International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

IoT malware detection using a novel 3-Sigma Auto-Funnel Transformer approach

Moushumi Barman
Department of Computer Science and Engineering
Assam Don Bosco University
Guwahati, Assam
moushumi77777@gmail.com

Bobby Sharma
Department of Computer Science and Engineering
Assam Don Bosco University
Guwahati, Assam
bobby.sharma@dbuniversity.ac.in

Abstract— The proliferation of Internet of Things (IoT) devices has ushered in a new era of connected technologies, but it has also brought significant security challenges, particularly in the area of malware detection. This research paper presents a novel approach, the “3 Sigma Auto Funnel Transformer,” that designed to address the specific complexities of malware detection in IoT devices. By leveraging advanced deep learning techniques and a multi-layered architecture, the proposed framework provides an innovative solution to detect and mitigate malware threats in IoT ecosystems. By combining the precision of the ”3 Sigma” approach with the efficiency of an ”Auto Funnel Transformer,” this architecture achieves superior detection accuracy and performance. Through comprehensive evaluations, this paper demonstrates the effectiveness of the proposed system in bolstering the security of IoT devices, thereby contributing to the ongoing efforts to protect these essential components of our interconnected world.

Keywords-Autoencoder; Funnel Transformer; Deep Learning; Auto-Funnel; 3-Sigma limit

I. INTRODUCTION

The task of network intrusion detection holds immense importance in the realm of modern cybersecurity as it endeavors to safeguard computer networks from illicit entry and malicious actions. In light of the growing intricacy and range of cyber threats, conventional rule-based and signature-based intrusion detection systems face noteworthy constraints when identifying emerging or unknown attacks. To confront this formidable challenge, scholars have begun exploring novel avenues that harness cutting-edge machine learning techniques, notably deep learning, with the aim of enhancing the efficacy of network intrusion detection methods [1, 2].

This research paper presents an innovative proposition that harmoniously blends the potency of funnel transformers and autoencoders to enhance network intrusion detection. Funnel transformers, distinguished for their sophisticated sequence modelling structures, exhibit remarkable proficiency in capturing prolonged dependencies and hierarchal patterns inherent in sequential data [3]. Conversely, autoencoders, a breed of unsupervised learning model, skillfully extract compressed representations of input data by astutely apprehending its fundamental traits. Their successful implementation across multiple domains for feature extraction and anomaly detection has been widely acknowledged [4].

The fusion of autoencoders and funnel transformers presents a promising solution for enhancing the capabilities of Intrusion Detection Systems (IDS) in identifying network intrusions. This proposed methodology unfolds in two distinct phases. Initially, the autoencoder model is employed to assimilate both macroscopic and microscopic connections within network traffic data, facilitating effective pattern recognition and detection of anomalous behavior. Subsequently, the encoded attributes are passed through the funnel transformer mechanism, allowing for the acquisition of a compressed representation of the data. This valuable process further enhances the IDS’s ability to detect potential infringements on a network by capturing even more subtle nuances inherent to malicious activities. In order to determine the efficacy of the suggested methodology, a series of extensive experiments have been conducted on a benchmark dataset containing various network traffic scenarios as well as instances of intrusion. To evaluate the detection capabilities of the model, performance metrics including accuracy, precision, recall, and F1 score have been utilized. Furthermore, a comparative analysis has also been undertaken with established intrusion detection techniques aiming to showcase the superiority of the proposed approach.

The contributions of this research are as follows:

- Introducing a novel approach that combines autoencoders and funnel transformers for network intrusion detection, leveraging the strengths of both models. Until now, the funnel transformer architecture has primarily found in the field of Natural Language Processing (NLP). However, this instance involves utilizing the funnel transformer on a network, which represents an innovative beyond its conventional use in NLP.

- Enhancing the detection capabilities of IDS by capturing global and local dependencies in network traffic data and learning compact representations of intrusions.
- Conducting comprehensive experiments on a benchmark dataset to evaluate the effectiveness and performance of the proposed approach.
- Performing comparative analysis with existing techniques to demonstrate the superiority of the proposed approach in terms of detection accuracy and efficiency.

However, precision and accuracy serve as measures of how well the work was done. Malware databases are actually skewed in reality. As a result, different evaluation indications need to be taken into account. For the suggested study, the benchmark dataset IoT-23 was employed in this context. Metrics for performance evaluation include the F1 score and recall. The entire workflow is displayed in Figure 1.

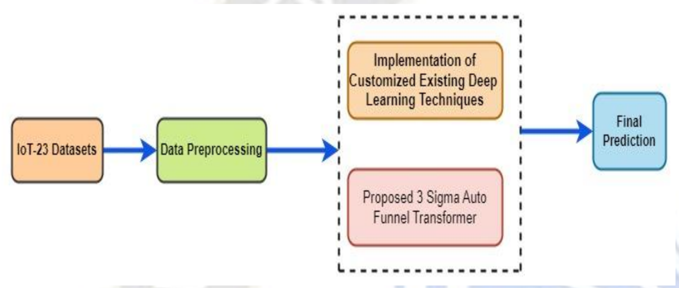

IoT-23 Datasets
Data Preprocessing
Implementation of Customized Existing Deep Learning Techniques
Proposed 3 Sigma Auto Funnel Transformer
Final Prediction

Figure 1. Proposed Framework overview

The rest of the paper is organized as follows: **Section II** provides an overview of related work in the field of network intrusion detection, highlighting the application of deep learning techniques. **Section III** presents **"3-Sigma AutoFunnel Transformer"** in details. **Section IV** describes the experimental setup, including the dataset, performance, and comparative analysis. **Section V** presents the results and discussion of the experiments. Finally, **Section VI** concludes the paper, summarizes the contributions, and outlines potential avenues for future research.

II. Background

In paper [3], the advanced sequence modeling topologies known as funnel transformers are introduced by the author. It describes how the use of funnel transformers, which are appropriate for network intrusion detection, allows them to extract hierarchical patterns and long-range dependencies from consecutive data.

In paper [5], the author offers two end-to-end deep learning-based techniques for identifying Android malware. It is proposed to feed raw byte-code from classes.dex files of Android applications to deep learning models. The dataset utilized to train and evaluate these models consist of 8,000 benign applications and 8,000 malicious applications. The proposed approaches, according to experiments, may reach detection accuracy of 93.4% and 95.8%, respectively.

Using the adopted strategy suggested in [6], the detecting module is divided into two tiers and achieves an outstanding accuracy of 98.29%. However, the principal impediment to this undertaking is the significant processing power needed to manage the two data layers. In paper [6], author employs a deep learning model that combines an autoencoder network with a grayscale picture representation of malware to discern between dangerous and benign software. Based on the autoencoder's reconstruction error, the utility of the greyscale image approach to malware is analyzed. The suggested detection model beat some traditional machine learning methods with the data we gathered on the Android side, achieving 96% accuracy and a stable F1-score of roughly 96%.

The samples from two families were automatically retrieved and described using their sequences, as stated in paper [3]. Word2Vec is used to convert the author's statement sequences to the word vector space. Next, a hierarchical language model that takes advantage of the underlying hierarchical structure of malware operations was developed based on a transformer encoder in order to categorize the samples. The sample analyst can determine the crucial feature with the aid of the correlation results between the model's weights and the sample's features. The author used IoT software samples from Mirai, Gafgyt, and Benign to train our model. The author's model surpassed previous methods in this investigation, achieving a 99.12% malware recognition rate and 94.67% classification accuracy.

The author presents DeepAMD, a novel defense approach utilizing deep Artificial Neural Networks (ANNs), to combat real-world Android malware. Through an efficiency comparison with conventional machine learning classifiers and state-of-the-art studies, DeepAMD exhibits superior performance in the detection and identification of malware attacks on both Static and Dynamic layers. Notably, on the Dynamic layer, DeepAMD attains exceptional accuracy rates, achieving 80.3% accuracy for malware category classification and 59% accuracy for malware family classification, surpassing the capabilities of existing techniques. These findings highlight the effectiveness and promise of DeepAMD as a robust tool for countering Android malware and its ability to enhance accuracy in both malware categorization and family attribution [4].

Droidetec has an advanced method based on deep learning that uses natural language sequences to effectively detect malicious Android software and locate dangerous code. Its groundbreaking feature extraction approach enables the identification of behavioral sequences within Android applications. To achieve this, Droidetec uses a bi-directional Long Short Term Memory network to detect malware. By creatively transforming each extracted behavioral unit into a vector, Droidetec analyzes the semantics of the sequence segments to ultimately expose malicious code. Extensive testing on 11,982 benign programs and 9,616 malicious programs demonstrates Droidetec's exceptional performance with an impressive F1 score of 98.21% and a hit rate of 97.22%. In terms of overall effectiveness in accurately identifying malicious code snippets, Droidetec can boast a success rate of 91% [7].

The author presents a novel approach to malware detection, highlighting the limitations of traditional methods and the increasing success of artificial intelligence-based techniques. The proposed model combines grey-scale image representations of malware with an autoencoder network in a deep learning framework. It assesses the feasibility of using grey-scale images to represent malware by analyzing the autoencoder's reconstruction error. Furthermore, the autoencoder's dimensionality reduction features are leveraged for classifying malware versus benign software. The results show that the

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

model achieves a remarkable accuracy of 96% and maintains a stable F-score of around 96%, outperforming several traditional machine learning detection algorithms when tested on an Android-side dataset. This approach demonstrates the potential of using deep learning and image representations for effective malware detection [8].

A. Autoencoder

An autoencoder denotes a class of artificial neural networks proficiently utilized for the purpose of unsupervised learning, predominantly within the domain of profound learning. Its fundamental objective is to acquire adept representations or encodings of input data devoid of external guidance [9]. Figure 2 shows the architecture that encompasses two principal constituents: an encoder and a decoder.

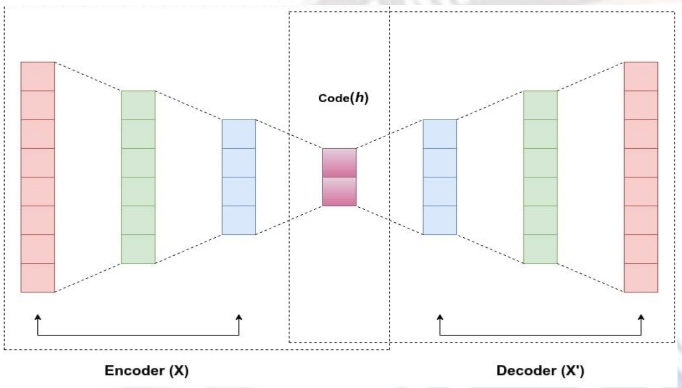


Figure 2. An architecture of Autoencoder

The role of the encoder is to convert the input data into a condensed depiction within a space of lesser dimensions. This process commonly involves layers that are concealed, with each progressive layer having diminished dimensions. Essentially, it is the encoder's responsibility to condense and encapsulate essential characteristics of the initial input.

In contrast, the decoder endeavors to retrieve and rebuild the primary input data from its encoded representation. Contrary to the workings of the encoder, this reconstruction entails augmenting data dimensions through successive layers where each subsequent layer possesses increased dimensions. The main purpose of this operation for the decoder lies in producing an output that faithfully resembles and approximates as closely as possible to its original form [10].

The following is a representation of an autoencoder:

Encoding:

Assume that x is a vector of n-dimensional input data. To create a hidden representation h, the encoder performs a series of transformations represented by a set of weights as:

$$h = f\,(W\,x + b), \tag{1}$$

where f is a rectified linear unit (ReLU) or sigmoid activation function.

Decoding:

In order to recreate the original input data, the decoder applies a new set of weights W' and biases b' to the hidden representation h.

$$R = g\,(W'\,h + b'), \tag{2}$$

Where g is the activation function used for decoding, can be used to represent this.

Loss function:

The disparity between the original input data x and the reconstructed output data r is what the autoencoder attempts to reduce. The mean squared error (MSE), which can be formulate in the following manner:

$$L\,(x, r) = \|\, x - r\,\|\; 2, \tag{3}$$

is a popular loss function used for this purpose.

B. Funnel Transformer

The funnel transformer model represents an architectural variation of the renowned transformer model commonly employed in natural language processing tasks. While preserving competitive performance, this innovative design aims to enhance computational efficiency and minimize memory requirements. The essence of its efficacy lies in a progressively narrowing structure, featuring layers that decrease in width. The fundamental goal of the Funnel Transformer is to make the self-attention mechanism for lengthy sequences less computationally complex. The self-attention mechanism in a typical Transformer has a complexity of $O(n^2)$, where n is the length of the input sequence. Because of this, it is computationally costly for extremely long sequences [11, 12].

The self-attention mechanism in the Funnel Transformer has been changed to have a complexity of O(n), enabling it to handle lengthy sequences more effectively. A number of downsampling layers are used to do this, gradually shortening the sequence length while lengthening the concealed dimensions. Figure 3 presents an idea of a funnel transformer [11, 13].

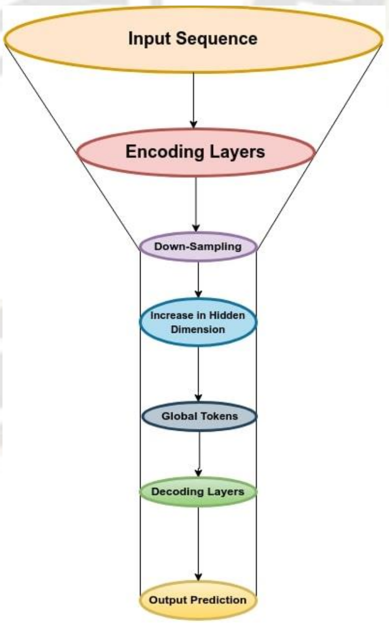


Figure 3. An architecture of Funnel Transformer

Input Sequence:

A sequence of tokens is used to represent the input sequence. Every token is contained within a large-dimensional vector.

Encoding Layers:

A number of encoding layers are applied to the input sequence. A position-wise feed-forward network comes first, then a multi-head self-attention mechanism, in each encoding layer. Each token in the sequence is given the ability to pay attention to the other tokens through the attention mechanism, capturing the connections and dependencies between them. Each token is subjected to non-linear changes individually by the feed-forward networks. Let's denote the input sequence as X = x_1, x_2, ………., x_n, where n is the length of the sequence. The Funnel Transformer can be mathematically represented as a series of encoding and decoding layers.

For each encoding layer I, the multi-head self-attention mechanism is defined as follows:

$$\text{Attention}(X) = \text{Softmax}((Q * K^T / \text{sqrt}(d\_k)) * V, \qquad (4)$$

where Q, K and V are the query, key and value matrices obtained from X. The softmax operation computes the attention weights and sqrt (d_k) is a scaling factor to stabilize the gradients. The attention output is then passed through a feed-forward network.

$$\text{EncodingLayer\_i}(X) = \text{LayerNorm}(\text{Attention}(X) + X) + \text{FeedForward}(\text{Attention}(X)), \qquad (5)$$

where LayerNorm is a layer normalization operation and FeedForward is a non-linear transformation.

Down-Sampling:

To shorten the sequence, a down-sampling procedure is used after each encoding layer. Utilizing strided convolutions, max-pooling, or other methods can accomplish this process. The downsampling collects high-level information from the input sequence while assisting in the reduction of computational complexity.

The down-sampling operation reduces the sequence length by a factor of r. Let's denote the down-sampled sequence as X', with length n. The down-sampling can be expressed as:

$$X' = \text{DownSample}(X), \qquad (6)$$

where DownSample is a function that applies a specific operation such as max-pooling or strided convolutions.

Increase in Hidden Dimension:

The tokens' hidden dimensions are also enhanced in conjunction with the downsampling. As a result, the model can now describe the input sequence in a more abstract and sophisticated manner.

The hidden dimensions of the tokens are increased along with the down-sampling. Let's denote the hidden dimension of the encoding layer i as d_i. The hidden dimension increase can be represented as:

$$d\_i = d\_i - 1 * r, \qquad (7)$$

where d_i-1 is the hidden dimension of the previous encoding layer.

Global Tokens:

A set of global tokens is chosen for each phase of the down-sampling process. The sequence's most instructive tokens are these global tokens. They summarize the entire sequence and capture the long-range dependencies.

$$G = \text{SelectGlobalTokens}(X'), \qquad (8)$$

where SelectGlobalTokens is a function that chooses the most informative tokens based on certain criteria.

Decoding Layers:

Following the downsampling procedures, a number of decoding layers are applied to the global tokens and the downsampled sequence. These layers function in the opposite manner from the encoding layers but are comparable. The down-sampled sequence and the global context are used by the decoding layers to refine the representations. For each decoding layer i, the decoding operation is similar to the encoding layer, but operates in the reverse direction. The decoding layer can be defined as:

$$\text{DecodingLayer\_i}(X', G) = \text{LayerNorm}(\text{Attention}(X', G) + X') + \text{FeedForward}(\text{Attention}(X', G)), \qquad (9)$$

where Attention (X', G) computes the attention between X', and G.

Output Prediction:

The output predictions are created by the last decoding layer. These forecasts can be applied to a variety of projects, including text classification, machine translation, and language modelling. The final decoding layer produces the output predictions. This can be represented as:

$$\text{Output} = \text{OutputPrediction}(\text{DecodingLayer\_i}(X', G)), \qquad (10)$$

where OutputPrediction is a function that produces the desired output based on the refined representations.

III. Proposed Methodology

A. *Data Preprocessing*

The comprehensive data preprocessing and analysis pipeline involves data loading, extraction, transformation, encoding, correlation analysis, normalization, and splitting to prepare the data to fit in the proposed model. Fig 4 depicts an overview of data preprocessing model.

Raw IoT-23 Data: The process begins with the raw IoT-23 dataset, likely derived from Zeek log files. This dataset contains various attributes and a "label" column representing the target variable.

Load Files and Eliminate Conversion Column: To start the data preprocessing, you load the dataset from a constructed CSV file. During this process, it's crucial to remove a specific column generated during the conversion from Zeek log to CSV. This column elimination simplifies the dataset for subsequent analysis.

Data Extraction and Categorical Encoding: Following the loading of data, you proceed to encode categorical variables using the "category" data type. This encoding transforms categorical data into numeric codes. It's important to highlight that this encoding step involves a function called "ConvertStringToInt", which converts string data to integers and handles any missing data.

Label Distribution Analysis: A significant part of the analysis involves examining the distribution of the "label" column. You create a set of unique labels, denoted as "L" and count the occurrences of each label. This analysis helps understand the distribution of the target variable in the dataset.

Correlation Analysis and Feature Selection: To gain insights into the relationships between variables, you compute a correlation matrix for the dataset. Visualizing this matrix as a heatmap provides a graphical representation of the correlations between features. It's worth noting that you decide to remove the "local_orig" and "local_resp" columns, which are deemed irrelevant for the heatmap analysis.

Data Preparation and Feature Selection: In this phase, you extract both the input features, referred to as "X" and the target variable, denoted as "Y" from the DataFrame. The target variable is transformed into one-hot encoding, a common technique for dealing with categorical labels. A specific subset of columns, including "ts", "uid", and various network-related attributes, is selected as input features (X), while the "label" column is designated as the target variable (Y).

Data Normalization: Normalization is crucial to ensure that all input features are on the same scale. The introduction of a MinMaxScaler object and apply it to the input features (X) for appropriate scaling. The same scaler is then used to fit and transform the target variable (Y), bringing it into alignment with the transformation applied to the features.

Data Splitting: Finally, the normalized data is divided into two distinct sets: the training set and the testing set. To ensure reproducibility, a fixed value is assigned to the random state parameter. The size pf testing data and training data specifies into 20 % and 80%, respectively. This division is essential for training and evaluating machine learning models.

Algorithm 1 Data Pre-Processing Algorithm

```
Require: IoT-23 datasets
Ensure: X_train, X_test, Y_train, Y_test
1: $df \Leftarrow load\_dataset()$ // Load dataset
   // Data Pre-processing
2: $df \Leftarrow df.drop(columns = [col1])$ // Remove extra "Unnamed" column generated during conversion from zeek file to csv file.
3: $df\_converted = ConvertStringToInt(df\_cleaned)$ // Encoded categorical column
4: $Count(label) = count(label\_i) : label\_i \in L$ // Count the values values in target variable.
5: $M \Leftarrow corr(df)$ // Calculate correlation matrix
6: $M = [m\_\{ij\}]$ // Visualize correlation matrix as a heatmap
7: $df \Leftarrow df.drop(columns = [col1, col2, col3, ..])$ // Drop no relation columns from the dataframe observing heatmap
8: $X \Leftarrow df[f1, f2, f3, ...].values$ // Extract features from the dataframe
9: $Y \Leftarrow df[t1, t2, ...].values$ // Extract target columns
10: $scaler \Leftarrow MinMaxScaler()$ // Scale feature X and Y
11: $normalized\_x[i, j] = (X[i, j] - min(X[:, j]))/(max(X[:, j]) - min(X[:, j]))$
12: $normalized\_y[i, j] = (Y[i, j] - min(Y[:, j]))/(max(Y[:, j]) - min(Y[:, j]))$
13: $X\_train \Leftarrow \{normalized\_x\_i | normalized\_x\_i \in normalized\_x, i \in I\_train\}$
    $X\_test \Leftarrow \{normalized\_x\_i | normalized\_x\_i \in normalized\_x, i \in I\_test\}$
    $Y\_train \Leftarrow \{normalized\_y\_i | normalized\_y\_i \in normalized\_y, i \in I\_train\}$
    $Y\_test \Leftarrow \{normalized\_y\_i | normalized\_y\_i \in normalized\_y, i \in I\_test\}$
14: return preprocessed data (X_train, Y_train, X_test, Y_test)
```

B. *Proposed 3SAFT architecture*

The study introduces a groundbreaking malware detection architecture known as the "3 Sigma AutoFunnel Transformer". This architecture is novel and innovative in its approach to identifying and mitigating malware threats. The "3 Sigma" likely signifies a high level of precision, emphasizing the model's accuracy in detecting malicious software. The "Auto-Funnel Transformer" suggests an automated and efficient method for processing and analyzing data, resembling the operations of a funnel that narrows down and processes information effectively. Together, this architecture leverages advanced techniques, potentially including deep learning, to enhance the security and reliability of malware detection, making it a significant advancement in the field of cybersecurity. The details of proposed architecture implementation are highlighted in Figure 5. The performance of the proposed architecture is compared with the existing models, as shown in Figure 6. Figure 7 presents a "3SAFT" flow chart. Figure 8 describes the mechanism of Auto Funnel Transformer.

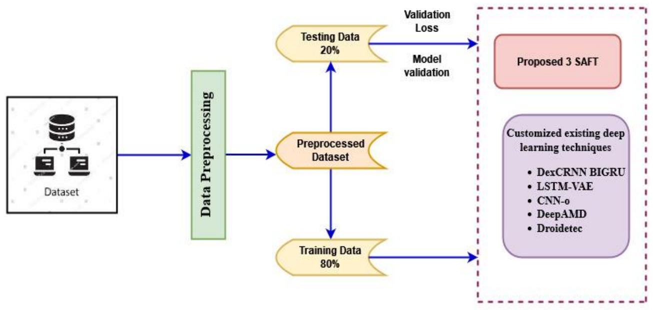


Figure 5. An architecture of Funnel Transformer

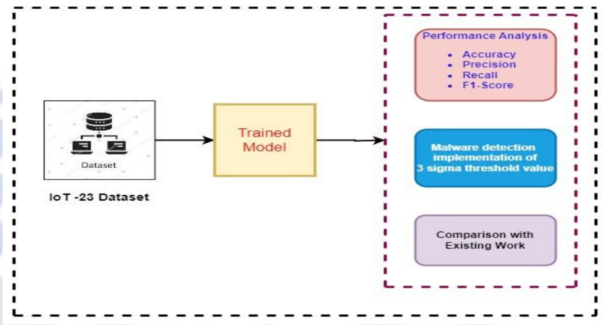

Dataset
IoT -23 Dataset
Trained Model
Performance Analysis
Accuracy
Precision
Recall
F1-Score
Malware detection implementation of 3 sigma threshold value
Comparison with Existing Work

Figure 6. An architecture of Funnel Transformer

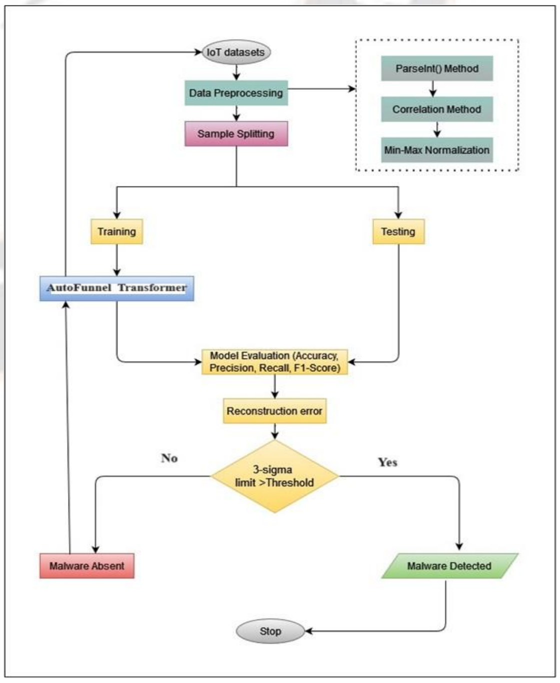


Figure 7. 3 Sigma Auto-Funnel Transform flow chart

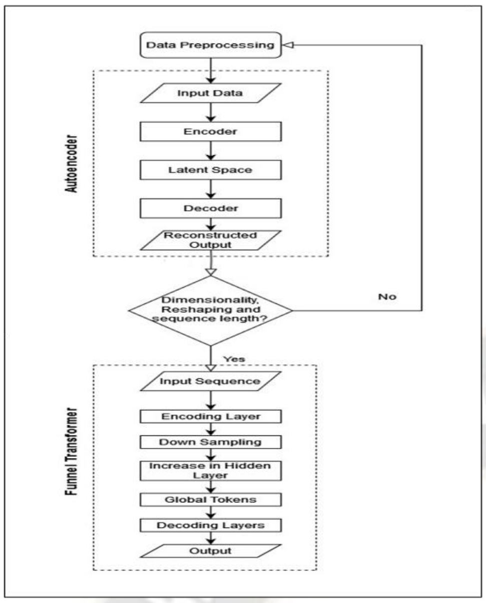


Figure 8. Mechanism of Auto funnel Transformer

C. 3 Sigma Auto-Funnel Transformer

The fusion of an autoencoder with a funnel transformer yields a robust architecture suitable for IoT devices. This model adeptly leverages the autoencoder's ability to acquire efficient representations and the computational efficiency inherent in a funnel transformer. In the ensuing discourse, we will delve into the elucidation of the proposed model, emphasizing its advantageous attributes by integrating the previously mentioned mathematical expressions into our analysis.

The autoencoder's role is encoding efficient representations of input data through its encoder-decoder structure, as previously explained. In contrast, the funnel transformer was intentionally designed to reduce computational complexity and memory requirements while maintaining competitive performance. This component features an architectural configuration resembling a funnel, characterized by a decrease in attention heads and an increase in hidden layer sizes.

Considering this, the proposed integrated model combines both the autoencoder and funnel transformer components to leverage their individual strengths. Specifically, the autoencoder acts as an initial training stage for assimilating semantically meaningful representations. Subsequently, the funnel transformer employs these acquired representations for further processing and predictive tasks.

In **Autoencoder Encoder**, the input data, denoted as 'x' is mapped into a lower dimensional latent space representation 'h' using the following equation:

$$h = f\,(W\_e * x + b\_e), \tag{11}$$

here, 'f' represents the activation, while 'W_e' and 'b_e' are the weight matrix and bias of the encoder, respectively.

In **Autoencoder Decoder**, the decoder component of the autoencoder takes the latent representation 'h 'and endeavors to reconstruct the original input data 'x' with the following equation.

$$r = g\,(W\_d * h + b\_d), \tag{12}$$

The activation function 'g 'is applied here, and 'W_d' and 'b_ d 'denotes the weight matrix and bias of the decoder.

Funnel Transformer is the part of the model takes the encoded representation 'h' from the autoencoder and processes it using a funnel-shaped architecture, characterized by decreasing attention heads and increasing hidden layer sizes. The self-attention and feed-forward network in each layer adhere to the previously discussed mathematical expressions.

The 3 Sigma (3σ) limit in malware detection uses statistical methods to define a threshold for discerning normal system behavior from potentially malicious activities. Deviations beyond this limit trigger alerts, indicating possible malware actions. This approach offers continuous, quantitative monitoring of system behavior, enabling proactive detection and response to anomalies, as a consequence strengthening cybersecurity defenses against threats.

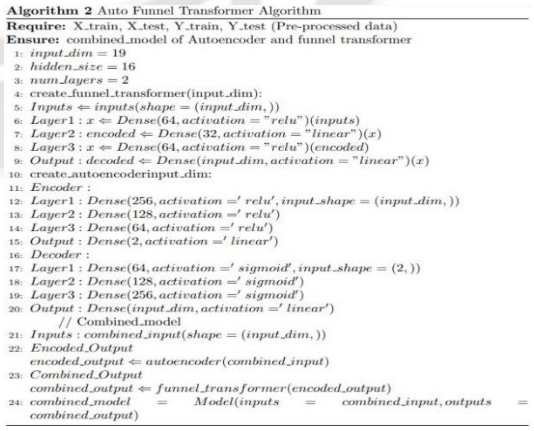
Hybrid Model: In this process, a unified model is created by combining a funnel transformer and an autoencoder. Initially, a funnel transformer model is instantiated, defining the expected input dimension. Similarly, an autoencoder model is created with the same input dimension. The input data is then passed through the autoencoder, resulting in an encoded representation. A combined input layer is introduced to accept inputs from both models, leveraging the specified input dimension. Finally, the encoded output from the autoencoder is processed by the funnel transformer to produce the ultimate combined output. The entire architecture is encapsulated in a combined model, offering a holistic approach that first encodes input data and then employs the funnel transformer for further processing, ultimately generating the model's final output observed in algorithm 2. Algorithm 3 describes the addressing malware issues in IoT devices using 3 Sigma limit. The following equations depict the combined model of funnel transformer and autoencoder. Fig 9 depicts a representation of a hybrid proposed model.

$$\text{inputs} = \text{combined\_input (shape = (input\_dim,))}, \tag{13}$$

$$\text{encoded\_output} = \text{autoencoder (combined\_input)}, \tag{14}$$

$$\text{combined\_output} = \text{funnel\_transformer(encoded\_input)} \tag{15}$$

$$\text{combined\_model} = \text{Model (inputs = combined\_inputs, outputs = combined\_output)}, \tag{16}$$



Algorithm 2 Auto Funnel Transformer Algorithm

```
Require: X_train, X_test, Y_train, Y_test (Pre-processed data)
Ensure: combined_model of Autoencoder and funnel transformer
 1: input_dim = 19
 2: hidden_size = 16
 3: num_layers = 2
 4: create_funnel_transformer(input_dim):
 5: Inputs ⇐ inputs(shape = (input_dim, ))
 6: Layer1 : x ⇐ Dense(64, activation = "relu")(inputs)
 7: Layer2 : encoded ⇐ Dense(32, activation = "linear")(x)
 8: Layer3 : x ⇐ Dense(64, activation = "relu")(encoded)
 9: Output : decoded ⇐ Dense(input_dim, activation = "linear")(x)
10: create_autoencoderinput_dim:
11: Encoder :
12: Layer1 : Dense(256, activation =' relu', input_shape = (input_dim, ))
13: Layer2 : Dense(128, activation =' relu')
14: Layer3 : Dense(64, activation =' relu')
15: Output : Dense(2, activation =' linear')
16: Decoder :
17: Layer1 : Dense(64, activation =' sigmoid', input_shape = (2, ))
18: Layer2 : Dense(128, activation =' sigmoid')
19: Layer3 : Dense(256, activation =' sigmoid')
20: Output : Dense(input_dim, activation =' linear')
        // Combined_model
21: Inputs : combined_input(shape = (input_dim, ))
22: Encoded_Output
    encoded_output ⇐ autoencoder(combined_input)
23: Combined_Output
    combined_output ⇐ funnel_transformer(encoded_output)
24: combined_model = Model(inputs = combined_input, outputs =
    combined_output)
```

Algorithm 3 Addressing Malware using 3 Sigma limit Algorithm

```
Require: combined_model,combined_input,combined_output
Ensure: precision, F1-score , accuracy, graphical representation of malware attacks
    // Model Compilation
 1: $compiled\_model \Leftarrow combined\_model.compile(optimizer = 'adam', loss = 'mse', metrics = ['accuracy'])$
 2: $Model\_Training$
    $combined\_model.fit(X\_train, X\_train, epochs = 50, batch\_size = 256, validation\_data = (X\_test, X\_test), verbose = 1)$
 3: $Target\_Variable\_Selection$
    $Y\_train \Leftarrow Y\_train[:, 0], Y\_test \Leftarrow Y\_test[:, 0]$
 4: $Model\_Training$
    $classifier.fit(X\_train, Y\_train)$
 5: $Predictions$
    $predictions \Leftarrow classifier.predict(X\_test)$
 6: $Precision\_Calculation$
    $precision \Leftarrow precision\_score(Y\_test, predictions, average = 'weighted')$
 7: $F1\_Score\_Calculation$
    $f1 \Leftarrow f1\_score(Y\_test, predictions, average = 'weighted')$
 8: $Reconstruction\_loss\_Calculation$
    $reconstruction\_errors \Leftarrow np.mean(np.square(X\_train - combined\_model.predict(X\_train)), axis = 1)$
 9: $Threshold\_Calculation$
    $threshold \Leftarrow np.mean(reconstruction\_errors) + 3 * np.std(reconstruction\_errors)$
10: $Anomaly\_Labels$
    $malware\_labels \Leftarrow np.where(reconstruction\_errors > threshold, 1, 0)$
```

IV. Experimental Setup

The IoT-23 dataset encompasses network traffic data from a variety of Internet of Things (IoT) devices, consisting of 23 instances. Among these, three instances contain benign traffic data, while 20 instances are associated with malicious software activities. This dataset was initially made public in January 2020, and it covers the period from 2018 to 2019, with monitoring conducted by the Stratosphere Lab at the AIC Group within FEL, CTU University in the Czech Republic. A crucial component of this dataset is the conn.log. labeled files generated using Zeek's network analysis, providing detailed insights and unique attributes for each entry. The dataset also includes the primary source data files known as pcap files, with a total of 325,307,990 observations. Notably, the majority of these records, specifically 294,449,255, pertain to malicious activities. In this context, N represents the total number of records in the collection, which is 1,444,674, and each record, denoted as 'x' is a vector representation composed of various attributes.

A. Key Performance Indicators

In the field of deep learning and artificial intelligence, performance metrics are used to assess the quality and effectiveness of machine learning models. These metrics help in understanding how well a model is performing on a particular task, such as classification, regression, or object detection. Some common performance metrics in deep learning include:

Accuracy: Accuracy measures the overall correctness of predictions and is computed as the ratio of correctly classified instances to the total number of instances.

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN). \quad (17)$$

where, TP = True Positives (Malware correctly identified).
TN = True Negatives (Non-malware correctly identified).
FP = False Positives (Non-malware incorrectly classified as malware).
FN = False Negatives (Malware incorrectly classified as non-malware).

Precision: Precision quantifies the model's ability to minimize false positives, i.e., it measures the proportion of correctly identified malware out of all positive predictions.

$$\text{Precision} = TP / (TP + FP) . \quad (18)$$

Recall: Recall assesses the model's ability to minimize false negatives, i.e., it measures the proportion of correctly identified malware out of all actual malware instances.

$$\text{Recall} = TP / (TP + FN). \quad (19)$$

F1-Score: F1 Score is the harmonic mean of precision and recall, providing a balanced evaluation that considers both false positives and false negatives.

$$\text{F1-Score} = 2 * ((\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})). \quad (20)$$

V. Results and Analysis

Performance analysis of 3SAFT: The data for the "3SAFT" method demonstrates an exceptional level of performance across multiple metrics. First, the training and validation loss indicate that the model exhibits efficient learning during training while avoiding overfitting on the validation set. Additionally, the high training and validation accuracy figures highlight that the model performs remarkably well not only on the training data but also on unseen validation data. The model's high accuracy of 99.73% suggests that it correctly classifies an overwhelming majority of the data points in both training and validation dataset. The precision score of 99.46% is equally impressive, indicating that when the model predicts a positive outcome, it is highly accurate with very few false positives. The recall rate of 99.73% underlines the model's ability to capture almost all actual positive cases, making it highly sensitive. Moreover, the F1-Score, which is very close to the accuracy at 99.59%, signifies a harmonious balance between precision and recall. In conclusion, "3SAFT" excels across all key metrics, making it an ideal choice for tasks where both precision and recall are paramount, with its robust performance seen in both the training and validation phases. This suggests a well-generalized model with a strong ability to classify and identify positive cases accurately.

The malware detection model's training accuracy and validation accuracy are shown in Figure 10. The percentage of samples that were properly identified during the training phase is measured by training accuracy, whereas the percentage is measured by validation accuracy using a different validation dataset. Figure 11 y-axis displays accuracy values, which range from 0 to 1, with 1 denoting perfect accuracy. Higher accuracy scores show that the model's predictions and actual labels are closely correlated, which reflects a better comprehension and representation of the data. The training epochs or iterations are represented on the x-axis, which shows how far along the training process is. The model is performing at a very high level overall, with a low rate of false positives and false negatives, according to these evaluation measures. The model appears to be successful in its classification task and can be regarded as reliable for the provided dataset based on the high precision, recall, F1 score, and accuracy values.

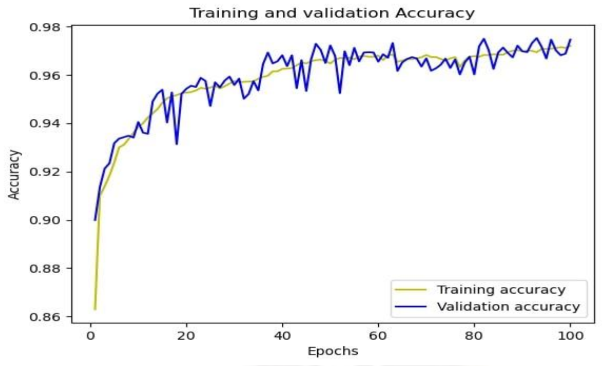


Figure 10. Training and validation accuracy

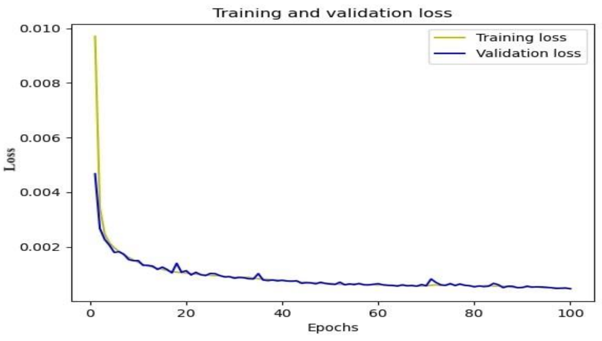


Figure 11. Training and validation loss

Performance analysis with existing work: In Table 1, a performance analysis is presented for the proposed work alongside existing work. The analysis includes two key metrics: validation loss and training loss. The validation loss measures the difference between model predictions and actual data on a separate validation dataset, while the training loss indicates the disparity between the model's predicted outputs and the actual ground truth labels during the training process. These loss values serve as numerical indicators of the model's effectiveness and are plotted on the y-axis in the accompanying figure. Lower loss values signify a better fit to the data, indicating that the model's predictions closely align with the ground truth labels. The x-axis represents the training epochs or iterations, illustrating the progression of the training procedure. This analysis provides a comparison of the proposed and existing approaches based on their loss values and training progress. Figure 13, Figure 14, Fig 15 and Figure 16 depicts a graphical representation of data mention in table 1.

TABLE I. PERFORMANCE COMPARISON OF PROPOSED WORK ALONG WITH EXISTING WORK

Methods	*Accuracy*	*Precision*	*Recall*	*F1-Score*
DexCNN BiGRU [9]	95.80 %	95.40 %	96.20 %	95.80 %
LSTM-VAE [7]	97.29%	94.63 %	96.07 %	95.34 %
CNN-o [11]	94.60 %	94.96 %	94.10 %	94.53 %
DeepAMD [4]	80.30 %	82.20 %	80.30 %	80.50 %
Droidetec [8]	97.22 %	98.21 %	98.35 %	98.20 %
3-SAFT (Proposed Approach)	99.73 %	99.46 %	99.73 %	99.59 %

In Figure 12, a visual depiction of the discovered anomalies or probable malware instances is provided via the plotted reconstruction errors and the labelled anomalies. Understanding the prevalence and severity of the reconstruction mistakes across the dataset is made easier by the figure. The probable malware cases are indicated by the red scatter spots that emphasize the areas where the reconstruction mistakes are greater than the threshold. Overall, based on the metrics for evaluation and the visualization, it appears that the malware detection system that uses a combination model and reconstruction error to identify malware samples is quite accurate and efficient.

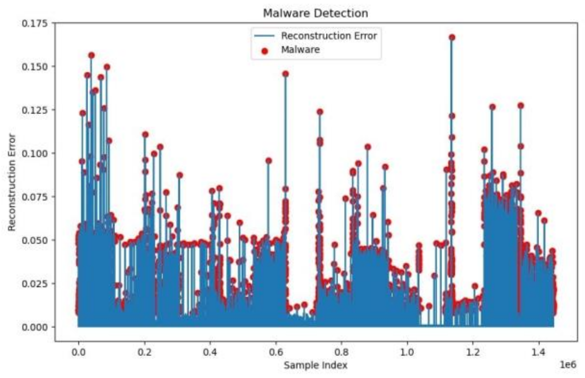


Figure 12. Addressing Malware

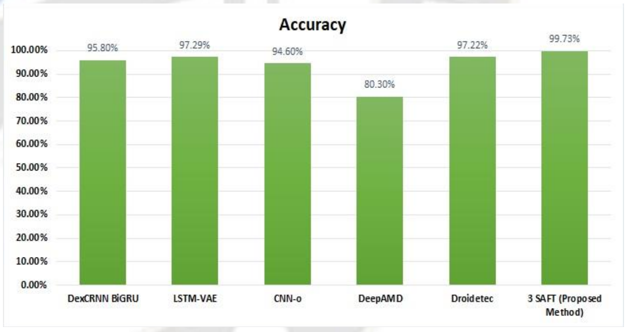


Figure 13. Accuracy

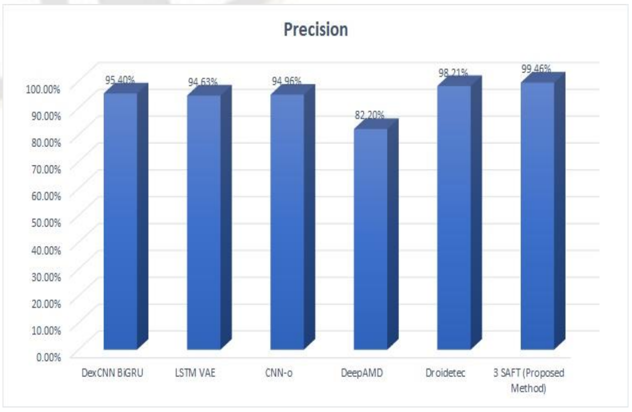


Figure 14. Precision

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

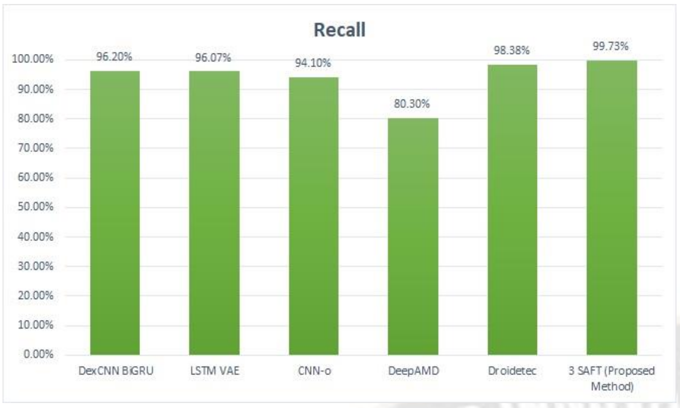


Figure 15. Recall

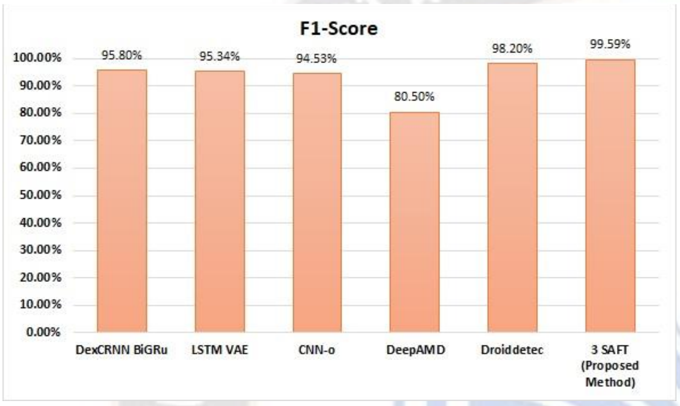


Figure 16. F1-Score

VI. Conclusion

In conclusion, the autoencoder and Funnel Transformer model combined has a lot of potential for solving the malware problem in IoT devices. This methodology effectively detects and mitigates malicious software threats in IoT environments by combining the advantages of both architectures. In order to detect aberrant patterns that can point to the existence of malware, the autoencoder component of the model facilitates the learning of meaningful representations of IoT device data. The autoencoder detects key characteristics and patterns that may be suggestive of malicious behavior by compressing the input data into a lower-dimensional latent space and then recreating it. The Funnel Transformer component improves the model's capacity to examine and categorize data from IoT devices. Its hierarchical structure effectively processes the sequential nature of IoT data and identifies complicated malware patterns by capturing both local and global dependencies.

Overall, the Autoencoder and Funnel Transformer model combined gives a viable solution to the problem of malware detection in IoT devices. It offers a strong and effective method for spotting dangerous patterns in IoT data by combining the strengths of both designs, thereby strengthening the security and integrity of IoT ecosystems. Through a diligent concentration on these forthcoming avenues of research, scholars have the opportunity to propel the amalgamated autoencoder and Funnel Transformer model towards confronting malware in IoT devices. These endeavors possess the potential to augment the model's efficacy, expand its capacity for growth, elucidate its interpretability, and notably bolster its practicality within real-world scenarios. Consequently, such endeavors will fortify the security and steadfastness of IoT ecosystems against incessantly transforming malware perils.

Acknowledgment

The authors would like to thank Department of Computer Science and Engineering, Assam Don Bosco University for all the support.

References

[1] M. K. Asif, T. A. Khan, T. A. Taj, U. Naeem and S. Yakoob, “Network Intrusion Detection and its strategic importance,” 2013 IEEE Business Engineering and Industrial Applications Colloquium (BEIAC), Langkawi, Malaysia, 2013, pp. 140-144, doi: 10.1109/BEIAC.2013.6560100.
[2] Z. Dai, G. Lai, Y. Yang, Y., and Q.V. Le,“Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing,”*arXiv e-prints*, 2020. doi:10.48550/arXiv.2006.03236.
[3] X. Hu, R. Sun, K. Xu, Y. Zhang and P. Chang, "Exploit Internal Structural Information for IoT Malware Detection Based on Hierarchical Transformer Model," 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 2020, pp. 927-934, doi: 10.1109/TrustCom50675.2020.00124.
[4] S. I. Imtiaz, S. Rehman, A.R. Javed, Z. Jalil, X. Liu and W.S. Alnumay, “ Deep-AMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network,” Future Generation computer systems, Elsevier, vol. 115, 2021, pp. 844-856, https://doi.org/10.1016/j.future.2020.10.008.
[5] Y. LeCun, Y. Bengio and G. Hinton, “ Deep learning,” nature, vol. 521, 2015, pp. 436-444, https://doi.org/10.1038/nature14539.
[6] T. Lin, Y. Wang, X. Liu and X. Qiu, “ A survey of transformers,” AI Open, Elsevier, vol. 3, 2022, pp. 111-132, https://doi.org/10.1016/j.aiopen.2022.10.001.
[7] M. Liu, S. Wei and P. Jiang, “ A hybrid modeling of mobile app dynamics on serial causality for malware detection”, Security and Communication Networks, vol. 2021, 2021, pp. 1-10, https://doi.org/10.1155/2021/9994588.
[8] Z. Ma, H. Ge, Z. Wang, Y. Liu and X. Liu, “Droidetec: Android malware detection and malicious code localization through deep learning,” arXiv preprint arXiv:2002.03594, 2020, https://ui.adsabs.harvard.edu/link_gateway/2020arXiv200203594M/doi:10.48550/arXiv.2002.03594
[9] Z. Ren, H. Wu, Q. Ning, I. Hussain and B. Chen, “End-to-end malware detection for android IoT devices using deep learning,” Ad Hoc Networks, Elsevier, vol. 101, Feb. 2020, pp. 102098, https://doi.org/10.1016/j.adhoc.2020.102098.
[10] Y. Tay, D. Bahri, D. Matzler, D.C. Juan, Z. Zhao and C. Zheng, “Synthesizer: Rethinking self-attention for transformer models,” in International conference on machine learning, PMLR, 2021, pp. 10183-10192.
[11] X. Xing, X. Jin, H. Elahi, H. Jiang and G. Wang, "A Malware Detection Approach Using Autoencoder in Deep Learning," in IEEE Access, vol. 10, pp. 25696-25706, 2022, doi: 10.1109/ACCESS.2022.3155695.
[12] J. Zhai, S. Zhang, J. Chen and Q. He, "Autoencoder and Its Various Variants," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 2018, pp. 415-419, doi: 10.1109/SMC.2018.00080.
[13] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, “Deep autoencoding gaussian mixture model for unsupervised anomaly detection.,” in International conference on learning representations, 2018.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

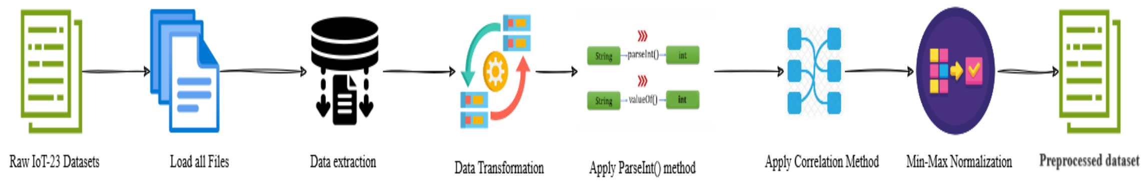


Figure 4. A brief overview on data preprocessing

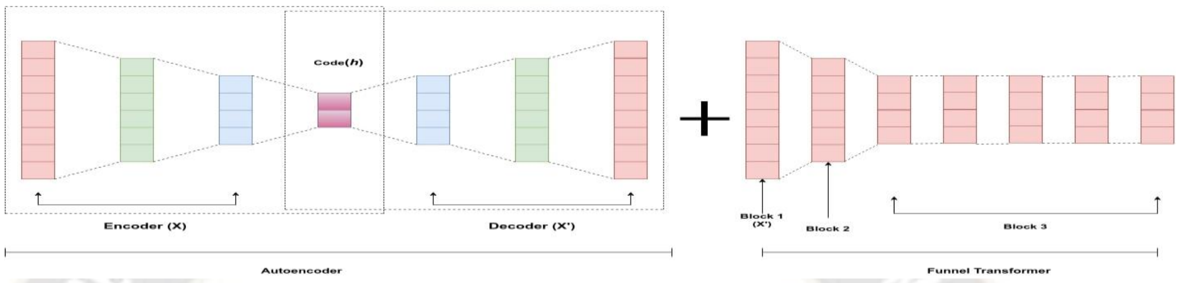


Figure 9. Proposed hybrid model