.66
ELM(0.2,1)	0.77	0.04	0.96	0.23	0.86	0.91	0.09	0.82	0.74
ELM(0.2,1+3)	0.83	0.01	0.99	0.17	0.95	0.95	0.05	0.88	0.83

Figure 15. Performance of the model implemented by our group

V. Conclusion and Future Work

This article has proposed a method for detecting attacks based on ELM technology using HPC cluster resources to train a classifier that is less time and resource consuming. The design and characteristics of ELM classification allow efficient computation and analysis of collected data in edge computing environments. The proposed method analyzes and classifies aggregate traffic flow online using NetFlows on specific edge nodes, close to the data source requiring protection. The article has presented the architecture of the proposed system and some implementation details. Specifically, the article has outlined how the process of training ELM classifiers can be separated and transferred to the cloud to leverage edge computing capabilities only for performing traffic classification based on more complex pre-built models. The article has also demonstrated the effectiveness of the proposed detection scheme through a series of experiments on real-world attack datasets.

In the future, we can enhance this system further by expanding the computing and storage capabilities of the cloud system, leading to superior computational and storage capabilities of classifiers trained on it.

VI. References

Kozik, R., Choraś, M., Ficco, M., & Palmieri, F. (2018). A scalable distributed machine learning approach for attack detection in edge computing environments. Journal of Parallel and Distributed Computing, 119, 18-26. https://doi.org/10.1016/j.jpdc.2018.03.006

Garcia, S., Grill, M., Stiborek, J., & Zunino, A. (2014). An empirical comparison of botnet detection methods. Computers and Security, 45, 100-123. doi:10.1016/j.cose.2014.05.011