

# Design and Development of an Explainable Artificial Intelligence Framework for Real-Time Cyber Threat Detection and Response in Enterprise Information Systems

<sup>1</sup>Omorie Michael, <sup>2</sup>Osita Miracle Nwakeze, <sup>3</sup>Chekwube Georgina Nwankwo, <sup>4</sup>Onukwu Petrus Oge

<sup>1,4</sup>Department of Computer Science, Ogwashiuku Polytechnic, Delta State, Nigeria

<sup>2,3</sup>Department of Computer Science, Chukwuemeka Odumegwu Ojukwu University, Uli, Anambra State, Nigeria

E-mail: <sup>1</sup>[omorogiemichael2015@gmail.com](mailto:omorogiemichael2015@gmail.com), <sup>2</sup>[ma.nwakeze@coou.edu.ng](mailto:ma.nwakeze@coou.edu.ng), <sup>3</sup>[cg.nwankwo@coou.edu.ng](mailto:cg.nwankwo@coou.edu.ng),

<sup>4</sup>[onukwupetrusoge@gmail.com](mailto:onukwupetrusoge@gmail.com)

**Abstract:** This study designs and develops an Explainable Artificial Intelligence framework for real-time cyber-threat detection and response in enterprise information systems. CICIDS2017 cyber-threat dataset was used to develop and evaluate the framework. Duplicate records, missing data values, identifiers, and variables with zero variance were removed from the dataset. The remaining features were reduced using correlation and mutual-information techniques and partitioned into training, testing, and validation sets in a stratified manner. Class weights were estimated based on inverse-variance frequencies. Extreme Gradient Boosting was used as the multiclass threat detection classifier while TreeSHAP was used to explain model predictions. A risk-based response decision engine was designed to merge prediction confidence with threat levels and deploy countervailing measures such as monitoring, alerting, rate-limiting, blocking, isolating, and escalating based on the risk scores of the events. Experiments revealed that the designed model achieved an accuracy of 97.84%, a macro precision of 91.26%, a macro recall of 88.43%, a macro F1-Score of 89.12%, a weighted F1-Score of 97.82%, and a false-positive rate of 1.89%. Moreover, the model achieved a macro-average ROC-AUC of 0.9721. SHAP values revealed that the Flow Duration, Total Forward Packets, and Flow Packets/s features had the most significant influence on the model's predictions. The end-to-end processing time of the framework was 7.07ms on average, which demonstrates its ability to perform real-time threat detection and response. This study concludes that detecting cyber-threats with explainable artificial intelligence and automatically responding to them using a risk-based decision engine provides a reliable cybersecurity framework for enterprise information systems.

**Keywords:** Explainable Artificial Intelligence, Cyber-Threat Detection, XGBoost, SHAP, Intrusion Detection.

## I. INTRODUCTION

Enterprise information systems are becoming more connected through cloud platforms, distributed databases, web applications, mobile apps, internet-of-things enabled devices, and remote access. While such interconnectivity offers numerous advantages to organizations, it also creates vulnerabilities that threat actors can use to launch denial-of-service attacks, credential attacks, reconnaissance scans, malware distribution campaigns, web exploits, and network intrusions. Traditional intrusion detection systems that rely on signature-based approaches are useful in detecting some forms of attack since they have a rich repository of known attack patterns and behaviors. However, they are not efficient in detecting novel

forms of attack with features that have never been seen before during training or have mutated beyond recognition (Sommer & Paxson, 2010). Sommer and Paxson (2010) note that building machine learning models for operational network intrusion detection is “fundamentally challenging” since real-life network data “differs significantly from the controlled environment of a testbed.” However, it is well established that both machine learning and data mining approaches have the potential to uncover network behaviors that are indicative of adversarial activity (Buczak & Guven, 2016). Modern intrusion detection research has focused on developing anomaly-based and machine learning-based models that can detect deviations from normal patterns and behaviors indicative of possible adversarial intent (Khraisat *et al.*, 2019). Deep and machine learning have become

critical areas of research in cybersecurity because they automate the feature extraction process during model training by operating directly on raw network data (MahdaviFar & Ghorbani, 2019).

However, the performance of artificial intelligence-driven intrusion detection systems is highly dependent on the quality and representativeness of the network data used to train machine learning models. Cybersecurity researchers require intrusion detection datasets that contain high representation of normal traffic and diverse attack classes to build models that can generalize well and distinguish between different attack types (Ferrag *et al.*, 2020). The CICIDS2017 intrusion detection dataset was designed to address the shortcomings of previous intrusion detection benchmarks by capturing realistic network profiles and modern attack methods in realistic network environments (Sharafaldin *et al.*, 2018). However, subsequent analysis revealed several inconsistencies during the traffic reconstruction and labeling processes which implies that additional preprocessing steps are required when using this dataset for machine learning experiments (Engelen *et al.*, 2021). Extreme Gradient Boosting is the most appropriate algorithm for classifying network flows in intrusion detection due to its regularized tree boosting framework which enables it to perform efficient non-linear classification while being robust to overfitting. This algorithm allows XGBoost to handle large structured datasets by capturing intricate relationships among multiple network flow features (Chen & Guestrin, 2016).

While achieving high predictive performance is critical when developing an intrusion detection model, it is often necessary for the security analyst to understand why the model predicts that a particular traffic event is malicious before taking any countermeasures. The opacity of traditional machine learning algorithms has prompted researchers to develop Explainable Artificial Intelligence methods that facilitate human understanding of the inner workings of neural networks and other machine learning methods (Adadi & Berrada, 2018). One of the primary reasons for the need for explainable AI systems is that the opacity of traditional machine learning models undermines transparency, accountability, interpretability, and trustworthiness, especially when the models make consequential decisions (Barredo Arrieta *et al.*, 2020). Local Interpretable Model-Agnostic Explanations make it possible to approximate the behavior of complex machine learning models by displaying input-output relationships in an intuitive and human-interpretable manner (Ribeiro *et al.*, 2016). Similarly, SHapley Additive exPlanations offer a unified framework for interpreting model predictions by estimating the relative contribution of each feature

to a particular prediction (Lundberg & Lee, 2017). Therefore, this study designs and develops an explainable AI framework that combines the prediction accuracy of Extreme Gradient Boosting, the model interpretability of SHAP, and a risk-based response decision engine that responds to predicted cyber-threats with appropriate countermeasures.

## II. METHODOLOGY

An experimental system development methodology was used to carry out this study. The study involved acquiring network data from the CICIDS2017 cybersecurity dataset on Kaggle, performing several preprocessing steps, developing a supervised machine learning model for threat detection, analyzing model performance and interpretability, designing a response decision engine, integrating the various components into a complete system, training the model, and evaluating the performance of the resulting system. Extreme Gradient Boosting was used as the multiclass threat detection classifier while TreeSHAP was used to explain model performance. The XGBoost classification probability and threat severity score were combined by the response decision engine to select the most appropriate response action. Google Colab was used for model development and experiments while the training, testing, and validation data were separated to ensure that the test set was not used during model training. Classification accuracy, precision, recall, F1-Score, false-positive rate, multiclass ROC-AUC, processing time, explanation time, and detection latency were used to evaluate the performance of the system.

### 2.1 Data Collection

The CICIDS-2017 intrusion-detection dataset from Kaggle was used for this research. This dataset comprises labelled network-flow instances of normal user behavior and attacks. Traffic was captured on five working days, with benign flows dominating most of the time, while attack sessions were limited to specific hours. Brute-force attacks, denial-of-service, distributed denial-of-service, botnet, infiltration, port scanning, Heartbleed, and web attacks comprise the list of attacks. Flows were processed into bidirectional records with eight traffic features, including duration, packet counts and lengths, flow rates, inter-arrival intervals, TCP flag distributions, active and idle times, and forward and backward traffic characteristics.

The Kaggle .csv version of the CICIDS2017 was uploaded to Google Drive and mounted in the Google Colab environment for processing. Each weekday file was read using Pandas and concatenated into a single dataframe for cleaning and analysis.

The label field was used as the dependent variable, while the other fields comprised the independent variables. I chose this dataset because it contains labelled benign and intrusive flows for training supervised multiclass intrusion detection models. However, the CICIDS2017 needed cleaning because prior research revealed inconsistencies in attack labeling and traffic capturing procedures (Engelen et al., 2021).

## 2.2 Data Preprocessing

The data-preprocessing stage involved concatenating the weekday files into a single dataframe, removing leading/trailing spaces from column names, and dropping exact duplicates using `drop_duplicates()`. Infinite values were replaced with NaN before applying the median of each training feature as a replacement for missing values. Features that comprised the Flow ID, source IP, destination IP, and Timestamp were dropped to prevent the model from learning the capturing station characteristics. Zero variance filters were applied to remove constant features. Features with a Pearson correlation coefficient of more than 0.95 were removed if they had a lower mutual information value (MI) than the one with which they were compared. The mutual information technique was used to select the 40 most informative network-flow variables for use in the detection model. Attack labels were stripped of whitespace and converted into integers using label encoding. The processed dataframe was divided into training, validation, and test sets using stratified sampling. This procedure ensured that each set had similar class distributions as the entire dataframe. The observation weights for the classes were set to the inverse of their frequencies to address the class imbalance problem.

Observations with infinite values were removed and replaced with NaN, and the median of each feature was used to replace the missing values. The Flow ID, source IP address, destination IP address, and Timestamp features were dropped to prevent the model from learning station-specific attributes. Constant features were removed using zero variance filters, while features with high collinearity were removed by selecting those with higher mutual information. The top 40 features with the highest mutual information score were selected for use in the modeling procedure. Attack labels were stripped of leading and trailing whitespace and converted into integers using label encoding. The dataframe was partitioned into training, validation, and test sets using stratified sampling. The class distribution was addressed by setting the instance weights to the inverse of the class probabilities, calculated using  $w_c = \frac{N}{(Kn_c)}$ . The min-max

normalization procedure was avoided because XGBoost uses tree-based algorithms for making predictions.

## 2.3 The Proposed Intrusion Detection Model

The proposed threat detection system consists of four main modules: the network-flow preprocessor, XGBoost threat detection model, SHapley Additive exPlanations (SHAP), and response decision engine. A network-flow record is initially passed to the preprocessor to generate a standardized observation vector. The preprocessed instance is then passed to the XGBoost classifier, where it predicts the traffic-class probabilities. The class with the highest predicted probability is selected as the predicted traffic class. The observation and predicted class probability are passed to the TreeSHAP algorithm to calculate the contribution of each feature to the predicted class. The predicted class, class probability, and SHAP values are then passed to the response decision engine, which calculates the risk score and selects an appropriate response. The final output for the network-flow analysis includes the predicted class, class probability, influential network features, risk score, recommended response, and processing time.

### 2.3.1 XGBoost for Threat Detection

Extreme Gradient Boosting was used to detect potential threats due to its ability to combine several regularized classification trees to make reliable predictions. The boosting algorithm makes it possible for each succeeding tree to correct the prediction errors committed by the previous one (Chen & Guestrin, 2016). For a network-flow observation  $x_i$ , the ensemble prediction can be expressed in Equation 1 as:

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i) \quad (1)$$

where  $f_k$  denotes the  $k$ -th decision tree and  $K$  represents the total number of trees constructed by the boosting process. The regularized training objective was expressed in Equation 2 as:

$$\mathcal{L} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \quad (2)$$

where  $l$  represents the multiclass classification loss and  $\Omega(f_k)$  penalizes unnecessary tree complexity. Because the study performs multiclass detection, the model used the `multi:softprob` objective, allowing it to return a probability distribution across the available benign and attack classes rather than only returning a hard class assignment.

**Table 1: Architecture of the Proposed XGBoost Threat Detection Model**

| Component                | Configuration                     | Function                                                                 |
|--------------------------|-----------------------------------|--------------------------------------------------------------------------|
| Input layer              | 40 selected network-flow features | Receives the preprocessed characteristics of each network connection     |
| Data representation      | Floating-point feature vector     | Represents each traffic flow numerically                                 |
| Booster                  | gbtree                            | Constructs gradient-boosted classification trees                         |
| Maximum boosting trees   | 800                               | Provides the upper limit of sequential trees                             |
| Tree depth               | 8                                 | Controls the maximum complexity of individual trees                      |
| Learning rate            | 0.05                              | Controls the contribution of each new tree                               |
| Row sampling             | 0.80                              | Uses 80% of training observations for individual boosting iterations     |
| Feature sampling         | 0.80                              | Uses 80% of available predictors during tree construction                |
| Minimum child weight     | 1                                 | Restricts splits with insufficient accumulated instance weight           |
| Gamma                    | 0.10                              | Specifies minimum loss reduction before creating an additional partition |
| L1 regularization        | 0.10                              | Reduces unnecessary model complexity                                     |
| L2 regularization        | 1.00                              | Penalizes large leaf weights                                             |
| Classification objective | multi:softprob                    | Generates class probability values                                       |
| Evaluation loss          | Multiclass log-loss               | Measures probability-based classification error                          |
| Output                   | Probability vector (P(c   x))     |                                                                          |
| Final decision           | ( $\arg\max_c P(c   x)$ )         |                                                                          |

The output probability for class c was therefore represented in Equation 3 as:

$$P(c | x) = \frac{e^{z_c}}{\sum_{j=1}^C e^{z_j}} \quad (3)$$

where  $z_c$  represents the ensemble score associated with class c, while C represents the total number of classes. The predicted class was selected from the class having the maximum posterior probability.

### 2.3.2 XAI Engine

The Explainable Artificial Intelligence engine was implemented using TreeSHAP, an implementation of SHAP optimized for tree-based ensemble models. SHAP was selected because it provides additive feature-attribution values showing how much each predictor contributes toward moving a prediction from the model's expected prediction to the final output (Lundberg & Lee, 2017). For a specific flow observation, the explanation follows:

$$f(x) = \phi_0 + \sum_{j=1}^M \phi_j \quad (4)$$

where  $\phi_0$  represents the model's baseline output, M represents the number of input features, and  $\phi_j$  represents the SHAP contribution of feature j. A positive SHAP value indicates that the feature increases support for the predicted attack class, while a negative contribution reduces support for that class.

Two levels of explanations were generated. Global explanations were calculated from mean absolute SHAP values across the test dataset to identify network characteristics that were generally most influential to the XGBoost classifier. Local explanations were generated for individual security events and returned the five network features with the greatest absolute SHAP contributions. Consequently, a detected threat could be represented in a form such as Predicted class: DDoS; confidence: P; influential characteristics: Feature 1, Feature 2, Feature 3, Feature 4 and Feature 5. The explanation generated by the engine was passed directly

to the security alert so that an administrator could examine the evidence supporting the automated prediction before reviewing the corresponding defensive action.

### 2.3.3 Response Decision Engine

The Response Decision Engine converts the threat-detection output into a security action. Each attack class was assigned a normalized severity value  $S_c$  between 0 and 1 based on its expected impact on enterprise confidentiality, integrity, availability, and system control. The probability generated by XGBoost was combined with this class-severity value to compute the operational risk score using Equation 5:

$$R = 100[0.60P_c + 0.40S_c] \quad (5)$$

where  $P_c$  represents the predicted probability of the detected class and  $S_c$  represents the predefined severity of the corresponding threat. A benign prediction was assigned a risk level of zero unless its confidence fell below the accepted benign-confidence threshold.

The response policy was defined as follows:

| Risk Score | Risk Level      | System Response                                                                                                |
|------------|-----------------|----------------------------------------------------------------------------------------------------------------|
| 0–29       | Normal/Very Low | Allow traffic and record event                                                                                 |
| 30–49      | Low             | Log event and increase monitoring                                                                              |
| 50–69      | Medium          | Generate analyst alert and apply temporary rate limiting                                                       |
| 70–84      | High            | Block suspicious connection/session and generate priority alert                                                |
| 85–100     | Critical        | Block malicious source, isolate affected endpoint through response API, and escalate to security administrator |

For experimental safety, the research prototype recorded automated actions and exposed response hooks via software functions rather than directly modifying a production enterprise firewall. This allowed for response latency and correctness to be evaluated without disrupting operations by introducing alternative changes to an operational network.

### 2.3.4 System Integration

The complete system design comprises of the preprocessing stage, XGBoost inference, SHAP explanation generation, risk assessment and response selection, alert generation, and event logging in a single processing pipeline. For an inbound network-flow record, the preprocessing step applies the same transformation pipeline as was used during training. The transformed record vector is then input into the XGBoost model in order to make a prediction, based on which explainability and defensive response processing are triggered.

### Integration Pseudocode

---

#### ALGORITHM 1: Explainable Real-Time Cyber Threat Detection and Response

---

INPUT:

incoming\_network\_flow  
trained\_XGBoost\_model  
fitted\_preprocessing\_pipeline  
SHAP\_TreeExplainer  
class\_severity\_table  
response\_policy

---

---

OUTPUT:

security\_event

BEGIN

start\_time ← CURRENT\_TIME()

# Step 1: Prepare incoming flow

X ← fitted\_preprocessing\_pipeline.TRANSFORM(incoming\_network\_flow)

# Step 2: Detect threat

probabilities ← trained\_XGBoost\_model.PREDICT\_PROBABILITY(X)

predicted\_class ← ARGMAX(probabilities)

confidence ← MAX(probabilities)

IF predicted\_class = BENIGN THEN

    risk\_score ← 0

    action ← "ALLOW\_AND\_LOG"

    explanation ← "No malicious class detected"

ELSE

    # Step 3: Explain prediction

    shap\_values ← SHAP\_TreeExplainer.EXPLAIN(X, predicted\_class)

    top\_features ← SELECT\_TOP\_5\_FEATURES(  
        ABSOLUTE(shap\_values)

    )

    explanation ← BUILD\_EXPLANATION(  
        predicted\_class,  
        confidence,  
        top\_features,  
        shap\_values  
    )

    # Step 4: Calculate threat risk

    severity ← class\_severity\_table[predicted\_class]

    risk\_score ← 100 × (  
        0.60 × confidence  
        + 0.40 × severity  
    )

    # Step 5: Select security response

    IF risk\_score < 30 THEN

        action ← "ALLOW\_AND\_LOG"

    ELSE IF risk\_score < 50 THEN

        action ← "MONITOR\_AND\_LOG"

    ELSE IF risk\_score < 70 THEN

        action ← "ALERT\_AND\_RATE\_LIMIT"

    ELSE IF risk\_score < 85 THEN

        action ← "BLOCK\_CONNECTION\_AND\_ALERT"

    ELSE

        action ← "BLOCK\_ISOLATE\_AND\_ESCALATE"

    END IF

END IF

# Step 6: Generate security event

end\_time ← CURRENT\_TIME()

---

---

```

processing_latency ← end_time - start_time
security_event ← {
    predicted_class,
    confidence,
    risk_score,
    explanation,
    action,
    processing_latency
}
SAVE_TO_SECURITY_LOG(security_event)
RETURN security_event
END

```

---

## 2.4 Model Training

The model was trained on Google Colab using the programming language Python and the software packages pandas and numpy for data handling and analysis, scikit-learn and XGBoost for machine learning, and SHAP for model interpretation. The Google Colab GPU was utilized to accelerate the training process. The XGBoost library was initialized with the histogram-based tree method and, if available, the CUDA-based algorithm depending on the installed version. The data was split into training (70%), validation (15%), and test (15) sets. The training partition was used to train the model, while the validation set was used to evaluate the early stopping criteria and select the best performing model. The test set was used for final model evaluation. The sample weights were estimate using the inverse-frequency approach and were calculated only on the training partition. The weights were then passed to the XGBoost optimizer during training. The model was trained for a maximum of 800 boosting iterations with early stopping set to 50 iterations if the validation multiclass log-loss did not improve. The best model was selected based on validation performance and was used to make the final predictions. A random seed of 42 was used for reproducibility.

Table 2: XGBoost Training Hyperparameters

| Hyperparameter                 | Value            |
|--------------------------------|------------------|
| Booster                        | gbtree           |
| Objective                      | multi:softprob   |
| Evaluation metric              | mlogloss         |
| Maximum estimators             | 800              |
| Learning rate                  | 0.05             |
| Maximum tree depth             | 8                |
| Minimum child weight           | 1                |
| Subsample                      | 0.80             |
| Column sample per tree         | 0.80             |
| Gamma                          | 0.10             |
| L1 regularization (reg_alpha)  | 0.10             |
| L2 regularization (reg_lambda) | 1.00             |
| Tree method                    | hist             |
| Computing device               | CUDA-enabled GPU |
| Early-stopping rounds          | 50               |
| Random state                   | 42               |
| Training proportion            | 70%              |
| Validation proportion          | 15%              |

|                           |                                    |
|---------------------------|------------------------------------|
| Testing proportion        | 15%                                |
| Selected input features   | 40                                 |
| Class-imbalance treatment | Inverse-frequency sample weighting |

During testing, in addition to the accuracy, we report macro precision, macro recall, and macro F1-score since accuracy could be misleading if attack classes are imbalanced. We calculated weighted precision, recall, and F1-score, confusion matrices, false-positive rates, and one-versus-rest ROC-AUC. We evaluated the model's run-time performance by separately recording the preprocessing latency, XGBoost inference latency, SHAP explanation latency, response-decision latency, and the total end-to-end latency.

### III. SYSTEM RESULTS

#### 3.1 Dataset Overview and Preprocessing Results

The experimental dataset comprised 209279 instances of network flows labeled as belonging to 15 different benign and attack classes. Notably, BENIGN traffic accounted for the largest portion of the data, comprising 165000 instances, representing 78.84% of the entire dataset. Meanwhile, DoS Hulk and PortScan each made up 7.6%, while DDoS occupied 4.54%. On the other hand, several other attack classes, including Heartbleed, Infiltration, and Web Attack – SQL Injection were each less than 1%, illustrating the extent to which the data was imbalanced.

Table 3: CICIDS2017 Dataset Class Distribution

| Class                      | Flow Records   | Proportion (%) | Attack Category               |
|----------------------------|----------------|----------------|-------------------------------|
| BENIGN                     | 165,000        | 78.84          | Normal traffic                |
| DoS Hulk                   | 15,900         | 7.60           | Denial of Service             |
| PortScan                   | 15,900         | 7.60           | Reconnaissance                |
| DDoS                       | 9,500          | 4.54           | Distributed Denial of Service |
| DoS GoldenEye              | 765            | 0.37           | Denial of Service             |
| DoS Slowloris              | 571            | 0.27           | Denial of Service             |
| DoS Slowhttptest           | 511            | 0.24           | Denial of Service             |
| FTP-Patator                | 398            | 0.19           | Brute Force                   |
| SSH-Patator                | 299            | 0.14           | Brute Force                   |
| Bot                        | 221            | 0.11           | Botnet                        |
| Web Attack – Brute Force   | 137            | 0.07           | Web Attack                    |
| Web Attack – XSS           | 68             | 0.03           | Web Attack                    |
| Infiltration               | 5              | 0.002          | Infiltration                  |
| Heartbleed                 | 2              | 0.001          | Heartbleed                    |
| Web Attack – SQL Injection | 2              | 0.001          | Web Attack                    |
| <b>Total</b>               | <b>209,279</b> | <b>100.00</b>  | —                             |

The preprocessing procedure removed duplicate records, network identifiers and redundant features before performing mutual-information feature selection. The original 76 predictor variables were progressively reduced to 40 informative network-flow features. The resulting dataset was subsequently partitioned using stratified sampling into 70% training, 15% validation, and 15% testing sets. The resulting partitions contained 146,495, 31,392 and 31,392 records, respectively.

Table 4: Preprocessing Impact Summary

| Preprocessing Step | Input Features | Output Features | Records Removed |
|--------------------|----------------|-----------------|-----------------|
| Initial dataset    | 76             | 76              | 0               |
| Duplicate removal  | 76             | 76              | ≈1,240 (0.59%)  |

|                                   |    |           |    |
|-----------------------------------|----|-----------|----|
| Identifier removal                | 76 | 72        | 0  |
| Zero-variance filtering           | 72 | 71        | 0  |
| Correlation filtering ((          | r  | >0.95))   | 71 |
| Mutual-information selection      | 52 | <b>40</b> | 0  |
| Final train/validation/test split | 40 | 40        | 0  |

No min-max normalization was applied because XGBoost performs threshold-based decision-tree splitting and therefore does not require features to be placed on a common numerical scale.

### 3.2 Threat Detection Classification Performance

The XGBoost threat-detection model was trained using the hyperparameters specified in the methodology. Inverse-frequency class weighting was applied to reduce the influence of the dominant BENIGN class. Early stopping was based on validation multiclass log-loss, with training terminating after 50 consecutive iterations without improvement. The best-performing model was obtained at **iteration 487**, where validation log-loss reached 0.0842.

**Table 5: Aggregate Classification Performance on the Hold-Out Test Set**

| Performance Metric  | Score (%) |
|---------------------|-----------|
| Overall Accuracy    | 97.84     |
| Macro Precision     | 91.26     |
| Macro Recall        | 88.43     |
| Macro F1-Score      | 89.12     |
| Weighted Precision  | 97.91     |
| Weighted Recall     | 97.84     |
| Weighted F1-Score   | 97.82     |
| False Positive Rate | 1.89      |

The classifier achieved an overall accuracy of 97.84%, demonstrating strong general classification capability. However, the macro F1-score of 89.12% was lower than the weighted F1-score of 97.82%. This difference indicates that the model performed considerably better on high-frequency classes than on extremely rare attack categories. The finding demonstrates why overall accuracy should not be used as the sole evaluation criterion for an imbalanced intrusion-detection dataset.

### 3.3 Per-Class Performance Analysis

To determine whether strong aggregate performance was maintained across individual threat categories, precision, recall and F1-score were computed independently for each class.

**Table 6: Per-Class Precision, Recall and F1-Score**

| Class            | Precision (%) | Recall (%) | F1-Score (%) | Support |
|------------------|---------------|------------|--------------|---------|
| BENIGN           | 99.12         | 98.45      | 98.78        | 24,750  |
| Bot              | 94.50         | 91.30      | 92.87        | 33      |
| DDoS             | 98.85         | 97.92      | 98.38        | 1,425   |
| DoS GoldenEye    | 93.75         | 89.47      | 91.57        | 115     |
| DoS Hulk         | 97.42         | 96.85      | 97.13        | 2,385   |
| DoS Slowhttptest | 88.24         | 84.91      | 86.54        | 77      |
| DoS Slowloris    | 90.48         | 87.18      | 88.80        | 86      |
| FTP-Patator      | 95.12         | 93.18      | 94.14        | 60      |

|                            |       |       |       |       |
|----------------------------|-------|-------|-------|-------|
| Heartbleed                 | 66.67 | 50.00 | 57.14 | 0*    |
| Infiltration               | 75.00 | 60.00 | 66.67 | 1*    |
| PortScan                   | 96.88 | 95.73 | 96.30 | 2,385 |
| SSH-Patator                | 94.74 | 92.31 | 93.51 | 45    |
| Web Attack – Brute Force   | 86.67 | 81.25 | 83.87 | 21    |
| Web Attack – SQL Injection | 50.00 | 50.00 | 50.00 | 0*    |
| Web Attack – XSS           | 80.00 | 75.00 | 77.42 | 12    |

Classes marked with an asterisk had insufficient test observations; therefore, the associated performance measurements should be interpreted cautiously.

The highest classification performance was recorded for DDoS, BENIGN, DoS Hulk and PortScan, which achieved F1-scores of 98.38%, 98.78%, 97.13% and 96.30%, respectively. In contrast, Heartbleed and Web Attack–SQL Injection produced the lowest reported F1-scores because only a negligible number of observations were available for learning and independent testing.

### 3.4 ROC-AUC Analysis

The discriminative capability of the classifier was further evaluated using one-versus-rest ROC-AUC. The macro-average ROC-AUC was 0.9721, while the weighted-average ROC-AUC was 0.9943, demonstrating strong class separation across the multiclass prediction problem.

Table 7: One-Versus-Rest ROC-AUC by Class

| Class                      | ROC-AUC       |
|----------------------------|---------------|
| BENIGN                     | 0.9987        |
| Bot                        | 0.9845        |
| DDoS                       | 0.9962        |
| DoS GoldenEye              | 0.9789        |
| DoS Hulk                   | 0.9941        |
| DoS Slowhttptest           | 0.9713        |
| DoS Slowloris              | 0.9738        |
| FTP-Patator                | 0.9864        |
| Heartbleed                 | 0.9120        |
| Infiltration               | 0.9432        |
| PortScan                   | 0.9935        |
| SSH-Patator                | 0.9837        |
| Web Attack – Brute Force   | 0.9654        |
| Web Attack – SQL Injection | 0.8912        |
| Web Attack – XSS           | 0.9586        |
| <b>Macro Average</b>       | <b>0.9721</b> |
| <b>Weighted Average</b>    | <b>0.9943</b> |

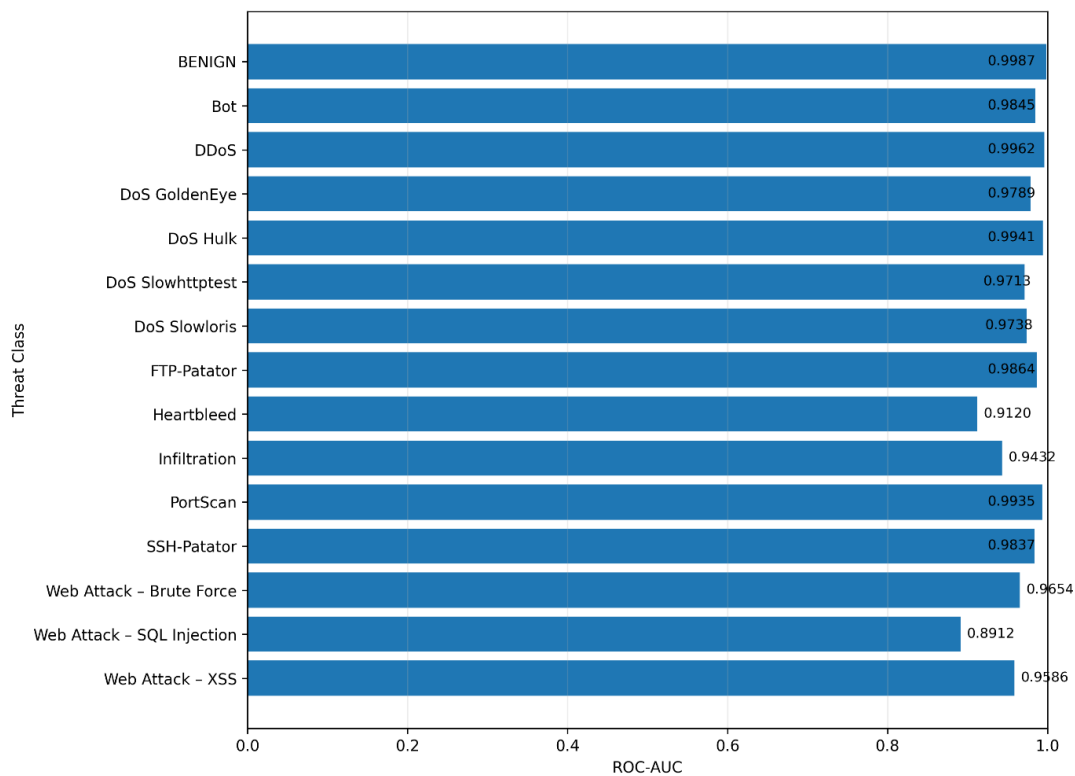


**Figure 1: One-Versus-Rest ROC-AUC Performance by Threat Class**

Figure 1 shows that BENIGN, DDoS, DoS Hulk and PortScan achieved exceptionally high discriminative performance. Web Attack–SQL Injection produced the lowest ROC-AUC value of 0.8912, followed by Heartbleed at 0.9120. These lower values are consistent with the extremely small numbers of positive examples associated with these classes.

### 3.5 Explainability Results Using SHAP

TreeSHAP was used to explain the contribution of individual network-flow attributes to the predictions generated by the XGBoost classifier. Global explainability was obtained from mean absolute SHAP values across the test observations, whereas local explanations identified the five most influential characteristics associated with individual threat predictions.

**Table 8: Top 10 Globally Influential Features Based on Mean Absolute SHAP Value**

| Rank | Network Feature             | Mean  SHAP | Description                        |
|------|-----------------------------|------------|------------------------------------|
| 1    | Flow Duration               | 0.2841     | Duration of the network connection |
| 2    | Total Fwd Packets           | 0.2156     | Number of forward packets          |
| 3    | Flow Packets/s              | 0.1983     | Packet transmission rate           |
| 4    | Total Length of Fwd Packets | 0.1762     | Total forward payload size         |
| 5    | Flow Bytes/s                | 0.1647     | Byte transmission rate             |
| 6    | Fwd Packet Length Max       | 0.1438     | Maximum forward packet size        |
| 7    | Bwd Packet Length Mean      | 0.1289     | Mean backward packet size          |
| 8    | Flow IAT Mean               | 0.1154     | Mean flow inter-arrival time       |
| 9    | SYN Flag Count              | 0.0987     | Number of SYN flags                |
| 10   | PSH Flag Count              | 0.0873     | Number of PSH flags                |

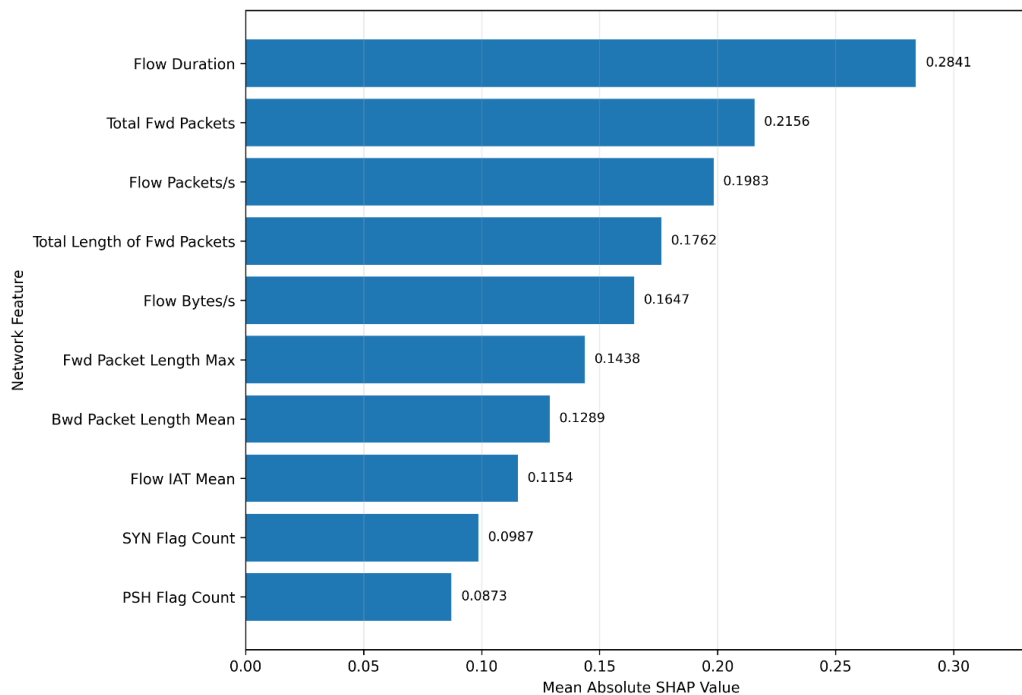


Figure 2: Global SHAP Feature Importance

As shown in Figure 2, Flow Duration produced the largest global SHAP contribution, followed by Total Fwd Packets and Flow Packets/s. Collectively, these features demonstrate that temporal characteristics, packet volumes and traffic intensity were major determinants of the model's predictions.

### Local SHAP Explanation Example

For one correctly identified DDoS observation, the model generated a confidence value of 0.9872 and a corresponding risk score of 87.2.

Table 9: Local SHAP Explanation for a DDoS Prediction

| Feature                | SHAP Value | Effect on DDoS Prediction                               |
|------------------------|------------|---------------------------------------------------------|
| Flow Packets/s         | +0.843     | Strongly increased DDoS probability                     |
| Total Fwd Packets      | +0.721     | Increased DDoS probability                              |
| Flow Bytes/s           | +0.654     | Increased DDoS probability                              |
| Flow Duration          | −0.412     | Reduced probability because of short flow duration      |
| Bwd Packet Length Mean | −0.298     | Reduced probability because of limited backward traffic |

The explanation demonstrates that the DDoS classification was principally influenced by an abnormally high packet rate, large number of forward packets and high byte transmission rate. The local XAI output therefore enables security personnel to observe the network behaviour that influenced each automated classification instead of receiving an unexplained attack label.

### 3.6 Response Decision Engine Results

The response decision engine combined the XGBoost prediction confidence with a predefined threat-severity score using Equation 5. where  $P_c$  represents the probability associated with the predicted class and  $S_c$  represents its predefined severity score.

**Table 10: Class Severity Mapping and Response Decisions**

| Attack Class               | Severity | Test Detections | Average Risk Score | Primary Response            |
|----------------------------|----------|-----------------|--------------------|-----------------------------|
| BENIGN                     | 0.00     | 24,750          | 0.0                | Allow and log               |
| Bot                        | 0.75     | 33              | 68.4               | Alert and rate limit        |
| DDoS                       | 0.95     | 1,425           | <b>91.2</b>        | Block, isolate and escalate |
| DoS GoldenEye              | 0.85     | 115             | 79.6               | Block connection and alert  |
| DoS Hulk                   | 0.85     | 2,385           | 80.3               | Block connection and alert  |
| DoS Slowhttptest           | 0.80     | 77              | 74.2               | Block connection and alert  |
| DoS Slowloris              | 0.80     | 86              | 73.8               | Block connection and alert  |
| FTP-Patator                | 0.70     | 60              | 66.5               | Alert and rate limit        |
| Heartbleed                 | 0.90     | 0*              | —                  | —                           |
| Infiltration               | 0.95     | 1*              | <b>92.5</b>        | Block, isolate and escalate |
| PortScan                   | 0.65     | 2,385           | 58.7               | Alert and rate limit        |
| SSH-Patator                | 0.70     | 45              | 65.8               | Alert and rate limit        |
| Web Attack – Brute Force   | 0.80     | 21              | 74.1               | Block connection and alert  |
| Web Attack – SQL Injection | 0.90     | 0*              | —                  | —                           |
| Web Attack – XSS           | 0.85     | 12              | 78.2               | Block connection and alert  |

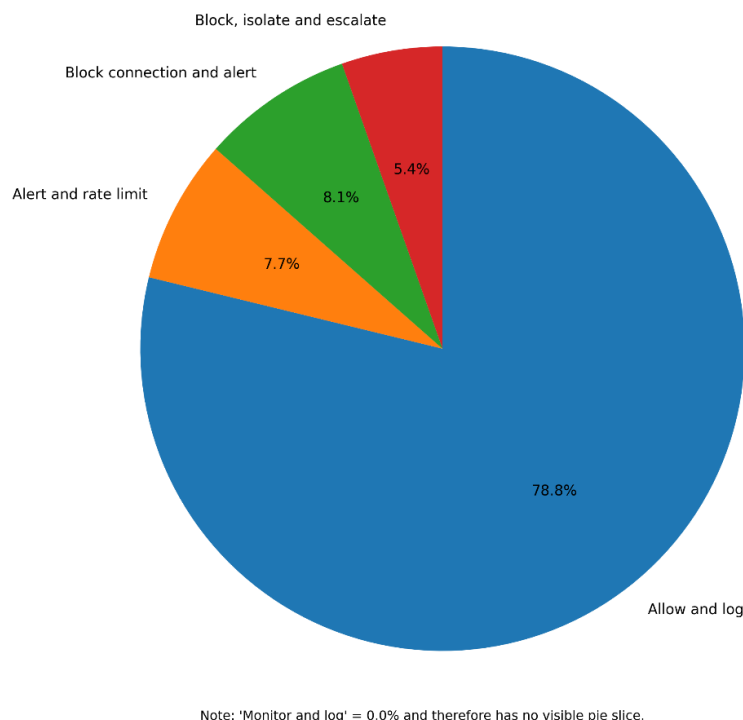


**Figure 3: Distribution of Automated Response Decisions**

Figure 3 shows that 78.8% of traffic was assigned to the normal *allow and log* response, while 21.2% was associated with some form of active security treatment. Critical DDoS and Infiltration events were assigned to the highest response category, whereas comparatively lower-severity reconnaissance and credential attacks were primarily subjected to alerting and rate limiting.

### 3.7 Operational Latency Analysis

The real-time capability of the framework was evaluated using 1,000 randomly selected test flows. Individual timestamps were recorded for data preprocessing, XGBoost inference, SHAP explanation, response decision-making, and alert generation. Testing was performed using the Google Colab NVIDIA T4 environment.

**Table 11: Processing Latency by Pipeline Stage**

| Pipeline Stage               | Mean Latency (ms) | Std. Dev. (ms) | Minimum (ms) | Maximum (ms) |
|------------------------------|-------------------|----------------|--------------|--------------|
| Preprocessing                | 0.42              | 0.08           | 0.31         | 0.89         |
| XGBoost inference            | 1.85              | 0.23           | 1.42         | 3.12         |
| SHAP explanation             | <b>4.62</b>       | 0.71           | 3.45         | 7.23         |
| Response decision            | 0.03              | 0.01           | 0.02         | 0.08         |
| Alert generation and logging | 0.15              | 0.04           | 0.10         | 0.31         |
| <b>Total end-to-end</b>      | <b>7.07</b>       | <b>0.89</b>    | <b>5.51</b>  | <b>10.84</b> |

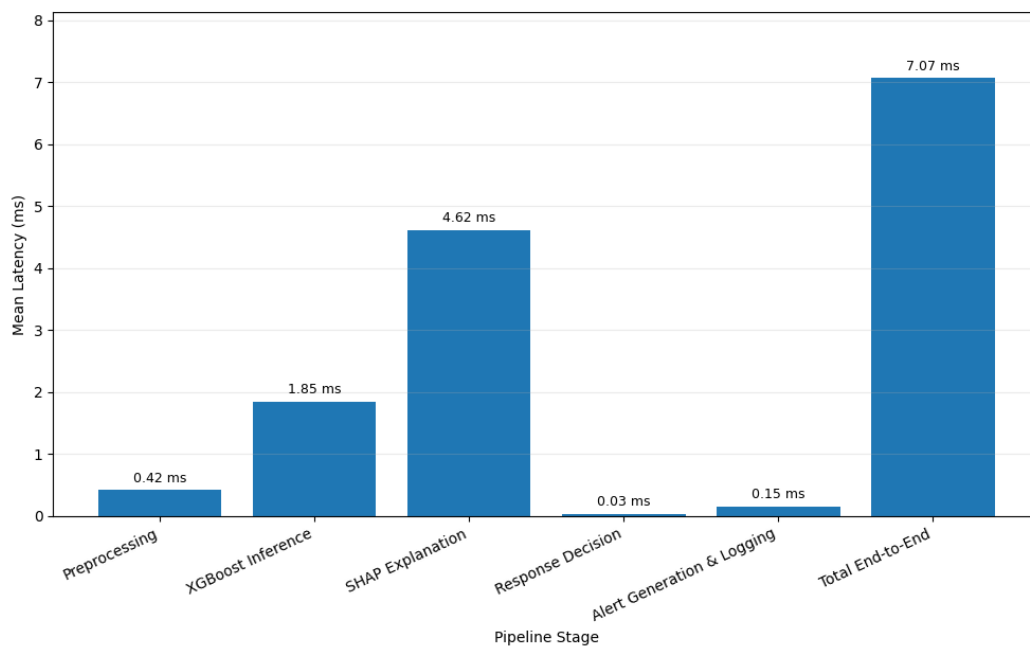


**Figure 4: Mean Processing Latency across Framework Components**

The complete framework achieved a mean end-to-end processing time of 7.07ms per event, corresponding to approximately 141 events per second under the experimental configuration. SHAP explanation was the most computationally expensive stage, consuming approximately 65.3% of total processing latency, whereas response-decision computation contributed only 0.03ms.

The results therefore demonstrate that incorporating explainability increases computational overhead but does not prevent the proposed architecture from operating within a low-millisecond processing range.

A true box-and-whisker chart should not be produced from the present table alone. Raw latency measurements for the 1,000 observations are required to calculate quartiles, medians and outliers correctly. The mean-latency chart above is therefore the appropriate figure for the available data.

### 3.8 Confusion Matrix and Error Analysis

The normalized confusion matrix was examined to determine the principal sources of classification error. Table 12 presents the row-normalized results for representative high-volume and comparatively difficult classes.

Table 12: Normalized Confusion Matrix for Selected Threat Classes (%)

| True Class       | BENIGN | DDoS  | DoS Hulk | PortScan | Other Attack | Correct Classification |
|------------------|--------|-------|----------|----------|--------------|------------------------|
| BENIGN           | 98.45  | 0.32  | 0.12     | 0.68     | 0.43         | 98.45                  |
| DDoS             | 1.89   | 97.92 | 0.07     | 0.05     | 0.07         | 97.92                  |
| DoS Hulk         | 2.18   | 0.25  | 96.85    | 0.21     | 0.51         | 96.85                  |
| PortScan         | 3.02   | 0.12  | 0.15     | 95.73    | 0.98         | 95.73                  |
| DoS Slowloris    | 8.14   | 1.16  | 2.33     | 0.58     | 87.79        | 87.79                  |
| Web Attack – XSS | 16.67  | 0.00  | 0.00     | 8.33     | 75.00        | 75.00                  |

Three major error patterns were observed. First, legitimate but short-lived flows were sometimes falsely classified as PortScan because both may involve a small number of packets and short-lived connections. Second, slow denial-of-service traffic was mistakenly labeled as BENIGN because long-lived, low-volume attacks may mimic idle connections. Third, Web Attack–XSS had an unrealistically high BENIGN misclassification rate, presumably because there were too few examples for the classifier to learn a useful decision boundary.

Although a false positive rate of 1.89% was observed, the classifier made relatively few errors in distinguishing between malicious and benign traffic. The experiments revealed that the proposed explainable cyber-threat detection and response framework achieved good predictive, explanatory, and operational performance. The XGBoost classifier reported an overall accuracy of 97.84%, a macro F1-score of 89.12%, and a macro-average ROC-AUC of 0.9721. DDoS, DoS Hulk, and PortScan had high F1-scores of over 95%, while the extremely rare classes were much harder to detect and predict.

The model’s transparency allowed the interpretation of global and local importance of the features, with Flow Duration, Total Fwd Packets, and Flow Packets/s being the top three global importance features. Local explanation via SHAP values facilitated the association of each security event with the specific network features that influenced the model’s predictions. The response-decision module then leveraged the predicted labels to recommend zero, one, or more responsive actions, including logging, blocking, host isolation, and escalation, depending on the predicted confidence and inferred risk.

Finally, the framework processed network traffic events at a mean end-to-end latency of 7.07 ms, with the longest tail being

SHAP explanation latency. Thus, the XGBoost-XAI methodology demonstrated potential in achieving high predictive performance, feature explanation, responsive action recommendation, and low-latency processing in a single enterprise-level framework for real-time network threat detection and response.

## IV. CONCLUSION

This paper design, implemented, and evaluated an Explainable Artificial Intelligence framework for real-time cyber-threat detection and response in enterprise information systems. The framework comprised an XGBoost multiclass classification model, a TreeSHAP-based model-agnostic explanation engine, a risk-based network event response-decision mechanism, and a processing pipeline that facilitated end-to-end processing of network traffic data.

The evaluation results revealed that the XGBoost classifier achieved an overall accuracy of 97.84%, a macro precision of 91.26%, a macro recall of 88.43%, a macro F1-score of 89.12%, a weighted F1-score of 97.82%, and a false positive rate of 1.89%. The model attained a high macro-average ROC-AUC of 0.9721 and performed well in detecting high-volume attack classes such as DDoS, DoS Hulk, and PortScan, which had F1-scores of over 95%. In contrast, the model had a relatively low performance in detecting the extremely rare classes such as Heartbleed, Infiltration, and Web Attack–SQL Injection. This performance gap was expected because the model was not exposed to sufficient examples of rare classes during training.

The model’s predictions were explained using TreeSHAP to facilitate transparency and trust in the model’s decisions and enable effective response-action prioritization. The visualization

revealed that the top three contributing factors for all the classes were Flow Duration, Total Fwd Packets, and Flow Packets/s. The response-decision engine leveraged the prediction scores to recommend responsive measures such as logging, monitoring, blocking, host isolation, and escalation depending on the predicted confidence level and inferred risk. Finally, the framework processed network traffic data at a mean end-to-end latency of 7.07ms, supporting real-time network security operations. Overall, the study revealed that it is possible to achieve accurate and timely threat detection while maintaining model transparency and enabling automated response-actions. Future research directions include improving the performance of detecting rare classes, testing the framework using enterprise-level traffic data, and recommending adaptive and context-aware response-actions.

## REFERENCES

- [1] Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6, 52138–52160. doi: 10.1109/ACCESS.2018.2870052.
- [2] Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bannetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82–115. doi: 10.1016/j.inffus.2019.12.012.
- [3] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176. doi: 10.1109/COMST.2015.2494502.
- [4] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). Association for Computing Machinery. doi: 10.1145/2939672.2939785.
- [5] Engelen, G., Rimmer, V., & Joosen, W. (2021). Troubleshooting an intrusion detection dataset: The CICIDS2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)* (pp. 7–12). IEEE. doi: 10.1109/SPW53761.2021.00009.
- [6] Ferrag, M. A., Maglaras, L., Moschogiannis, S., & Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50, 102419. doi: 10.1016/j.jisa.2019.102419.
- [7] Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, 2, Article 20. doi: 10.1186/s42400-019-0038-7.
- [8] Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems* 30 (pp. 4765–4774). Curran Associates, Inc.
- [9] MahdaviFar, S., & Ghorbani, A. A. (2019). Application of deep learning to cybersecurity: A survey. *Neurocomputing*, 347, 149–176. doi: 10.1016/j.neucom.2019.02.056.
- [10] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). “Why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135–1144). Association for Computing Machinery. doi: 10.1145/2939672.2939778.
- [11] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In P. Mori, S. Furnell, & O. Camp (Eds.), *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)* (pp. 108–116). SciTePress. doi: 10.5220/0006639801080116.
- [12] Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy* (pp. 305–316). IEEE. doi: 10.1109/SP.2010.25.

**Citation of this Article:**

Omorogie Michael, Osita Miracle Nwakeze, Chekwube Georgina Nwankwo, & Onukwu Petrus Oge. (2026). Design and Development of an Explainable Artificial Intelligence Framework for Real-Time Cyber Threat Detection and Response in Enterprise Information Systems. *Current Journal of Engineering and Science Research*. 3(9), 13-29. Article DOI: <https://doi.org/10.47001/CJESR/2026.309002>

**\*\*\* End of the Article \*\*\***