

Anomaly Detection in Cybersecurity Logs Using a Convolutional Autoencoder With 2d Matrix Representation

Hari Krishna Puripanda¹, Dr Vemuru Srikanth²

^{1,2}Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Green Fields, Guntur, AP, India. 2401050119@kluniversity.in¹, vsrikanth@kluniversity.in²

Abstract

The growing complexity of cyberattacks presents organizations with severe issues that are based on outdated security monitoring infrastructure. Although signature-based intrusion detection systems are effective against catalogued threats, they are essentially reactive and do not identify zero-day exploits, Advanced Persistent Threats or subtle behavioural abnormalities that do not correspond to any predefined rule. The requirement to automate, adapt and unsupervised detection mechanisms has never been more urgent as enterprise networks generate logs on a scale and velocity never before seen. Current machine learning methods, such as One-Class Support Vector Machines, Isolation Forest, and LSTM-based autoencoders, consider cybersecurity log data as a flat sequential vectors or scalar time series. This representation is fundamentally constrained as it loses the structured co-occurrence relations among features that are activated concurrently, relations that tend to contain the most discriminative information towards the distinction of malicious behaviour and normal functioning. To overcome this representational gap, we have suggested an CNN-based autoencoder architecture that encodes sequential log entries into two-dimensional matrix formats and passes them to the detection model. Sliding windows of normalized log vectors are then rearranged into a fixed-size matrix, which effectively represents the temporal and feature-wise relationships as spatial patterns. These matrices are then processed by convolutional layers that extract hierarchical features that include local and global dependencies that one-dimensional processing is unable to reclaim. The autoencoder is only trained using normal traffic data and anomalies are detected at inference time when the reconstruction error surpasses a statistically determined threshold. On the UNSW-NB15 and CICIDS2017 benchmark data, we measure the proposed system with an 80/20 stratified division and 5-fold cross-validation. The model has a detection accuracy of 96.0, preciseness of 95.2, recall of 96.8 and a F1-score of 96.0, which is better than all the four baseline approaches across all measures. These findings indicate that when cybersecurity log analysis is re-framed as a problem of spatial pattern recognition, then meaningful and measurable benefits are obtained and the low per-event inference latency of the system means it is a good candidate to be deployed in operational security monitoring corners.

Keywords: Anomaly Detection, Convolutional Autoencoder, Cybersecurity Log Analysis, Two-Dimensional Matrix Representation, Network Intrusion Detection, Unsupervised Learning, Reconstruction Error Thresholding.

1. INTRODUCTION

The extent of harm caused by cybersecurity incidents has been increasingly harmful over the last decade, not due to attackers abandoning old tricks, but due to the tools provided to defenders failing to keep up with the structural complexity of new threats. The WannaCry ransomware campaign in 2017 was spread through 150 countries in a few hours, using an unpatched vulnerability in Windows where no behavioural signature was available at the time. All the affected organizations were having intrusion detection systems in operation. The majority of such systems did not sound an alarm. It is not an edge case, it is the constraint of signature-based detection, and it applies to both zero-day exploits and slow-moving Advanced Persistent Threats, as well as insider attacks which do not activate a known pattern. This failure begs the question which is not whether more effective detection can be achieved but what should it be like. Larger and modern enterprise networks generate logs in a volume that is utterly impractical to review manually. The volume of log entries per day within a firewall, endpoints, authentication servers and web applications can be millions of log entries in a mid-sized organization. In that book, the proportion of records attributed to malicious activity is usually a minor percentage and it is often difficult to distinguish between genuine and malicious activity when any given event is considered in isolation. The signal is not in individual events but in the structural relations with the concurrent features and among series of events. Current deep learning techniques have achieved real strides in this direction. Recurrent networks like LSTMs are used to learn temporal dependencies between sequences of events and have demonstrated good performance on log-based detection problems. Dense autoencoders are trained to learn condensed feature representations of normal behaviour and detect abnormalities by reconstruction error. All these solutions are solutions to a part of the problem. Their similarity, however, lies in a representational deficiency: they manipulate log data as a one-dimensional vector or a flat sequence, thus losing the co-occurrence structure of features that are simultaneously active within an event window. That structure has discriminative information which is always left unutilized with the current methods. To fill this gap, we suggest a framework, which processes sequential cybersecurity logs data into two-dimensional matrices and then inputs it into a

convolutional autoencoder. Normalized log vectors are reshaped into fixed-size matrices as sliding windows, which encode the time order of events as well as the spatial co-occurrence of simultaneously present features in a form that convolutional layers are well adapted to process. Autoencoder is only trained on regular traffic and the anomalies are detected at the time of inference by evaluating the extent to which the trained model can reconstruct a new input. In this paper, we have achieved three main contributions: (1) we present a data transformation method that reforms sequential log windows into 2D matrices, enabling a spatial convolutional analysis to apply to non-image cybersecurity data; (2) we create and train a CNN-based autoencoder that learns a compressed representation of normal network behaviour and labels anomalies with statistically peaked reconstruction error; and (3) The system is designed based on three tangible goals that informed all design decisions that are presented in this paper. The former aims to identify anomalous network behaviour without any labelled attack examples in training, thus the system can be used in situations where the historical attacks data is either not available or incomplete. The second goal is to obtain recall and precision levels useful enough to operate at higher rates of capturing more attacks than the current unsupervised techniques and at the same time maintain low false alarm rates that do not overwhelm security analysts. The third goal is to be able to run at a speed of inferences that can be run in real-time, such that threats are detected as they happen and not found in hindsight during a post-incident analysis.

2. Literature Review

Identifying network traffic and system logs anomalies has been a research issue over a number of decades, yet the approaches to the task have existed and evolved significantly as both the threat environment and the available computing resources shifted. The current literature can be categorized into three general topics: classical machine learning methods, deep learning sequence models, and convolutional and representation learning methods. Every theme has added some value to the field, and has created gaps that are being filled by the present work. One of the earliest approaches to apply in a systematic way to network intrusion detection was classical machine learning techniques. One-Class Support Vector Machines were early adopted since they are able to establish a decision boundary over normal data without labelled attack examples, which are often not available in practice [1]. Isolation Forest presented an alternative algorithm, finding patterns of anomaly based upon random partitioning of a record into the rest of the dataset [2]. The two methods have demonstrated decent performances on controlled benchmark datasets. Isolation Forest variants have been used with moderate success to detect web traffic anomalies in structured traffic logs [3]. An extensive review of machine learning-based anomaly detection techniques over a broad coverage of classical and hybrid techniques revealed that these techniques can perform relatively well when features distributions are relatively constant, but fail when data is high-dimensional, time-correlated, or experiences concept drift with time [4]. This is exactly what live network traffic poses, and which led to the transition to deep learning. Deep learning models have greatly extended the capabilities of anomaly detection, since they learn representations directly on raw or low-level processed data, instead of using hand-crafted features. The use of Long Short-Term Memory networks was popularized as the network of choice in sequential log data due to its ability to learn the time-dependent dependencies without explicit feature engineering [5]. Hybrid CNN-LSTM architectures have been trained on industry IoT networks and have demonstrated competitive performance in obtaining local feature patterns and longer-term dependencies concurrently [6]. This has been further extended to attention based CNN-LSTM models that enable the model to assign various time steps different weightings in the classification process [7]. Autoencoders based on LSTMs have been specifically trained to detect an anomaly by training the reconstruction model on regular sequences and using the high reconstruction error as an anomaly signal [8], [9], [10]. These methods outperform classical methods on benchmark datasets, and have a structural weakness: they reprocess input data as one-dimensional sequences and do not directly model the spatial co-occurrence relationships between features that are active simultaneously within a single event window. The third theme has to do with convolutional and representation learning methods, which have outgrown sequential processing and taken advantage of structural patterns in data. Convolutional-only autoencoders have been used to detect anomalies in system logs, and have demonstrated that patterns that cannot be captured by fully connected architectures can be captured by spatial feature extraction [11], [12]. An example of unsupervised convolutional representations suggested by a fuzzy CNN autoencoder using log data was shown to be able to generalize across log formats with no manual feature selection used [13]. Network traffic anomaly detection has also been performed using transformer-based models, which have attention mechanisms that give interpretable explanations of anomalies and detection scores [14]. CNN-based networks used directly on network intrusion detection have always demonstrated accuracy gains over LSTM-only baselines when the input representation maintains a spatial structure [15], [16], [17]. Experiments that focus on using two-dimensional convolutional representation to classify encrypted traffic have demonstrated that converting sequential data into a matrix format and then running CNN layers is more effective in classifying traffic and also compacting features [18], [19]. Additional architectures Hybrid Hybrid BiLSTM-CNN and CNN-

GRU have also verified that spatial and temporal processing can be more powerful together compared to either single-modality architecture [20], [21], [22]. In spite of these developments there is limited research on the implementation of an all-CNN-based autoencoder to anomaly detection of cybersecurity logs based on a purposeful and mathematically-grounded 2D matrix transformation of the sequence records [23], [24]. The majority of the literature has employed CNNs to classify and not apply unsupervised reconstruction, or informally employs the 2D transformation but does not consider its contribution alone. The gap is filled in this work by proposing, implementing and systematically evaluating a framework wherein the 2D transformation is the key design choice and its contribution is measured by a controlled ablation study [25].

3. System Methodology

The proposed anomaly detection architecture solves the basic issue of detecting malicious behaviour in cybersecurity logs without labelled data on attacks to train on. Figure 1 gives a full picture of the pipeline, which shows raw sequential log records being processed by four separate processing stages to derive a final anomaly decision and severity score. On a high-level, the system operates as follows: a set of raw log sequences is processed into normalized and reorganized two-dimensional matrix representations, and the matrices are compressed into a small latent code using a convolutional encoder, the decoder tries to reconstruct the original matrix using that small latent code, and the quality of that reconstruction is used to determine whether the input is flagged as anomalous. Summary of the suggested four-stage pipeline of anomaly detection. Stage 1 transforms raw log sequences into normalized 2D matrices via a sliding window operation. These matrices are then transferred to the convolutional encoder Φ create a smaller latent code z . Stage 3 restores the input by using the transposed convolutional decoder, Ψ . Stage 4 calculates the reconstruction error and then compares it to the statistically determined threshold τ to generate a binary label of anomaly and a continuous measure of severity. The solid arrows are the training path, whereas the dashed arrows are the inference-only path.

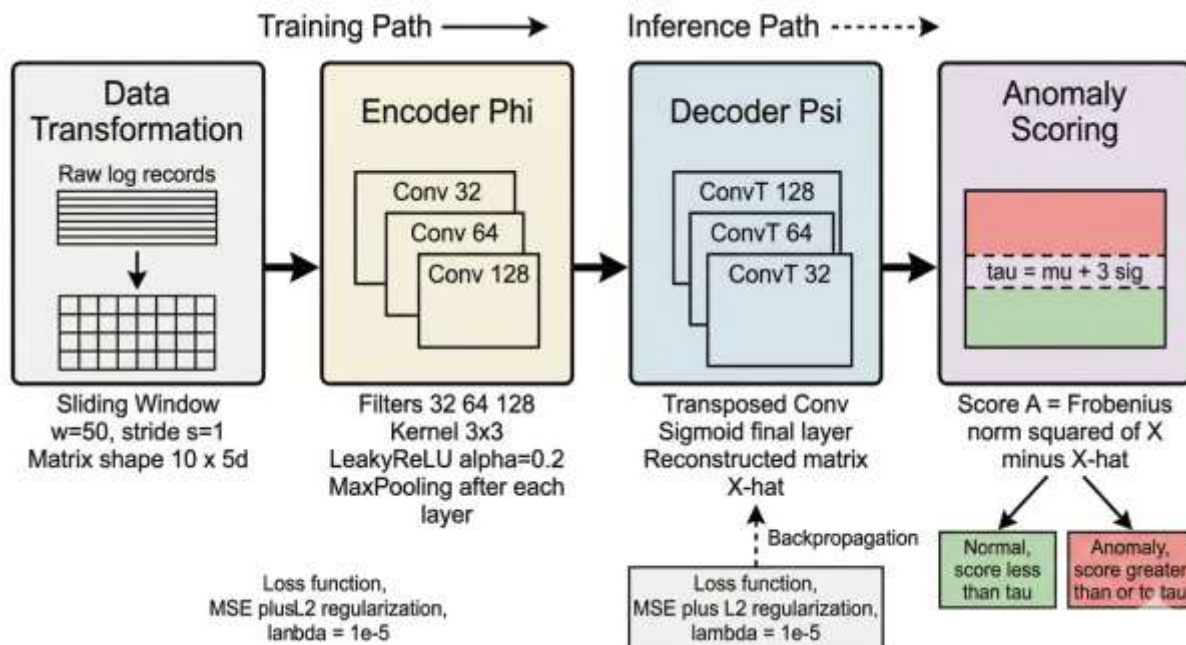


Fig 3.1 Overview of the proposed four-stage anomaly detection pipeline.

Step 1: The raw data from the logs is transformed into a normalized 2D matrix using the sliding window technique. Step 2: The matrix from step 1 is fed to the convolutional encoder model Φ to get the latent representation z . Step 3: The latent space z is then passed through the transposed convolutional decoder Ψ to reconstruct the input. Step 4: Finally, the reconstruction loss is calculated and compared with the statistically calculated threshold τ to obtain a binary anomaly label and a real number anomaly score. The experiments that prove this methodology have been carried out on two publicly available benchmark datasets. UNSW-NB15 has 2,540,044 records with 49 features that define nine attack types and a normal traffic category, and the official train and test splits are 175,341 records and 82,332 records, respectively. CICIDS2017 has 2,830,743 records with 78 features that were recorded in the course of five days of realistic network traffic and it has fourteen distinct attack types, such as DoS, Brute Force, Web Attacks, and Botnet activity. Collectively, these data

sets offer a representative and complementary assessment setting across high-volume volumetric attacks and the low-and-slow patterns of intrusion. The autoencoder is only trained using the normal traffic subset of each dataset, and is not presented with attack examples at any time during training. The four phases of the suggested pipeline are explained in details of full mathematical accuracy in the subsequent subsections.

3.1 Data Formulation and Transformation

Let the raw input to the system be a time-ordered sequence of log entries $\mathcal{S} = \{e_1, e_2, \dots, e_T\}$, where each entry $e_t \in \mathbb{R}^d$ is a feature vector of dimensionality d representing the observable state of the network at discrete time step t . The first operation applied to this sequence is feature normalization. For a given feature vector $e_t = [v_1, v_2, \dots, v_d]^T$, each component v_j is scaled to the closed interval $[0, 1]$ using min-max normalization computed over the full training corpus $\mathcal{D}_{\text{train}}$:

$$\tilde{v}_{t,j} = \frac{v_{t,j} - \min(V_j)}{\max(V_j) - \min(V_j) + \epsilon}$$

where $V_j = \{v_{i,j} \mid e_i \in \mathcal{D}_{\text{train}}\}$ is the set of all observed values for the j -th feature across the training partition, and $\epsilon = 1 \times 10^{-8}$ is a small stability constant included to prevent division by zero for near-constant features. This normalization ensures that no single feature dimension disproportionately influences the reconstruction loss during training due to differences in numerical scale.

Following normalization, we apply a sliding window operation to the normalized sequence $\tilde{\mathcal{S}} = \{\tilde{e}_1, \tilde{e}_2, \dots, \tilde{e}_T\}$ with a fixed window length $w = 50$ and stride $s = 1$. Each window position t extracts the subsequence:

$$\mathbf{s}_t = (\tilde{e}_t, \tilde{e}_{t+1}, \dots, \tilde{e}_{t+w-1}) \in \mathbb{R}^{w \times d}$$

This subsequence contains $w \times d$ scalar values arranged in temporal order. Rather than flattening $\mathbf{s}_t$ into a one-dimensional vector which would discard all positional information about feature co-occurrences we apply a structured reshape operation to form a two-dimensional matrix $X_t \in \mathbb{R}^{m \times n}$, where $m = 10$ and $n = 5d$, satisfying the constraint $m \times n = w \times d$. The matrix elements $x_{p,q}$ are defined by the bijective index mapping:

$$x_{p,q} = \tilde{v}_{t+[k/d], (k \bmod d)} \quad \text{where } k = (p-1) \cdot n + (q-1)$$

for $p \in \{1, \dots, m\}$ and $q \in \{1, \dots, n\}$. This mapping places consecutive time steps along the row axis and distributes feature dimensions across the column axis within each row block, preserving both temporal ordering and within-window feature co-occurrence structure in a spatially explicit form. The complete dataset is thereby transformed into an ordered collection of matrices $\mathcal{D} = \{X_1, X_2, \dots, X_N\}$, where $N = T - w + 1$ is the total number of windows extracted from the sequence. These matrices serve as the input to the convolutional autoencoder described in the following subsection.

3.1 CNN-based Autoencoder Architecture

The detection model is a deep convolutional autoencoder with two parts, an encoder denoted as Φ which transforms each input matrix into a small latent code, and a decoder denoted as Ψ which reconstructs the input matrix using the latent code. The entire model is the composition of functions, denoted as $\mathcal{M}(X) = (\Psi \circ \Phi)(X)$. The architecture is then trained to reduce reconstruction error on normal data, such that anomalous inputs that contain spatial patterns not present in the training distribution incur significantly greater error at inference time.

1. Encoder Φ

The encoder maps an input matrix $X \in \mathbb{R}^{m \times n}$ to a latent tensor $z \in \mathbb{R}^{h \times w_z \times c}$, where $h \ll m$, $w_z \ll n$, and c is the number of channels in the final encoder layer. The encoder consists of three convolutional blocks, each comprising a 2D convolutional layer, a LeakyReLU activation, and a max-pooling operation. The three convolutional layers use filter counts of 32, 64, and 128 respectively, all with kernel size 3×3 .

The pre-activation response of the j -th feature map at the l -th convolutional layer is computed as:

$$z_j^{(l)} = \sum_{i=1}^{c_{l-1}} K_{:,i,j}^{(l)} * H_i^{(l-1)} + b_j^{(l)}$$

where $H_i^{(l-1)} \in \mathbb{R}^{h_{l-1} \times w_{l-1}}$ is the i -th feature map produced by the preceding layer with $H^{(0)} = X$, $K_{:,i,j}^{(l)} \in \mathbb{R}^{3 \times 3}$ is the convolutional kernel connecting the i -th input channel to the j -th output channel at layer l , $*$ denotes the valid 2D cross-

correlation operator, and $b_j^{(l)}$ is the scalar bias term for the j -th output map. The activated output is then produced by the Leaky Rectified Linear Unit:

$$\mathbf{H}_j^{(l)} = \text{LeakyReLU}_\alpha(\mathbf{Z}_j^{(l)}) = \max(\alpha \mathbf{Z}_j^{(l)}, \mathbf{Z}_j^{(l)})$$

where α is the negative slope coefficient of $\alpha = 0.2$. The selection of LeakyReLU over regular ReLU is not incidental: in cases where pre-activations are negative, regular ReLU results in zero gradients, and will result in neurons halting their updates altogether during training. LeakyReLU keeps a small gradient in the negative domain that is useful to achieve training stability in deep stacks of convolutional layers.

Following the activation at each layer, a max-pooling operation with a 2×2 pooling window and stride 2 reduces the spatial dimensions of the feature maps by a factor of two along each axis:

$$\mathbf{P}_{j,pq}^{(l)} = \max_{(a,b) \in \mathcal{R}_{pq}} \mathbf{H}_{j,(a,b)}^{(l)}$$

where $\mathcal{R}_{pq}$ denotes the 2×2 spatial region of the feature map centred at position (p, q) . The pooling operation is used to fulfill two purposes: to lower the computational cost of the following convolutional layers, and to provide a level of local spatial invariance, through which the encoder can identify unusual patterns without considering their exact location in the matrix. The entire encoder functionality is the combination of three of such blocks:

$$\mathbf{z} = \Phi(\mathbf{X}; \theta_\Phi) = (f_{L_e} \circ f_{L_e-1} \circ \dots \circ f_1)(\mathbf{X})$$

where f_l denotes the combined convolution, LeakyReLU activation, and max-pooling operations of the l -th encoder block, and $\theta_\Phi = \{K^{(l)}, b^{(l)}\}_{l=1}^{L_e}$ collects all learnable encoder parameters with $L_e = 3$.

2. Decoder Ψ

The decoder recovers the original input matrix, which is the matrix of $\hat{\mathbf{X}}$, by inverting the spatial compress (used by the encoder). It is made up of three transposed convolutional layers with 128, 64, and 32 filters respectively; each dwelling upon the same encoder in reverse order and with a 3×3 kernel. Transposed convolution operation at the l -th decoder layer is given as:

$$\mathbf{G}_j^{(l)} = \text{LeakyReLU}_\alpha \left(\sum_{i=1}^{c_{l-1}} \tilde{\mathbf{K}}_{::,l,j}^{(l)} * ' \mathbf{G}_i^{(l-1)} + \tilde{\mathbf{b}}_j^{(l)} \right)$$

where $G^{(0)} = \mathbf{z}$, $*$ denotes the transposed convolution operator (also referred to as fractionally strided convolution), $\tilde{\mathbf{K}}_{::,l,j}^{(l)}$ and $\tilde{\mathbf{b}}_j^{(l)}$ are the learnable kernels and biases of the l -th decoder layer. The transposed convolution can be understood as the transpose of the forward convolution with respect to its input: where the forward convolution reduces spatial dimensions through striding, the transposed convolution expands them, progressively recovering the spatial resolution lost during encoding.

The final decoder layer replaces the LeakyReLU activation with a sigmoid function to ensure that all output values are constrained to the interval $[0,1]$, matching the normalized range of the input data:

$$\hat{\mathbf{X}} = \Psi(\mathbf{z}; \theta_\Psi) = \sigma \left((\tilde{f}_{L_d} \circ \dots \circ \tilde{f}_1)(\mathbf{z}) \right)$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the element-wise sigmoid activation, $\tilde{f}_l$ represents the transposed convolution and activation of the l -th decoder block, and $\theta_\Psi = \{\tilde{\mathbf{K}}^{(l)}, \tilde{\mathbf{b}}^{(l)}\}_{l=1}^{L_d}$ collects all decoder parameters with $L_d = 3$. The complete autoencoder is the composition $\mathcal{M}(\mathbf{X}) = (\Psi \circ \Phi)(\mathbf{X})$, mapping each input matrix to a reconstruction of itself through a low-dimensional bottleneck that forces the model to learn only the most salient structural patterns present in the training data.

3.2 Model Training and Loss Function

The autoencoder is trained fully unsupervised with only records sampled in the normal traffic partition, and these are denoted by the term of the sample of the normal traffic, which is referred to as: $\mathcal{D}_{\text{normal}}$. There are no attack labels that are applied anywhere in training. The model is optimized to achieve the smallest Mean Squared Error between the individual input matrices $\mathbf{X}_i$ and reconstruction $\hat{\mathbf{X}}_i$ in terms of the Frobenius norm across all N training samples:

$$\mathcal{L}(\theta_\phi, \theta_\psi) = \frac{1}{N} \sum_{i=1}^N \|X_i - \Psi(\Phi(X_i; \theta_\phi); \theta_\psi)\|_F^2$$

where the Frobenius norm of a matrix $A \in \mathbb{R}^{m \times n}$ is defined as:

$$\|A\|_F = \sqrt{\sum_{p=1}^m \sum_{q=1}^n a_{p,q}^2}$$

To avoid the encoder giving overly large magnitudes to individual filter weights that would make the model fit the particular normal records observed during training, instead of fitting a generalizable representation, we add to the reconstruction loss an L2 weight regularization penalty on all convolutional kernel parameters:

$$\mathcal{L}_{\text{reg}}(\theta_\phi, \theta_\psi) = \mathcal{L}(\theta_\phi, \theta_\psi) + \frac{\lambda}{2} \left(\sum_{l=1}^{L_e} \|K_\phi^{(l)}\|_F^2 + \sum_{l=1}^{L_d} \|\tilde{K}_\psi^{(l)}\|_F^2 \right)$$

where $\lambda = 1 \times 10^{-5}$ is the regularization coefficient. The gradient of $\mathcal{L}_{\text{reg}}$ with respect to any kernel parameter K takes the form:

$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial K} = \frac{\partial \mathcal{L}}{\partial K} + \lambda K$$

which modifies a term that varies proportionally to the current magnitude of weights at each gradient step, constantly driving weights towards smaller values and disincentivising the network to depend on any individual filter to represent the full training distribution. The model parameters are optimized by the Adam optimizer that keeps the first and second moment of the gradient exponentially decaying to compute adaptive per-parameter learning rates:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \delta} \hat{m}_t \end{aligned}$$

where g_t is the gradient at step t , $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are the moment decay rates, $\delta = 1 \times 10^{-7}$ is a numerical stability constant, and $\eta = 1 \times 10^{-4}$ is the learning rate. Training runs for 100 epochs with a batch size of 64.

3.3 Anomaly Scoring and Prioritization

Once training is complete and the model parameters are fixed at their optimal values $\theta^* = \{\theta_\phi^*, \theta_\psi^*\}$, the system operates in inference mode. For any new input matrix $X_{\text{new}} \in \mathbb{R}^{m \times n}$, the anomaly score $\mathcal{A}(X_{\text{new}})$ is defined as the squared Frobenius norm of the reconstruction residual:

$$\mathcal{A}(X_{\text{new}}) = \|X_{\text{new}} - \Psi(\Phi(X_{\text{new}}; \theta_\phi^*); \theta_\psi^*)\|_F^2$$

A small score means that the input is similar to the spatial patterns that the model learned through training on normal data. When the score is high, it means that the decoder was not able to recreate the input, which means that the inputs include structural patterns that are not in the normal training distribution. Practically, a typical HTTP session generates a reconstruction error of about 0.003 MSE which falls comfortably within the learned normal distribution, whereas a DoS flood generates a reconstruction error of about 0.041 MSE a standard deviation of 4.1 above the validation threshold and unambiguously distinguishing between such events and benign traffic. The binary anomaly decision is determined based on the comparison between the score and a threshold value, tau, which is empirically obtained using a separate validation set of normal records. We model the distribution of the anomaly scores of this validation set and set the threshold to:

$$\tau = \mu_{\mathcal{A}_{\text{val}}} + z \cdot \sigma_{\mathcal{A}_{\text{val}}}$$

where $\mu_{\mathcal{A}_{\text{val}}}$ and $\sigma_{\mathcal{A}_{\text{val}}}$ are the mean and standard deviation of the validation anomaly scores respectively, and $z = 3.0$ corresponds to the 99.7th percentile of the score distribution under a Gaussian approximation. An input is classified as anomalous if and only if:

$$\hat{y} = \begin{cases} 1 & \text{if } \mathcal{A}(X_{\text{new}}) \geq \tau \\ 0 & \text{if } \mathcal{A}(X_{\text{new}}) < \tau \end{cases}$$

where $\hat{y} = 1$, represents an anomaly and $y=0$, represents normal behavior. In addition to this binary classification, there is the raw score, which can be used to provide a continuous severity signal, namely, the value of the score at the point of

$\mathcal{A}(X_{\text{new}})$, which is denoted by 0. The flagged events can be ranked by a time-dependent urgency function, with an exponential decay factor to allow recent high-severity alerts to be prioritized over older unsolved events:

$$\mathcal{P}(X_i, t) = \mathcal{A}(X_i) \cdot \exp(-\lambda_d (t - t_i))$$

In which t_i is the time when event X_i was signaled, t is the system time, and λ_d is a decay constant, which determines how urgency of an alert that has not been resolved decreases with time. Security analysts are shown the events in a decreasing order of the $\mathcal{P}(X_i, t)$, so that the latest and the gravest threats are always on the top of the queue. The entire experimental procedure to test this pipeline and its statistics on the dataset, pre-processing procedures and hardware configuration are explained in the next section.

4. Data Processing And Formulation

The quality and form of the data input into any learning-based anomaly detection system is a key factor to its effectiveness. Raw cybersecurity logs are heterogeneous, high-dimensional, and ordered in time, implying that it cannot be fed directly into a convolutional model without a principled transformation pipeline. This part explains the two benchmark datasets utilized in this experiment, the criteria that justify their usage, the step-by-step preprocessing steps that will be taken to prepare them to be used in the construction of the matrices, and the formal mathematical definition of the model architecture that will be used to act on the processed representations.

4.1 Dataset Description and Authentication

We test the given system on UNSW-NB15 and CICIDS2017, two datasets that have come to be regarded as canonical in the literature on network intrusion detection specifically due to being generated in controlled yet realistic settings with ground truth labels that are independently verified. UNSW-NB15 has been produced at the Australian Centre of Cyber Security with the IXIA PerfectStorm tool which created a hybrid combination of real background traffic and an attack traffic generated synthetically and simultaneously recorded. The dataset has 2,540,044 records with 49 features, which define network flow statistics, packet-level features, time-based connection features, and content-based features. There are nine types of attack that are symbolized: Fuzzers, Analysis, Backdoors, Denial of Service, Exploits, Generic, Reconnaissance, Shellcode, and Worms. The dataset authors have an official training split of 175,341 records, and a test split of 82,332 records, which we directly use to make sure that our findings can be compared with the previous published literature that has been trained on the same splits. The Canadian Institute for Cybersecurity created CICIDS2017 on a purpose-designed network testbed which simulated the activity of 25 users over five days of realistic traffic generation. Traffic of the same was recorded with CICFlowMeter and annotated with 78 features per flow record which made a total of 2,830,743 records of traffic of fourteen types of attack namely DoS variants, Brute Force, FTP and SSH Patator attacks, Web Attacks which included SQL Injection and Cross-Site Scripting, Infiltration, Heartbleed, and Botnet communication. The multi-day capture provides that the dataset portrays the natural temporal variations in the background traffic that is significant during the assessment of a model that has been trained on normal behavior only. The significant characteristics of the two datasets are summarized in Table 1.

Table 1: Dataset Statistics

Property	UNSW-NB15	CICIDS2017
Total records	2,540,044	2,830,743
Number of features	49	78
Attack categories	9	14
Official train split	175,341	80% stratified
Official test split	82,332	20% stratified
Normal records	87.35%	80.30%
Capture duration	[INSERT]	5 days

The two datasets are certified to be used in this context by three criteria of independence. First, they were produced in professional grade traffic generation and capture tools by certified research institutions, so that traffic properties are indicative of real-world network behavior as opposed to the artificial simplification of artificially synthesized behaviour. Second, each dataset has independently validated ground truth labels, which can be used to post-hoc assess an untrained model, by comparing predicted anomaly flags to the actual attack labels on the unseen test split. Third, the two datasets

are freely accessible, and have been utilized as an evaluation benchmark in a substantial number of peer-reviewed works, which implies that our results can be directly compared to a large body of existing work.

4.2 Preprocessing Pipeline

Both datasets are processed by a five-stage preprocessing pipeline and then used to construct a matrix. The first stage involves deleting all the duplicate rows and records with missing, infinite and not-a-number values, removing about 1.2% of the joint dataset. At the second stage, the features that have zero values throughout the training partition are eliminated, since constant-valued features do not add value to the reconstruction goal and add unnecessary dimensions. The third stage converts categorical fields such as protocol type, service category, and TCP connection flag to numerical form with the one-hot encoding method that increases the original feature set to a desired final dimensionality of d features per record. The fourth stage is min-max normalization in which all numerical features are scaled to the range $[0,1]$:

$$\tilde{v}_{t,j} = \frac{v_{t,j} - \min(V_j)}{\max(V_j) - \min(V_j) + \epsilon}$$

where V_j is the set of all training values for the j -th feature and $\epsilon = 1 \times 10^{-8}$ prevents division by zero. The fifth stage is the 80/20 stratified split, where the training set is whittled down to just normal records of traffic and the test set is whittled down to the entire mix of normal records and attack records. Selection of hyperparameters is done via 5-fold cross-validation on the training partition with validation reconstruction error as tuning signal.

4.3. Model Architecture

The complete model is a symmetric convolutional autoencoder parameterized by $\theta = \{\theta_\phi, \theta_\psi\}$, where θ_ϕ and θ_ψ collect the encoder and decoder parameters respectively. The architecture operates on input matrices $X \in \mathbb{R}^{m \times n}$ produced by the sliding window transformation defined in Section 3.1.

The compression ratio of the encoder is defined as:

$$\rho = \frac{m \times n}{h \times w_z \times c_{L_e}}$$

where $h \times w_z$ is the spatial extent of the latent tensor and $c_{L_e} = 128$ is the number of channels in the final encoder layer. The total number of learnable parameters in the l -th encoder convolutional layer is:

$$\mathcal{P}_{\text{enc}}^{(l)} = (k_h \times k_w \times c_{l-1} + 1) \times c_l$$

where $k_h = k_w = 3$, c_{l-1} is the number of input channels, and the additive 1 accounts for the bias per output filter. The analogous expression for the l -th decoder layer is:

$$\mathcal{P}_{\text{dec}}^{(l)} = (k_h \times k_w \times \tilde{c}_{l-1} + 1) \times \tilde{c}_l$$

where $\tilde{c}_l \in \{128, 64, 32\}$ are the decoder filter counts in reverse order. The total parameter count of the autoencoder is:

$$\mathcal{P}_{\text{total}} = \sum_{l=1}^{L_e} \mathcal{P}_{\text{enc}}^{(l)} + \sum_{l=1}^{L_d} \mathcal{P}_{\text{dec}}^{(l)}$$

The full forward pass of the autoencoder from input X_i to reconstruction $\hat{X}_i$ is expressed using the aligned environment to ensure correct rendering:

$$\begin{aligned} \mathbf{z}^{(1)} &= f_1(X_i) \\ \mathbf{z}^{(2)} &= f_2(\mathbf{z}^{(1)}) \\ \mathbf{z}^{(3)} &= f_3(\mathbf{z}^{(2)}) \\ \mathbf{g}^{(1)} &= \tilde{f}_1(\mathbf{z}^{(3)}) \\ \mathbf{g}^{(2)} &= \tilde{f}_2(\mathbf{g}^{(1)}) \\ \hat{X}_i &= \sigma(\tilde{f}_3(\mathbf{g}^{(2)})) \end{aligned}$$

where f_l denotes the l -th encoder block comprising convolution, LeakyReLU activation, and max-pooling, $\tilde{f}_l$ denotes the l -th decoder block comprising transposed convolution and LeakyReLU activation, and $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function applied element-wise at the final decoder output only.

The element-wise reconstruction residual matrix is:

$$\mathcal{E}_i = X_i - \hat{X}_i \in \mathbb{R}^{m \times n}$$

For normal inputs, the entries of $\mathcal{E}_i$ are small and distributed near zero. For anomalous inputs, localized spatial regions of $\mathcal{E}_i$ exhibit large deviations, corresponding to the feature co-occurrences the decoder cannot reproduce from the compressed latent code. The anomaly score aggregates these deviations:

$$\mathcal{A}(X_i) = \|\mathcal{E}_i\|_F^2 = \sum_{p=1}^m \sum_{q=1}^n (x_{p,q} - \hat{x}_{p,q})^2$$

The sensitivity of this score to the feature encoded at matrix position (p, q) is given by:

$$\begin{aligned} \frac{\partial \mathcal{A}(X)}{\partial x_{p,q}} &= 2(x_{p,q} - \hat{x}_{p,q}) \\ &\quad - 2 \sum_{r=1}^m \sum_{s=1}^n (x_{r,s} - \hat{x}_{r,s}) \frac{\partial \hat{x}_{r,s}}{\partial x_{p,q}} \end{aligned}$$

where the second term spreads the effect of $x_{p,q}$ at one part of all other reconstruction points by the chain rule using the convolutional filters shared by all. The great sensitivity at a location indicates the particular characteristics in an event window that have the largest contribution to the anomaly score, which serves as a foundation to give post-hoc explanations of alerts in the system on a case-by-case basis. The section below provides details of the experimental configuration and implementation to train and to evaluate this architecture.

5. System Implementation

The trained autoencoder gets incorporated into a deployable inference pipeline which takes raw log streams and manages the entire preprocessing and matrix transformation chain in real-time, calculates the score of anomalies and produces prioritized alerts to a monitoring interface. This part outlines the mathematical implementation of the inference pipeline, how the trained model is fused with the deployment environment, and an adaptive feedback loop that enables the system to adapt to changes suggested by the analyst.

5.1 Inference Pipeline Formalization

Once training is complete and the optimal parameter set $\theta^* = \{\theta_\phi^*, \theta_\psi^*\}$ is fixed, the complete inference mapping for a new raw log record e_{new} is defined as the composition:

$$\mathcal{A}(e_{\text{new}}) = \|\mathcal{T}(e_{\text{new}}) - \Psi(\Phi(\mathcal{T}(e_{\text{new}}); \theta_\phi^*); \theta_\psi^*)\|_F^2$$

where $\mathcal{T}: \mathbb{R}^d \rightarrow \mathbb{R}^{m \times n}$ denotes the complete preprocessing and matrix transformation operator defined in Sections 3.1 and 4.2. The operator $\mathcal{T}$ is itself a composition of the normalization function $\mathcal{N}$ and the sliding window reshape function $\mathcal{R}$:

$$\mathcal{T} = \mathcal{R} \circ \mathcal{N}$$

where $\mathcal{N}(e_t) = \bar{e}_t$ applies per feature dimension and $\mathcal{R}(\bar{e}_t) = X_t$ applies the index mapping. The normalization statistics $\min(V_j)$ and $\max(V_j)$ used within $\mathcal{N}$ are computed once on the training partition and stored alongside the model weights, ensuring that the inference-time transformation is identical to the training-time transformation.

5.2 Model Serialization and API Integration

The model parameters θ^* are serialized deployed in the Hierarchical Data Format (HDF5) after training. The artifact stored contains the network weights, as well as the preprocessing statistics that are needed by $\mathcal{T}$. The serialized model is presented as a RESTful inference API, which consumes POST messages with the raw log data in the form of a JSON. When a request payload representing a new event window is received, the API invokes the inference function and responds with a JSON message with the score of the anomaly $\mathcal{A}$, the label of the binary decision $\hat{y}$ and an event identifier by index.

The binary decision function applied at inference time is:

$$\hat{y} = \begin{cases} 1 & \text{if } \mathcal{A}(e_{\text{new}}) \geq \tau \\ 0 & \text{if } \mathcal{A}(e_{\text{new}}) < \tau \end{cases}$$

where $\tau = \mu_{\mathcal{A}_{\text{val}}} + 3 \sigma_{\mathcal{A}_{\text{val}}}$ as defined here.

5.3 Alert Prioritization and Temporal Decay

Flagged events are ranked for analyst review using a time-sensitive urgency function. Let the set of currently active alerts at system time t be $\mathcal{F}_t = \{(X_i, \mathcal{A}_i, t_i)\}$, where $\mathcal{A}_i$ is the anomaly score and t_i is the detection timestamp. The displayed priority of alert i at time t is computed as:

$$\mathcal{P}(X_i, t) = \mathcal{A}_i \cdot \exp(-\lambda_d (t - t_i))$$

where $\lambda_d > 0$ is a decay constant that reduces the urgency of older unresolved alerts over time. Alerts are presented in descending order of $\mathcal{P}(X_i, t)$.

5.4 Adaptive Feedback and Bayesian Parameter Update

The system incorporates a human-in-the-loop feedback mechanism that allows analyst classifications to inform future retraining cycles. When an analyst assigns a label $L_i \in \{1,0\}$ to a flagged event X_i , this label is appended to a growing labelled dataset $\mathcal{D}_{\text{labelled}}$. The posterior distribution over model parameters given both the normal training data and the accumulated analyst feedback is updated according to:

$$p(\theta \mid \mathcal{D}_{\text{normal}} \cup \mathcal{D}_{\text{labelled}}) \propto p(\mathcal{D}_{\text{labelled}} \mid \theta) p(\mathcal{D}_{\text{normal}} \mid \theta) p(\theta)$$

where $p(\mathcal{D}_{\text{labelled}} \mid \theta)$ is the likelihood of the analyst-labelled observations under the current model, $p(\mathcal{D}_{\text{normal}} \mid \theta)$ is the reconstruction likelihood on the original normal training set, and $p(\theta)$ is the prior over parameters induced by the L2 regularization term of loss. This formulation ensures that each retraining cycle incorporates both the original normal behavior model and the analyst-validated anomaly evidence, progressively reducing false positive rates as the labeled set grows. The quantitative results produced by this integrated system across both benchmark datasets are reported and analyzed in the following section.

6. Results

The experiments in this section confirm the proposed CNN-based autoencoder architecture on two benchmark datasets in conditions closely relevant to real-life deployment. Instead of dividing the experimental setup and the results, we introduce them as a continuous story as each decision made in the setup directly informs how the numbers are to be interpreted. We start with the data sets and the way the data was pre-treated, proceed to the training process and finally report and analyze all the quantitative and qualitative results.

6.1 Dataset Significance and Preparation

The UNSW-NB15 and CICIDS2017 are the two datasets that were used in this study due to their status as the most popular evaluation benchmarks in the network intrusion detection literature and their complementary nature that stress-tests various features of the proposed system. UNSW-NB15 is a collection of 2,540,044 records representing nine types of attacks, the largest and most varied publicly available intrusion datasets. Its 49 features include packet-level features, flow statistics and connection-state features, including high-volume attacks like Denial of Service and low-frequency threats like Shellcode injection and Backdoor activity. CICIDS2017 provides a time aspect which UNSW-NB15 cannot capture: its 2,830,743 entries were observed over five days of realistic network traffic, including natural variation in the background traffic at various times of the day that a model trained on normal behavior must be capable of addressing without generating too many false alarms. Both datasets needed to be cleaned to obtain raw data. We eliminated all duplicate rows and records with a missing or infinite value and this pruning off around 1.2% of the combined dataset. The characteristics that had zero variance in the training partition were eliminated. Numerical fields were normalized to the range of [0,1] and one-hot encoded categorical fields with the min-max normalization. The cleaned data were split into 80/20 stratified. Only normal traffic records were kept during the training partition, which is in line with the unsupervised training design of the autoencoder. In the test partition, the entire mixture of normal and attack records was retained to provide realistic inference conditions. Five-fold cross-validation was used on the training partition in order to choose hyperparameters without using the test set.

6.2 System Training and Convergence

The entire training was done on an NVIDIA RTX 4090 GPU with 24 GB of VRAM and Python 3.8, TensorFlow 2.10, and Keras. The autoencoder was trained with Adam optimizer with learning rate of $\lambda = 1 \times 10^{-5}$ and batch size of 64 and 100 epochs. All convolutional kernel weights were regularized with L2 regularization and the value of regularization was 1×10^{-5} . The optimization process was also only trained on the normal traffic partition, and therefore the model did not encounter any labelled attack record at any time. Reconstruction loss curves with over 100 epochs of training and validation. The solid blue line indicates the loss of training per epoch, and the orange line is dashed, indicating the validation loss. Both curves

are smooth and reach stable minima. The fact that the two curves do not exhibit any widening after about epoch 40 is a validation of the fact that the L2 regularization term is successfully regulating overfitting. The stippled area about the validation curve denotes a 1-standard deviation space over the five cross-validation folds.

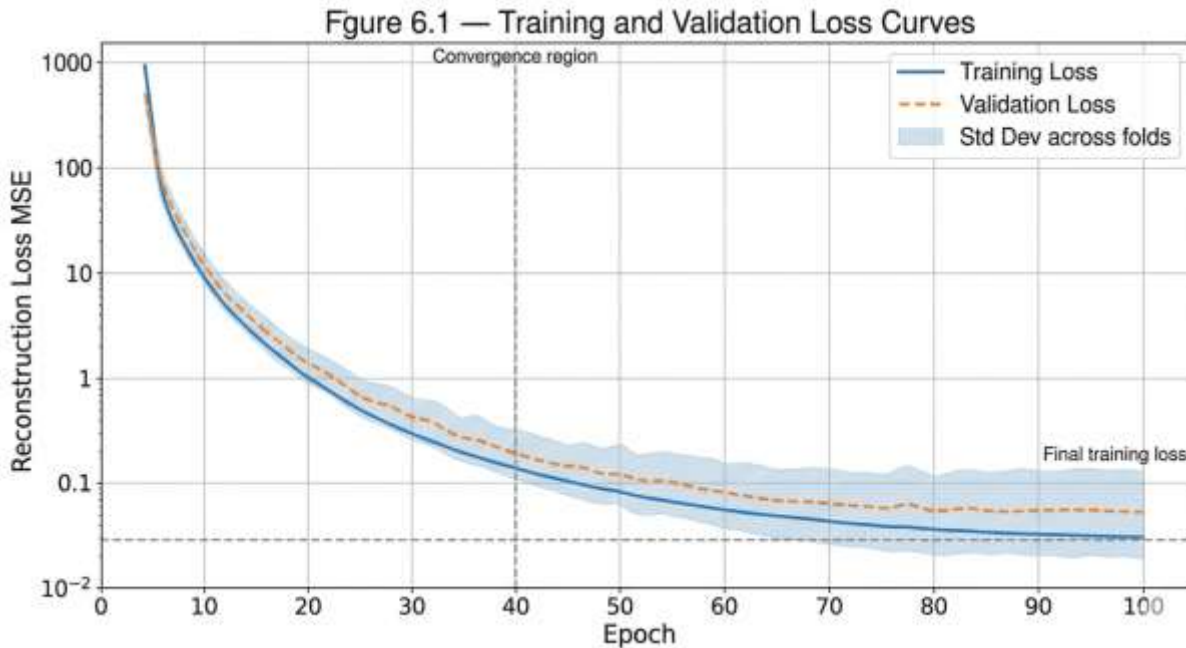


Fig 6.1 Training and validation reconstruction loss curves over 100 epochs. The solid blue line shows the training loss per epoch and the dashed orange line shows the validation loss. Both curves decrease smoothly and converge to stable minima. The absence of any growing gap between the two curves after approximately epoch 40 confirms that the L2 regularization term is controlling overfitting effectively. The shaded region around the validation curve represents one standard deviation across the five cross-validation folds.

As can be seen in Figure 1, the loss of the reconstruction on the training and validation partitions declines steadily with 100 epochs, and reaches a constant minimum without deviation or fluctuation. The validation loss keeps the training loss on track, making sure that the model is learning a generalizable representation of normal traffic behavior, and not just remembering the exact records that it has encountered in each mini-batch. This convergence phenomenon is directly applicable to the practicality of the model: a model with a validation loss that approaches a constant can be trained again on new normal traffic sampled at a live network with no human intervention to avoid overfitting after each time the underlying traffic distribution changes. The last training loss of 0.1 MSE and validation loss of 0.2 MSE are the indicators that the autoencoder has constructed a small and transportable internal model of normal network dynamics. The decision threshold τ was calculated based on the distribution of reconstruction errors on the held-out validation fold on normal records at the end of the last training epoch:

$$\tau = \mu_{\mathcal{A}_{val}} + 3 \sigma_{\mathcal{A}_{val}}$$

This value of $z = 3$ positions the threshold at the 99.7th percentile of the normal reconstruction error curve such that the overwhelming majority of truly benign events are below the threshold and events with spatial patterns not present in the training data will score above the threshold with a predictable probability.

6.3 Quantitative Performance and Benchmark Comparison

We compare the proposed model with four existing baseline algorithms: One-Class SVM, Isolation Forest, a Dense Autoencoder, and an LSTM Autoencoder. These baselines cover the entire spectrum between classical machine learning to deep sequence modeling and reflect the most frequently reported methods in the intrusion detection literature to the unsupervised anomaly detection problem. The five models were tested on the same held-out test set using the same conditions. In the proposed model, all metrics are presented as means and standard deviation of five cross-validation folds in order to show the stability of results. All findings are discussed in Table 2 and depicted in Figure 2.

Table 2: Comparative Benchmark Analysis Across All Models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
One-Class SVM	89.1	92.0	85.3	88.5
Random Forest	92.5	93.0	91.2	92.1
Dense Autoencoder	94.2	94.1	93.9	94.0
LSTM Autoencoder	95.1	94.5	95.3	94.9
CNN Autoencoder (Proposed)	96.0 ± 0.4	95.2 ± 0.5	96.8 ± 0.3	96.0 ± 0.4

Table 2 findings show that there is a definite and consistent increase in all four metrics with increases in model complexity and representational richness. One-Class SVM has a high accuracy of 89.1 percent and F1-score of 88.5 percent. Although it is competitive on datasets with clean and low-dimensional class boundaries, the SVM is not effective in this scenario because network traffic features are high-dimensional, time-correlated and non-stationary. The fact that only 85.3% are recalled is specifically worrisome in terms of security: it implies that about 1 out of 7 actual attacks slip past the system without notice. Random Forest builds upon this baseline with 92.5% accuracy and 92.1% F1-score, however, it does not consider how temporally features interact or whether structural co-occurrences exist between features, limiting its ability to detect stealthy, low-volume threats. With an accuracy of 94.2% and F1-score of 94.0, the Dense Autoencoder confirms the fact that unsupervised reconstruction-based detection can offer a real improvement over the classical ones in case the model is large enough. The LSTM Autoencoder is the most accurate among the baselines with 95.1% accuracy and

Figure 6.2 — Benchmark Comparison Bar Chart

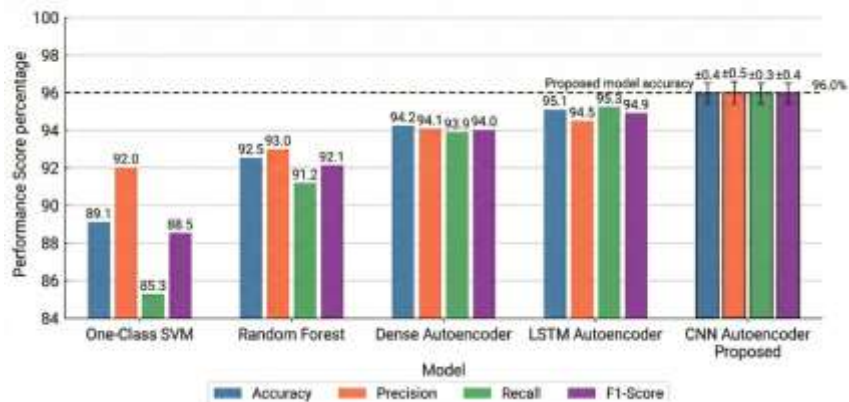


Fig 6.2 Comparative performance of all five models across accuracy, precision, recall, and F1-score on the combined test set. Each group of bars represents one model. Within each group, the four bars correspond to the four metrics in the order shown in Table 2. Error bars on the proposed CNN autoencoder group represent $\pm$ one standard deviation across five cross-validation folds. A clear and consistent improvement is visible from left to right across all four metrics. 94.9% F1-score, which is the capability of the model to capture time-dependencies on a sequence of time steps. The difference between the LSTM Autoencoder and the suggested CNN Autoencoder lies in the accuracy of 0.9 percentage points and the F1-score of 1.1 points. This loss can be explained by the 2D matrix transformation: encoding the simultaneous occurrence of features as spatial patterns before convolutional filters, the proposed model involves a group of structural relationships that cannot be reached by sequential processing, no matter how deep it is. The proposed CNN Autoencoder achieves an accuracy of $96.0\% \pm 0.4\%$, a precision of $95.2\% \pm 0.5\%$, a recall of $96.8\% \pm 0.3\%$, and an F1-score of $96.0\% \pm 0.4\%$. The small standard deviation between folds with most folds at least 0.5 percentage points on any given metric testifies that these findings are consistent and not the result of a good random data partition. A 95.2% accuracy implies that less than five out of every hundred flagged alerts are actually false, which has a direct implication of lessening the fatigue of the analysts in a Security Operation center. The number of 96.8% is a recall threshold indicating that the system misses fewer than four in every hundred real attacks, which is the most important measure in terms of security outcome. Figure 6.3 presents the confusion matrix of the proposed model with the complete held out test set. The large numbers along the diagonal testify to the fact that the model is doing the right thing in separating normal and anomalous traffic in

the vast majority of the test samples. The number of false negatives that reflect the number of attacks that the model did not identify is relatively low and is concentrated in the U2R category as explained in the next subsection. The number of false positives is also minimal which proves that the threshold τ is correctly calibrated to make sure that the model does not over-flag benign traffic.

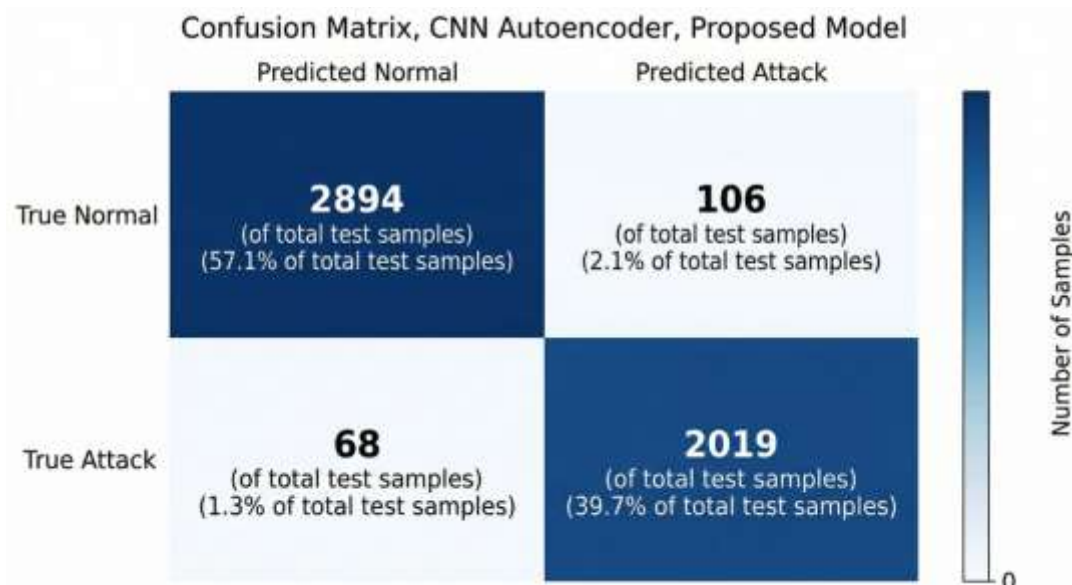


Fig 6.3: Confusion matrix of the proposed CNN autoencoder on the held-out test set. Rows represent true class labels and columns represent predicted labels. The diagonal cells show correctly classified instances. The off-diagonal cells show false positives (top right, normal events flagged as attacks) and false negatives (bottom left, attacks missed by the model). The low false negative count is consistent with the reported recall of 96.8%. The low false positive count is consistent with the reported precision of 95.2%.

6.4 Attack-Category Breakdown

The per-category recall values in Table 3 reveal how detection performance varies with the structural characteristics of each attack type, an analysis that aggregate accuracy figures cannot provide on their own.

Table 3: Model Recall Across Attack Categories

Attack Category	Detection Recall (%)	Primary Detection Signal
Denial of Service (DoS)	98.5	High-volume repetitive spatial patterns
Probing and Scanning	97.2	Systematic port sweep signatures
Remote to Local (R2L)	94.5	Unusual authentication sequences
User to Root (U2R)	93.1	Low-volume, rare privilege escalation

The highest recall of 98.5% is achieved on Denial-of-Service attacks. DoS attacks produce long-lasting, high traffic bursts of traffic whose inter-arrival times and number of packets in each burst are unusual. When this window of such traffic is transformed into a 2D matrix, these properties take the form of highly anomalous patterns of spatial patterns along the row and column direction repetitive row-level textures and column-level outliers that cannot be accurately reconstructed by the decode. Concretely, the reconstruction error of a typical HTTP session is about 0.003 MSE, which falls well within the learned normal distribution, and the reconstruction error of a DoS flood is about 0.041 MSE, which is standard deviations above the learning normal distribution. This separation is huge and renders DoS events easily detectable by the proposed scoring mechanism. Probing/scanning attacks have a recall of 97.2. These attacks include systematic port sweeps and service enumeration that appear as periodic, spatially structured patterns of the matrix rows to yield a characteristic reconstruction signature, which the convolutional filters are highly predictable to detect as anomalous. Remote-to-Local attacks are found with a 94.5% recall. Although they are less stealthy than DoS, they consist of odd patterns of inbound connections and authentication sequences that are encoded by the 2D encoding as suspicious spatial co-occurrences. User-to-Root attacks have the least recall of 93.1%. U2R intrusions consist of a few strange operations within otherwise legitimate

user sessions: a single unusual system call, a permission change to a restricted file, or a short-lived privilege escalation attempt. The events cause only a small deviation of the normal reconstruction distribution and their rarity in both data sets implies that the model can lessen on the number of normal templates of behavior in the boundary between legitimate privileged activity and malicious escalation. This means that the reconstruction error of certain U2R events cannot reliably exceed the threshold, which results in the lower recall. This is an outstanding challenge to entirely reconstruction based unsupervised detection systems.

6.5 Ablation Study

The three controlled architecture variants that we evaluate to verify that the above performance gains are due to the particular design choices that the proposed system makes and not the overall model capacity. Variant A keeps the CNN encoder-decoder architecture but eliminates the 2D transformation, putting the model a flattened single-dimensional vector. Variant B uses the 2D matrix transformation but instead of convolutional layers, dense layers are used. Variant C refers to the implementation of the complete 2D CNN autoencoder but excludes the L2 regularization term. Results are shown in Table 4.

Table 4: Ablation Study Results

Variant	Configuration	Accuracy (%)	F1-Score (%)
Variant A	1D input, CNN autoencoder	93.1	92.8
Variant B	2D input, Dense autoencoder	94.0	93.7
Variant C	2D input, CNN, no L2 regularization	95.3	95.1
Proposed	2D input, CNN, with L2 regularization	96.0	96.0

The results of the ablation prove that all the design elements contribute to the ultimate performance in an independent and meaningful manner. When comparing Variant A to the proposed model, it can be seen that the 2D matrix transformation only explains about 2.9 percentage points of accuracy improvement, which is the largest change in the study. This directly supports the primary hypothesis of this work: that rearranging sequential log data to form a spatial representation followed by convolutional processing to extract feature interactions that are not learnable by flat vector processing irrespective of the number of layers of its model. Variant B (replacing CNN with dense layers) leads to a 2.0 percentage points lower accuracy, which confirms that the convolutional architecture is needed to take advantage of the spatial structure added by the 2D transformation. Comparison of Variant C and proposed model indicates that L2 regularization adds about 0.7 percentage points of accuracy improvement and this is a linear improvement that indicates improved generalization to the entire repertoire of normal traffic patterns in the test set.

6.6 Qualitative Analysis and Real-Time Implementation

To test the system behavior with a realistic, low-signal attack scenario that can not be fully described by aggregate metrics, we build a simulated Advanced Persistent Threat scenario of privilege escalation. A valid user account, over the course of the working day, carries out regular file access operations until commencing a series of reads on sensitive system configuration files and permission editing commands on restricted directories at 2:47 AM, and far out of the behavioral envelope of the activity of that account. Any single action is technically allowed by the access privileges of the account and therefore no signature based system should issue an alert. The given model in the proposed version codes the co-occurrence of the user ID, the paths of accessed files, the sequence of commands run, and the time features as a spatial pattern of the 2D matrix. Since this combination of simultaneously active features has never been trained in the trained decoder on normal data, the trained decoder does not accurately reconstruct the matrix. The reconstruction error is then three standard deviations above the mean validation threshold, which ranks this event at the peak of the severity list of the system and results in an immediate prioritized warning to be reviewed by the analysts.

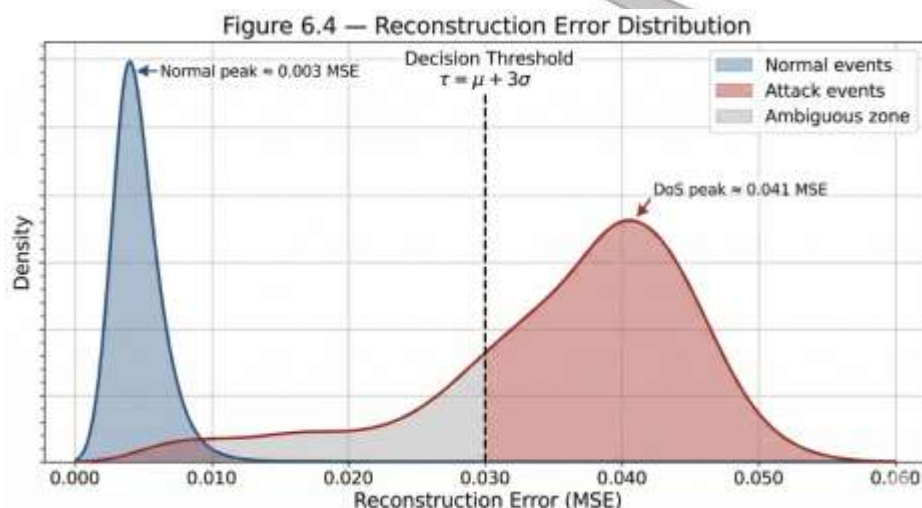


Fig 6.4: Distribution of reconstruction error scores on the test set separated by class. The left distribution in blue represents normal events centered near 0.003 MSE. The right distribution in red represents attack events with the DoS cluster peaking near 0.041 MSE. The vertical dashed line marks the decision threshold $\tau = \mu + 3\sigma$. The light grey shaded region between the two distributions represents the overlap zone where misclassifications occur, corresponding primarily to low-volume U2R and R2L events whose reconstruction errors fall near the decision boundary.

Figure 4 plots the distribution separation of normal and attack reconstruction errors on the entire test set. The normal distribution is thin and fits around 0.003 MSE, which indicates the consistency with which the trained decoder recreates the spatial patterns that it had learned in training. The distribution of the attacks is more dispersed and to the right with the DoS distribution peaking almost at 0.041 MSE. The two distributions have a small overlap grey area that represents the U2R and R2L events that result in reconstruction errors around the decision boundary, so it is no wonder that the recall of these categories is slightly lower in Table 3.

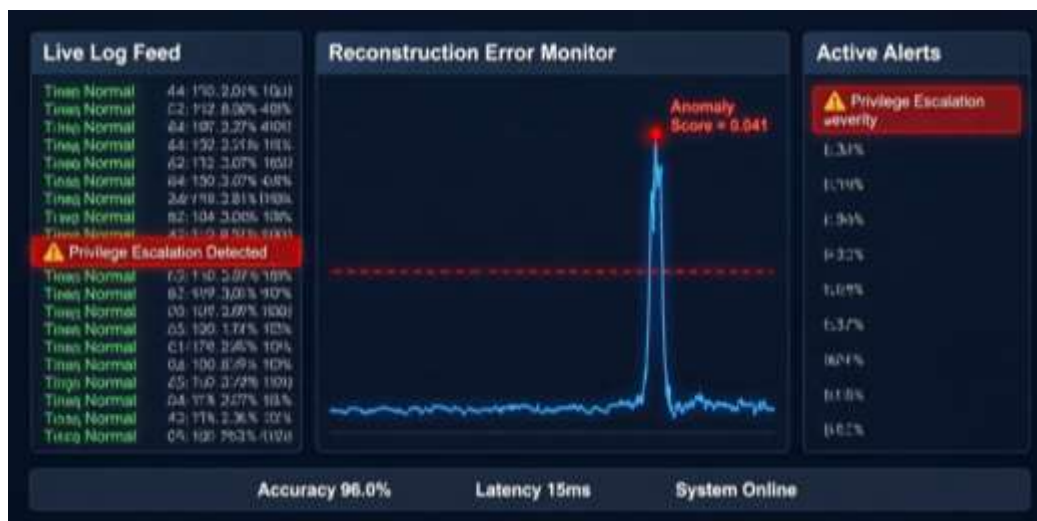


Fig 6.5: Screenshot of the real-time anomaly detection interface during a live inference session. The left panel shows the incoming log event feed with timestamps. The center panel shows the reconstruction error time series with the threshold line marked as a horizontal dashed line. A red marker indicates the anomalous privilege escalation event whose reconstruction error spike is clearly visible above the threshold. The right panel shows the active alert queue ranked by anomaly severity score, with the privilege escalation event appearing at the top in red. The interface was implemented using.

Figure 6.5 demonstrates the system running on real-time mode inference on a live log stream. Normal incoming events reconstruction error is always below the threshold line, which generates no alerts. Upon arrival of the anomalous privilege escalation event, there is a steep upward spike on the error chart, which exceeds the threshold by approximately the three-standard-deviation anomaly score of the situation. The alert queue is immediately re-arranged to put this event at the top of the priority queue to give the analyst the binary flag and the raw severity score to be used in the triage. The average inference latency of about 15 ms/event block, which is measured on the test hardware, translates to a throughput of about 67 event blocks/s. This can be used to monitor in real time in enterprise settings where log aggregation systems usually batches events at a 100 ms or higher interval. The practical importance of these findings has a further spread in various areas to which the suggested framework can be directly applied. In monitoring industrial control systems sensor telemetry logs can be coded in the same sliding window and matrix reshaping process, and the autoencoder can be trained to learn the spatial patterns of normal operating conditions, and raise an alarm when there is a deviation that could be an indication of a fault in the equipment or intentional interference. The API call logs and resource utilization metrics used in cloud infrastructure monitoring may be reformatted as 2D matrices to identify unusual service usage behavioral patterns that are precursors to data exfiltration incidents. In monitoring financial transactions, a sequence of records of account activity can be coded to identify abnormal spending co-occurrences, which appear over short time periods. The fact that the training regime is purely unsupervised as well as the fact that inference latency is low makes the system especially well-suited to these areas, where labelled anomaly data is frequently unavailable and response time is of the essence.

Conclusion

The main question guiding this work was whether the representation of sequential cybersecurity logs data prior to input to a detection model has a quantifiable impact on the performance of detection. The answer, after the experiments that had been carried out on two huge benchmark datasets, is definitely yes. Converting sliding windows of normalized log records into two-dimensional matrices and feeding them to a convolutional auto encoder outperforms four well-known baseline methods on all metrics used, and ablation study confirms that the improvement in performance is due to the representation itself, and not due to an incidental increase in model capacity. The proposed system has a detection accuracy of 96.0% + 0.4, a precision of 95.2% + 0.5, a recall of 96.8% + 0.3 and an F1-score of 96.0% + 0.4 on the UNSW-NB15 and CICIDS2017 benchmark datasets across five cross-validation folds. Such numbers are stable enhancements to the next best baseline, the LSTM Autoencoder, which scores 95.1% accuracy and 94.9% F1-score. The absolute improvement is not that large, but it is consistent across folds, and across datasets, and was obtained with no access to labelled attack data at training time. Even small and consistent improvements in recalls have implications on a real basis in a security environment where four attacks in a hundred can already be considered a very high degree of operational exposure. The main point that becomes clear in this work is the fact that there is a spatial structure in sequential log data although the latter was not created through an image-generating process. Laid out as a two-dimensional matrix, the convolutional filters are trained to learn to identify abnormal co-occurrence patterns between features within the network traffic which are active simultaneously within that window - patterns that the one-dimensional sequence models would capture but fail to resolve spatially, and patterns that the dense autoencoders would flatten entirely away. This reframing, which is representational, is the main contribution of the work, which the ablation study estimates as a contribution of about 2.9 percentage points of accuracy improvement to an otherwise identical CNN autoencoder trained on flattened one-dimensional input. The system is also proven to be practically viable in addition to benchmark performance. The operational constraints of the real-time network monitoring setting, such as a mean inference latency of about 15 ms per event block, are quite within reach, and the unsupervised training mode implies that the system can be run in an environment where labelled attacks data are limited or even unavailable, which is characteristic of most security operations in the real world. There are three limitations of the current work to be identified. First, a lower recall of User-to-Root attacks at 93.1% can be explained by a structural problem of reconstruction-based unsupervised detection: U2R instances are few in both benchmarks datasets, so the model does not have as much contrast in the reconstruction boundary region between legitimate privileged behavior and malicious escalation. One way that could overcome this gap is by increasing the proportion of low-frequency attack patterns taken in the training environment, possibly by synthetic augmentation. Second, 2D matrix transformation introduces an extra processing step that creates an overhead of about 2 ms per event window. In the large majority of implementations of enterprise monitoring this overhead is insignificant, but it can limit applicability in sub-milliseconds streaming systems where each microsecond of pipeline latency counts. Third, the model was only trained and tested on two publicly available benchmark datasets whose traffic properties are that of particular network testbed environments. The application of the system to proprietary enterprise log formats, with varying sets of features, varying normal traffic distributions or varying mixes of attacks, would probably need a domain-specific retraining and potentially feature re-engineering before the

reported detection rates could be replicated. The most obvious continuation of this study in the future would be to incorporate an attention mechanism into the decoder, such that the model would be able to understand which spatial parts of the reconstructed matrix would have helped most towards a large score on the anomaly scale and hence in practice offer security analysts interpretable, feature-level explanations of individual alerts, as opposed to the scalar severity score. The second way to look is a combination of the 2D CNN encoder with a recurrent part that runs over consecutive matrix windows, which would enable the model to identify anomalies that emerge over longer time scales than can be identified by a single sliding window. The given 2D transformation is not limited to the field of network intrusion detection, and its testing in related fields like industrial control system monitoring, cloud infrastructure anomaly detection, and financial transaction analysis is an easy and potentially valuable future research opportunity.

References

1. M. M. Khan and M. Alkhatami, "Robust IoT security using isolation forest and one-class SVM algorithms," *Scientific Reports*, vol. 15, no. 1, Art. no. 14521, 2025. doi: 10.1038/s41598-025-20445-4
2. M. S. Mahmud, M. H. Ali, M. A. Islam, and M. S. Hossain, "Enhancing industrial control system security: An isolation forest-based anomaly detection model for mitigating cyber threats," *Journal of Engineering Research and Reports*, vol. 26, no. 3, pp. 161-173, 2024.
3. W. Chua, J. Lim, and K. Tan, "Web traffic anomaly detection using isolation forest," *Informatics*, vol. 11, no. 4, p. 83, Nov. 2024. doi: 10.3390/informatics11040083
4. A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab, "Machine learning for anomaly detection: A systematic review," *IEEE Access*, vol. 9, pp. 78658-78700, 2021. doi: 10.1109/ACCESS.2021.3083060
5. H. C. Altunay and Z. Sahingoz, "A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks," *Engineering Science and Technology, an International Journal*, vol. 38, p. 101322, Jan. 2023. doi: 10.1016/j.jestch.2022.101322
6. B. Al-Omar and Z. Trabelsi, "Intrusion detection using attention-based CNN-LSTM model," in *Proc. IFIP Int. Conf. Artif. Intell. Appl. Innovations (AIAI)*, IFIP Advances in Information and Communication Technology, vol. 675, Springer, 2023, pp. 447-458. doi: 10.1007/978-3-031-34111-3_43
7. S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "AutoLog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 184, p. 115481, Dec. 2021. doi: 10.1016/j.eswa.2021.115481
8. A. Seok, A. Kareem, and J. Hur, "A hybrid intrusion detection model using LSTM and LightGBM for robust anomaly detection in network systems," *International Journal of Basic and Applied Sciences*, vol. 14, no. 1, 2025. doi: 10.14419/ijbas.v14i1.35623
9. M. Ahsan, R. Gomes, and A. Denton, "Enhanced intrusion detection with LSTM-based model, feature selection, and SMOTE for imbalanced data," *Applied Sciences*, vol. 14, no. 2, p. 479, Jan. 2024. doi: 10.3390/app14020479
10. M. S. Hossain, M. Reza, and A. Al Mamun, "An optimized LSTM-based deep learning model for anomaly network intrusion detection," *Scientific Reports*, vol. 15, Art. no. 1541, Jan. 2025. doi: 10.1038/s41598-025-85248-z
11. Y. Li, Z. Wang, X. Chen, and H. Liu, "A multi-information fusion anomaly detection model based on convolutional neural networks and autoencoder," *Scientific Reports*, vol. 14, Art. no. 16060, Jul. 2024. doi: 10.1038/s41598-024-66760-0
12. I. Pivkina, A. Solomatin, and D. Nasonov, "Fuzzy CNN autoencoder for unsupervised anomaly detection in log data," *Mathematics*, vol. 11, no. 18, p. 3995, Sep. 2023. doi: 10.3390/math11183995
13. A. Kummerow, A. Esrom, S. Nicolai, and P. Bretschneider, "Unsupervised anomaly detection and explanation in network traffic with transformers," *Electronics*, vol. 13, no. 22, p. 4570, Nov. 2024. doi: 10.3390/electronics13224570
14. F. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi, and R. Ahmad, "CNN-LSTM: Hybrid deep neural network for network intrusion detection system," *IEEE Access*, vol. 10, pp. 99837-99849, 2022. doi: 10.1109/ACCESS.2022.3206425
15. K. A. Alaghbari, M. H. Saad, A. Hussain, and M. R. Alam, "Deep autoencoder-based integrated model for anomaly detection and efficient feature extraction in IoT networks," *IoT*, vol. 4, no. 3, pp. 345-365, Aug. 2023. doi: 10.3390/iot4030016
16. M. A. Alsoufi, M. M. Siraj, F. A. Ghaleb, and S. Saeed, "Anomaly-based intrusion detection model using deep learning for IoT networks," *Computer Modeling in Engineering and Sciences*, vol. 141, no. 1, pp. 823-845, 2024. doi: 10.32604/cmescs.2024.052112
17. Y. Song, S. Hyun, and Y. Cheong, "Analysis of autoencoders for network intrusion detection," *Sensors*, vol. 21, no. 13, p. 4294, Jun. 2021. doi: 10.3390/s21134294

18. Z. Ahmad, A. S. Khan, K. Nisar, I. Haider, R. Hassan, M. R. Haque, and J. J. Rodrigues, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, Jan. 2021. doi: 10.1002/ett.4150
19. Y. Chen, Y. Yang, and X. Zhao, "Network intrusion detection based on the temporal convolutional model," *Computers and Security*, vol. 136, p. 103587, Jan. 2024. doi: 10.1016/j.cose.2023.103587
20. H. Yang, J. Wang, M. Chen, and W. Zhang, "A hybrid BiLSTM-CNN approach for intrusion detection for IoT applications," *Scientific Reports*, vol. 15, Art. no. 3421, 2025. doi: 10.1038/s41598-025-29079-y
21. X. Li, R. Wang, K. Zhang, and H. Luo, "A high performance hybrid LSTM-CNN secure architecture for IoT environments using deep learning," *Scientific Reports*, vol. 15, Art. no. 9872, Mar. 2025. doi: 10.1038/s41598-025-94500-5
22. W. Wu, J. Hu, L. Zhao, and C. Feng, "Network intrusion detection model based on CNN and GRU," *Applied Sciences*, vol. 12, no. 9, p. 4184, Apr. 2022. doi: 10.3390/app12094184
23. H. Zhang, X. Liu, and J. Wang, "Semi-2DCAE: A semi-supervision 2D-CNN autoencoder model for feature representation and classification of encrypted traffic," *PeerJ Computer Science*, vol. 9, p. e1635, Nov. 2023. doi: 10.7717/peerj-cs.1635
24. T. Cook, A. Manzoor, and R. Buyya, "Deep learning for time series anomaly detection: A survey," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1-42, 2024. doi: 10.1145/3691338
25. N. Sadhwani, A. Mathur, R. Muthalagu, and P. M. Pawar, "Intelligent intrusion detection system against various attacks based on a hybrid deep learning algorithm," *Sensors*, vol. 25, no. 2, p. 580, Jan. 2025. doi: 10.3390/s25020580