# An Adaptive Hybrid Ensemble Intrusion Detection System (AHE-IDS) Using LSTM and Isolation Forest

IMTIAGE AHMED[1], RIPAN MIA[2], AND NUR ALAM FARHAD SHAKIL[3]

[1] Department of Computer Science and Engineering, World University of Bangladesh, Bangladesh
[2] School of Science & Technology, Bangladesh Open University, Bangladesh
[3] Department of Computer Science & Engineering, Hamdard University Bangladesh, Bangladesh

*Submitted: 11-June-2020.||Accepted: 30-OCT-2020 || Published: 11-NOV-2020*

*Abstract*

**Network intrusion detection is vital for the security of today's computer networks against malicious behavior. Existing detection systems often fail at achieving a good trade-off between the detection of known patterns of attacks and the detection of novel, unseen attacks. In this paper, we propose an Adaptive Hybrid Ensemble Intrusion Detection System (AHE-IDS) based on a supervised deep learning detector and an unsupervised anomaly detector to enhance detection rates. The suggested system integrates an Isolation Forest outlier detection technique and a Long Short-Term Memory (LSTM) neural network in a complementary way. The LSTM is trained with normal and malicious traffic patterns over the CSE-CIC-IDS2018 benchmark dataset that has a wide variety of attack scenarios. While the LSTM is perfroming this, the Isolation Forest is learning normal patterns to detect anomalies that could be new intrusions. A weighted voting method that adaptively weighs the two models' outputs dynamically combines the outputs of both models in such a way that the ensemble is able to prioritize the more dependable detector in accordance with prevailing conditions. Both known attacks and previously unseen anomalies outside standard traffic patterns can be detected using this hybrid method. AHE-IDS is tested on the CSE-CIC-IDS2018 dataset. Experiment results demonstrate that the ensemble achieves low false alarm rates and high detection rates, outperforming both LSTM and Isolation Forest individual models. According to the findings, AHE-IDS greatly enhances the accuracy and recall of an individual LSTM classifier at a low false positive rate. It successfully decreases missed attacks without influencing precision. The adaptive weighting scheme improves robustness as it adapts to concept drift and changing attack patterns over time. Consequently, AHE-IDS performs well in dynamic environments. The system is adaptive that can react to both familiar and unfamiliar kinds of cyber attacks.** 

**keywords**: *Adaptive ensemble learning, Anomaly detection, Deep learning, Intrusion detection system, Isolation Forest, LSTM, Network security*

## 1. INTRODUCTION

Intrusion Detection Systems (IDS) are critical in the cybersecurity domains today, constantly observing system activity and network traffic for unusual behavior that has evaded preventive security measures like firewalls, access control, or authentication systems [1]. Whereas preventive solutions represent the first line of defense in blocking or allowing traffic according to predefined policies, IDS products take on the complementary role in identifying malicious traffic after it has crossed a protected network. Through packet analysis, log analysis, system call, and application data analysis, an IDS can alert security personnel and offer intelligence to centralized management consoles, thus facilitating prompt incident response, forensics, and remediation. [2]

IDS solutions are also categorized both by the vantage point from which they examine data and by the approach they utilize to detect intrusions. From the viewpoint perspective, IDS is split into Network-based (NIDS) and Host-based (HIDS) systems [3]. Network-based IDS sensors are strategically installed in the network to intercept and analyze traffic in transit, whereas host-based IDS agents execute on target machines to monitor internal events, file-system activity, and process activity. Methodologically, IDS is most commonly characterized as signature-based—relying on established attack patterns—or anomaly-based—relying on behavioral baselines to reveal anomalies [4]. More recently, hybrid solutions blend both approaches to achieve a balance between detection efficacy for known threats and the capability to detect unknown or new exploits.

A Network-based IDS is installed at network choke-point areas such as border routers, switches with mirror/span ports, or network taps that are specifically designed for this purpose. The most critical task of the NIDS is to perform deep packet inspection, examining both header and payload sections of each packet [5]. Through rebuilding sessions from TCP streams or UDP datagrams, NIDS can run protocol decoders to recognize application-level activity. In passive deployments, NIDS examines a copy of the traffic without introducing latency, while inline deployments send the traffic through the IDS device so that it

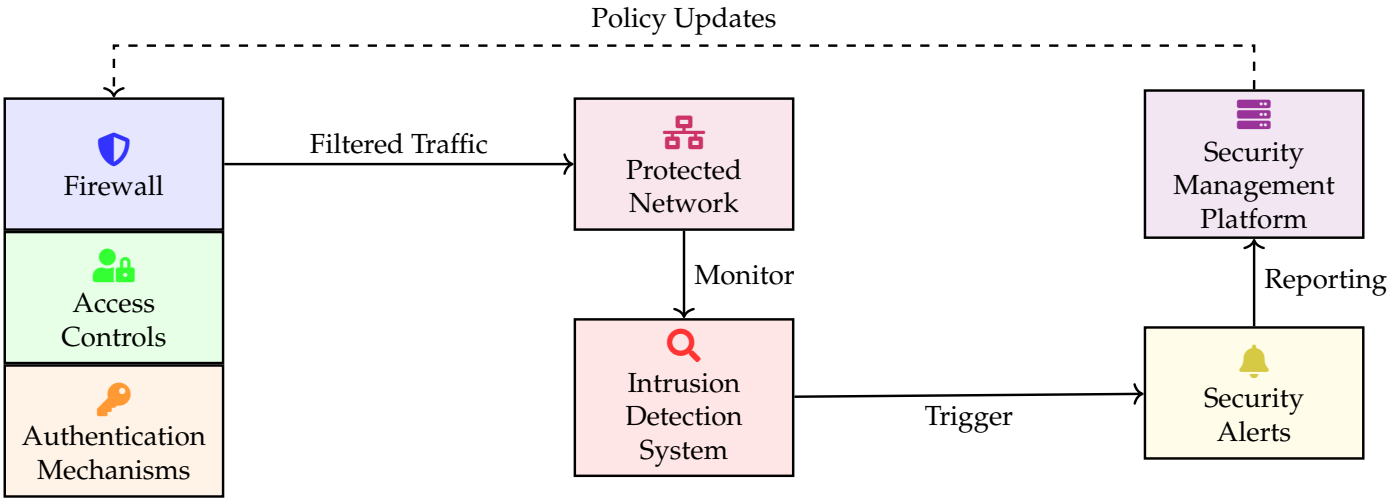



**Fig. 1.** Architecture of modern Intrusion Detection Systems (IDS) showing integration with preventive security measures (left), network monitoring capabilities (center), and security management ecosystem (right). The IDS operates as a secondary defense layer analyzing filtered network traffic and system behaviors, generating alerts for security teams, and integrating with centralized management platforms.

can actively drop or reset suspicious connections [6]. With signature matching and heuristic examination, NIDS is particularly good at identifying external scan attempts, denial-of-service, protocol misuse, and malware embedded in network traffic.

Host-based IDS is run on specific endpoints, e.g., servers, virtual machines, workstations, and even IoT devices [7]. HIDS agents hook into the kernel of the operating system or audit APIs to collect detailed logs of system calls, file integrity checksums, user privilege escalations, process creations, and registry modifications on Windows or configuration file modifications on Unix-like systems. In contrast to NIDS, which can miss encrypted or internal traffic, HIDS offers deep visibility into the internal state of a host. Through correlating application logs with system events, HIDS can detect insider abuses, privilege escalation attacks, evasive malware that circumvents network-based sensors, and unauthorized modifications to key binaries or configuration files. [8]

An IDS architecture is typically centered on three significant parts. The first part is the distributed sensors or agents, which gather raw data from network taps, log files, or kernel hooks [9]. These sensors perform pre-processing tasks such as normalization, de-duplication, and timestamp alignment with data. The second component is the detection engine, which performs signature matching, statistical models, machine learning algorithms, or a combination of these to identify suspicious events [10]. The third component is the management console and database, which are used to centralize rule creation, policy updates, alert aggregation, and reporting. This management level usually offers real-time monitoring dashboards, historical reporting for compliance audits, and interfaces to consolidate with Security Information and Event Management (SIEM) technologies or automated response systems. [11]

IDS solutions ingest a broad array of data sources to build a security image. Network-based sensors rely on full packet captures (PCAP) or summarized flow records (e.g., NetFlow, IPFIX) to extract metadata such as source and destination IPs, port numbers, protocol flags, and payload signatures. Host-based agents ingest system and application logs, audit logs, and file integrity monitoring feeds [12]. The majority of HIDS deployments supplement these logs with process-level telemetry, capturing command-line arguments, dynamic library loads, and inter-process communication. Deep sensors also leverage hypervisor APIs or container runtimes to monitor traffic and activity within virtualized or containerized environments [13]. All information is normalized into a common schema and enriched with contextual information—such as asset tags, user identity, and business criticality—to provide prioritized alerting.

Signature-based detection remains the workhorse of intrusion detection for identifying known threats with a high degree of accuracy [14]. A signature is a specific pattern—a byte string, a multi-layer protocol regular expression, or a behavioral rule—that uniquely characterizes an exploit or malicious payload. When incoming data is matched by a stored signature, the detection engine generates an alert or carries out a configured response action. Signature libraries span a spectrum of threats from buffer overflow shellcode, SQL injection payloads, hashes of malware files, and command and control beaconing [15]. Signature databases must be updated on a constant basis in order to cover the most recently discovered vulnerabilities as well as newly developed types of malware.

The effectiveness of signature-based IDS depends on the efficiency of its pattern-matching engine [16]. They feature popular methods like the Aho–Corasick algorithm, which constructs a joint finite-state automaton to find multiple patterns in a single pass over the data stream, performing linear time complexity relative to data size. Heuristic skip tables are exploited by the Boyer–Moore algorithm family to skip non-matching data portions, lowering average-case matching time [17]. For advanced protocol-aware rules, IDS products leverage optimized regular expression engines that translate expressions to nondeterministic or deterministic finite automata, with a few employing Just-In-Time (JIT) compilation for performance at runtime. Advanced IDS products divide the work of matching between CPU cores and leverage SIMD instructions to parallelize byte-level comparisons [18]. Anomaly-based IDS methods are set up to detect departures from learned patterns of normal behavior and therefore are well placed to detect zero-day exploits and new attack techniques. Training of an anomaly detection

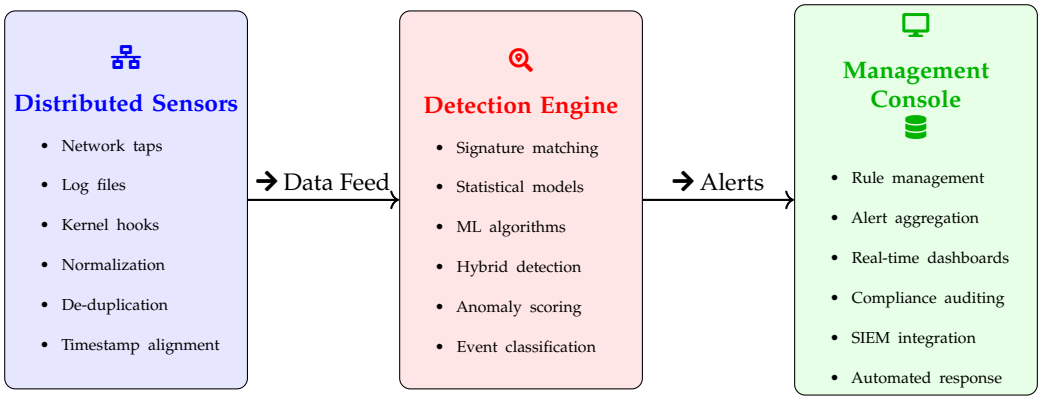



**Fig. 2.** Intrusion Detection System with three primary components: Distributed sensors for data collection/preprocessing, Detection engine for analysis, and Management console for coordination and reporting.

model starts with a training phase in which the system learns baseline behavior—i.e., normal packet size distributions, login times, sequences of commands issued by a user, or application call graphs. After the baseline is learned, live data is scored against the model in real time [19]. Those events that are outside acceptable statistical boundaries or contradict learned behavior patterns are reported as anomalies. Anomaly IDS, when compared to signature-based systems, needs to be tuned in an attempt to create an effective tradeoff between sensitivity and false positives. [20]

At the center of the majority of anomaly-based systems are conventional statistical techniques. Simple univariate models monitor metrics such as the average number of failed login attempts per hour and alert when counts deviate beyond a threshold set based on standard deviation ranges [21]. Multivariate methods consider the correlation between measurements—such as packet size and inter-arrival time—using techniques like Principal Component Analysis (PCA) to project data onto lower dimensional spaces and identify outliers. Time-series methods such as Exponential Weighted Moving Averages (EWMA) or Cumulative Sum (CUSUM) charts identify gradual drift in metrics over time. Entropy-based metrics measure the randomness of features—for example, the variability of destination IP addresses in DNS requests—to alert on sudden bursts of variation that can signal scanning or exfiltration. [22]

Machine learning advances have prompted the use of both supervised and unsupervised techniques in IDS. Scholars have proposed several machine learning strategies for intrusion detection [23]. Supervised classifiers such as Support Vector Machines (SVM), Random Forests, or gradient-boosted trees are trained on labeled datasets of benign and malicious samples and learn decision boundaries between the two classes. Unsupervised clustering algorithms such as k-means or DBSCAN can group similar events together and consider small or isolated clusters to be threats [24]. One-class SVM and Isolation Forests explicitly learn a model of "normal" data and label deviations as anomalies. More recently, deep learning architectures—e.g., autoencoders and recurrent neural networks (e.g., LSTM models)—have demonstrated strong abilities to model intricate temporal dynamics in network flows or system call sequences, reconstruct expected inputs, and compute anomaly scores based on reconstruction error. [25]

Hybrid IDS architectures combine the strengths of signature-based and anomaly-based detection to provide layered protection. In a common deployment, incoming traffic is first passed through a signature engine that removes known threats with high certainty. The remaining traffic is fed into an anomaly detection module, which analyzes behavioral deviations more deeply [26]. Hybrid models may also include a reputation-based component that enriches alerts with threat intelligence feeds, IP reputation scores, or domain blacklists. Through correlating multiple detection engines in parallel, hybrid IDS reduce the volume of information that the anomaly engines must process and lower the overall false positive rate without sacrificing the potential for detecting novel attacks. [27]

## 2. PROBLEM STATEMENT AND RESEARCH OBJECTIVES

Anomaly detection systems are likely to have greater numbers of false alarms because all deviations are not necessarily reflective of malicious activity. Researchers have designed hybrid intrusion detection models that combine various approaches to improve coverage and accuracy due to the limitations of using each approach separately [28]. Indeed, the combination of supervised learning with unsupervised anomaly detection is able to synergistically leverage the advantages of both paradigms. This aligns with the general ensemble learning model that a combination of models may yield better performance, especially when the individual models supplement each other's strength and weakness. Supervised machine learning methods (e.g., neural networks) and deep learning can be taught to recognize complicated patterns of familiar intrusions by examples of labeled training data, yielding high true positive rates against comparable attacks encountered during training [29]. Amongst these, recurrent neural networks like Long Short-Term Memory (LSTM) have proven effective at modeling temporal relationships in network traffic data in order to identify attack patterns that unfold over time. Unsupervised methods do not rely on prior knowledge about attack signatures, however [30]. Techniques like Isolation Forest, an ensemble-based tree-based anomaly detector, build a model of regular network behavior and then identify outliers potentially indicative of intrusions. Isolation Forests randomly partition data and separate anomalous points more rapidly (in fewer splits) than regular points, making them appropriate for the detection of atypical network behavior [31]. These two techniques – sequence learning to detect patterns and anomaly detection to detect outliers – can be combined by a hybrid system and, in theory, detect a larger variety of intrusions than either one alone.

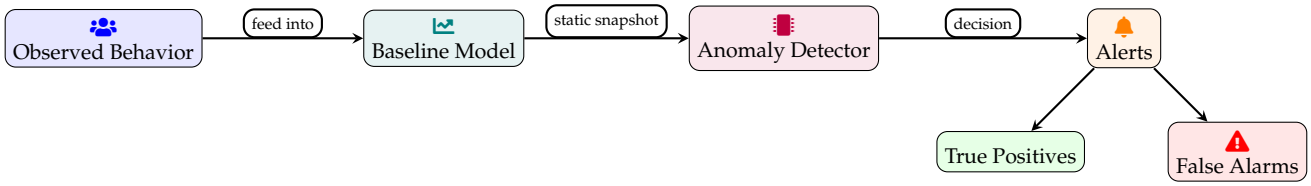



**Fig. 3.** Problem: High rate of false alarms in anomaly detection due to benign deviations being flagged as anomalies.

In this paper, we propose an Adaptive Hybrid Ensemble Intrusion Detection System (AHE-IDS) that combines the LSTM classifier with the Isolation Forest anomaly detector and combines their predictions through a dynamic weighted voting scheme [32]. The use of "adaptive" highlights that the weighting of the ensemble is not fixed; instead, the system can dynamically change the contribution of each component based on their previous performance or confidence. For example, if the LSTM model is observed to be very precise in detecting certain high-volume attacks, the system can raise its weight temporarily for such instances. Conversely, if there is a new traffic pattern that lies outside the experience of the LSTM, the Isolation Forest output can be weighted more to indicate the anomaly [33]. This merging dynamic aims to reduce the total false negative rate (missed attacks) by causing each detector to miss an intrusion only when the other misses. As it does so, it can also control false positives by utilizing the more confident detector when the other generates an uncertain alarm. [34]

We compare proposed AHE-IDS on the CSE-CIC-IDS2018 dataset, which presents a realistic and diverse benchmarking opportunity for intrusion detection research. The dataset represents extensive network traffic data with both benign activity as well as attacks of different forms, such as brute-force password attacks, exploitation of vulnerabilities (e.g., the Heartbleed attack), denial-of-service (DoS and DDoS) flooding, botnets, web attacks (e.g., SQL injection and XSS), and insider intrusions [35]. By training and testing on this data set, we are assured that our analysis identifies definite attack patterns which are easily distinguishable from normal traffic as well as malicious attack patterns that subtly mimic normal traffic. The scale and variety of CSE-CIC-IDS2018 (with millions of network flow records across multiple simulated days) allow us to thoroughly test the detection precision, false alarm management, and adaptability of the AHE-IDS over a range of attack scenarios.

Over the past decade, a variety of machine learning techniques have been applied to intrusion detection problems [36]. Classic methods such as decision trees, support vector machines, and clustering algorithms were the basis for anomaly detection in earlier IDS studies, whereas more recent studies have depended increasingly on deep learning models (such as multilayer perceptrons, convolutional neural networks, and recurrent neural networks) to detect advanced patterns in network traffic. Every technique has its pros and cons: deep neural networks are highly accurate given sufficient training data, but difficult to interpret and require fine-tuning, while less complex models are easier to interpret but won't necessarily identify subtle indicators of attacks [37]. Ensemble techniques have evolved as a method of exploiting the advantages of multiple models. Techniques like bagging and boosting combine classifiers of the same class to reduce variance or bias, whereas hybrid ensembles combine different classes of detectors (e.g., misuse and anomaly-based) to expand coverage [38]. Our AHE-IDS falls in the latter category, a stacking ensemble that combines heterogeneous learning strategies. This type of supervised-unsupervised hybrid is comparatively less studied than pure supervised ensembles, but the potential gains are large in cybersecurity applications, as our results demonstrate. With evaluating our system on a modern, large-scale dataset (CSE-CIC-IDS2018) that reflects current attack patterns, we aim to show that such a hybrid approach can attain the high performance rates reported in recent intrusion detection literature, while addressing the detection of new threats in a principled manner. [39]

## 3. METHODOLOGY

Adaptive Hybrid Ensemble IDS consists of two core detection modules executed in parallel and an ensemble decision logic which combines their outcomes. The initial module is an LSTM neural network classifier that monitors sequences of network traffic features with the aim to identify patterns which indicate attacks. The second module is an Isolation Forest (IF) algorithm that identifies anomalies by locating outlier observations in the feature space. Each module independently makes a judgment or score as to whether the traffic observed is malicious. The ensemble part dynamically combines these with weights to create a final decision. By nature, the LSTM component excels at recognizing temporal patterns of intrusions learned from training data, while the Isolation Forest excels at recognizing abnormal events that do not conform to normal patterns, like potential zero-day attacks that were not part of the training set. The weighted voting combination tries to eliminate each's drawback: if the LSTM doesn't catch a new attack, the anomaly detector might catch it, and if the Isolation Forest overreacts toward benign anomalies, the LSTM can veto the false positives.

The LSTM sequence classification model is AHE-IDS's first building block. In our approach, the LSTM is used to model sequences of network traffic data and classify them as malicious or benign. Network traffic is treated as an ordered sequence of feature vectors (such as consecutive network flow records or packets) capturing the behavior within a short time window. The LSTM handles one element of the sequence at a time, maintaining an internal state that can hold information about past elements of the sequence. Formally, at each time step $t$, the LSTM calculates its current hidden state $h_t$ and cell state $c_t$ based on the current input vector $x_t$, the last hidden state $h_{t-1}$, and the last cell state $c_{t-1}$. The update equations for the LSTM's gating mechanism are:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (2)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (3)$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (4)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (5)$$
$$h_t = o_t \odot \tanh(c_t) \quad (6)$$

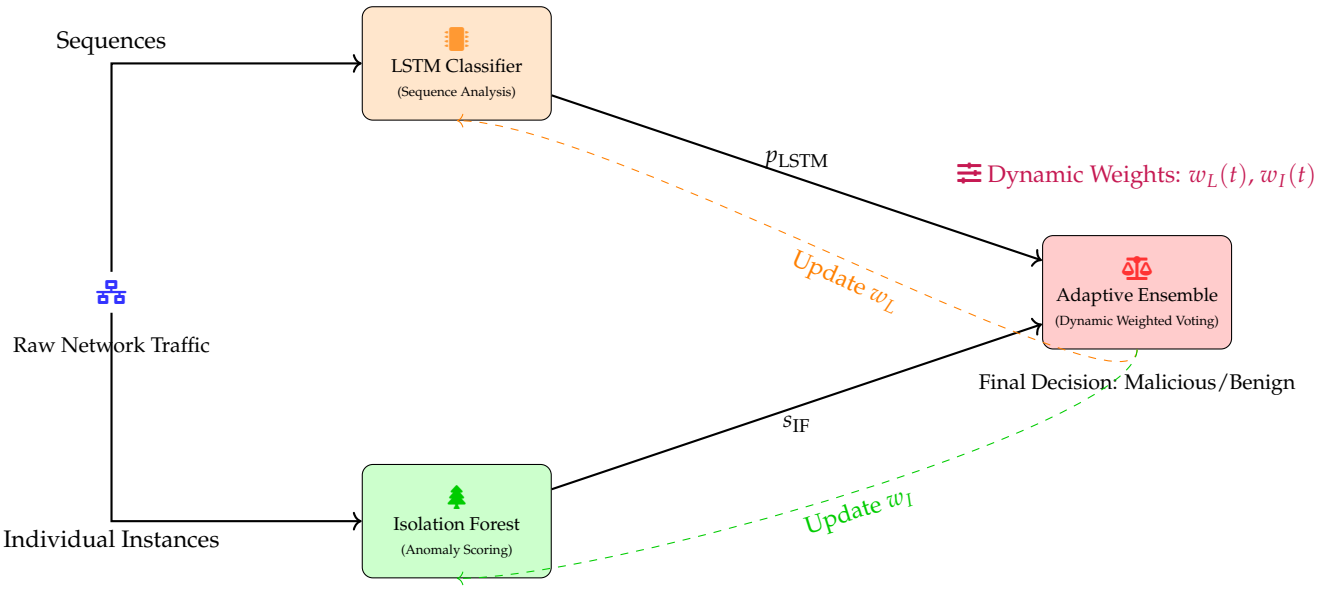
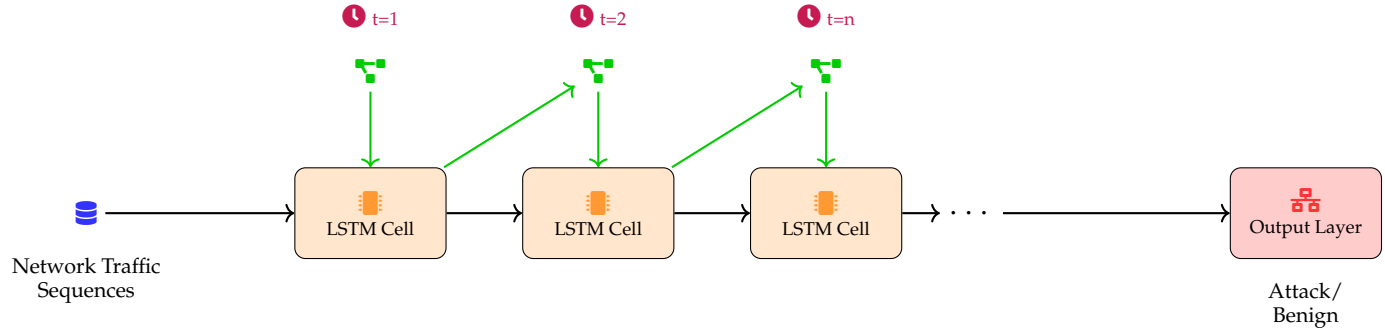



**Fig. 4.** Adaptive Hybrid Ensemble Architecture: Processes network traffic through parallel LSTM (sequence analysis) and Isolation Forest (anomaly detection) modules. The ensemble logic dynamically adjusts weights ($w_L$, $w_I$) based on performance feedback to combine predictions.




$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c)
\end{aligned}
$$

**Fig. 5.** LSTM Module Architecture: Processes sequential network traffic data using memory cells and gating mechanisms. Colored icons represent different components (input, LSTM cells, hidden states, output). The equations below show the core LSTM operations.

Here, $i_t$, $f_t$, and $o_t$ are input, forget, and output gates (which are sigmoid $\sigma$-activated), $\tilde{c}_t$ is the candidate cell state (with tanh activation), and $\odot$ denotes element-wise multiplication. These gates control how information is propagated through the LSTM: $i_t$ controls what new information from the current input to include, $f_t$ controls what information to retain from the past, and $o_t$ controls how much of the cell state to reveal to the output. When adjusting these gates incrementally, the LSTM can remember important patterns from long sequences and forget irrelevant information, avoiding the limitations of smaller recurrent networks (e.g., the vanishing gradient issue in standard RNNs). This architecture enables the LSTM to retain useful information across long sequences of events and forget useless noise, which is essential in network traffic analysis where useful clues can be distant in time. After going through a sequence of $T$ time steps (for example, $T$ network events), the LSTM produces a final hidden state $h_T$ which represents the information from the whole sequence. A fully connected output layer (binary classification with sigmoid activation) takes in $h_T$ and produces a prediction $\hat{y} = \sigma(W_y h_T + b_y)$, which can be interpreted as the probability of the input sequence being an intrusion. As training, the weights of the LSTM ($W_*$, $U_*$, $b_*$, etc.) are learned on labeled data (attack vs normal sequences) minimizing a binary cross-entropy loss. The model is hence trained to output $\hat{y}$ as close to 1 for attack sequences and 0 for normal sequences. At deployment, the LSTM module outputs a score (the predicted probability or a binary classification after thresholding at 0.5) for each sequence of network traffic under observation.

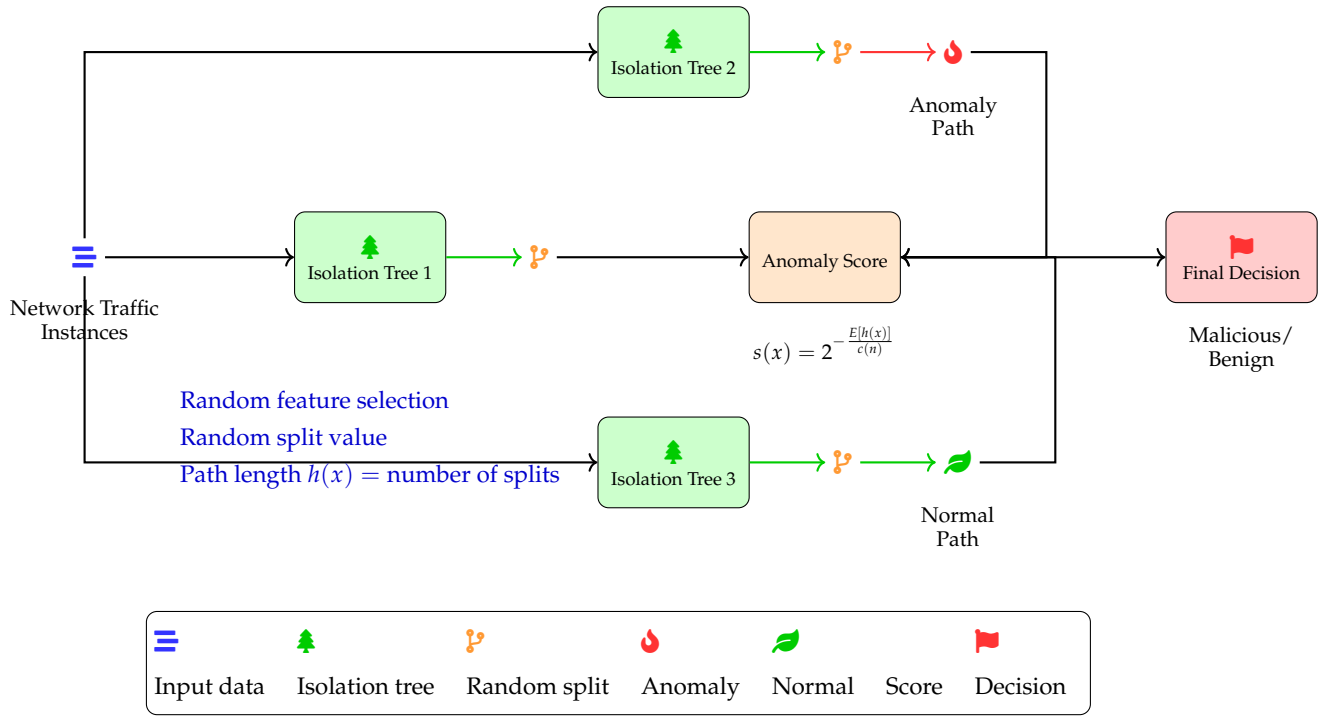



**Fig. 6.** Isolation Forest Module Architecture: Detects anomalies through ensemble random partitioning. Input instances are processed by multiple isolation trees with random splits (). Anomalies () are isolated with shorter paths than normal instances (). The average path length is converted to an anomaly score and thresholded for final decision.

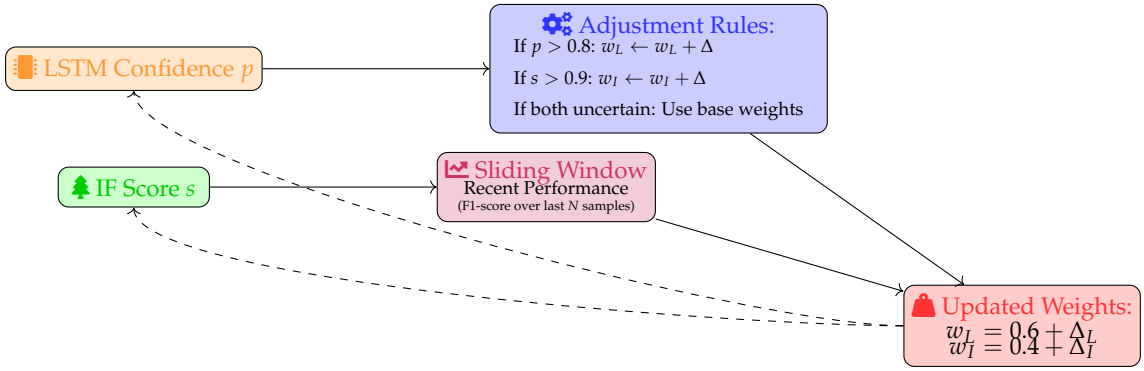



**Fig. 7.** Dynamic Weight Adjustment Logic: Updates ensemble weights based on model confidence scores and recent performance metrics. Includes both instant rule-based adjustments and longer-term sliding window evaluation.

The second module is the Isolation Forest anomaly detector. The Isolation Forest component of AHE-IDS offers an unsupervised attack detection ability that does not require the prelabeling of attacks. An Isolation Forest is a collection of random decision trees (isolation trees) that grow by selecting features and split values randomly. The key point is that anomalies (intrusive cases of traffic) are statistically easier to identify as outliers from the remainder of the data because they have a different difference from "normal" cases. To train the Isolation Forest, we primarily utilize a set of benign cases of traffic to determine what is "normal.". Every tree in the isolation trees is built by recursive partitioning of the data: at every node, a random feature is chosen and a random split value chosen between the minimum and maximum of the feature. The process continues until the instance is parted to a leaf or a maximum tree depth is spent. Anomalous instances fall into leaves at more shallow depths (smaller number of splits) than normal instances. For a novel data point $x$, an anomaly score is computed by Isolation Forest based on the average path length $h(x)$ of $x$ across all trees in the forest. Let the average number of splits to isolate $x$ be $E[h(x)]$. This is scaled by a term $c(n)$, i.e., the average path length of an average normal point in a tree for training data size $n$. The anomaly score $s(x)$ is then defined as:

$$s(x) = 2^{-\frac{E[h(x)]}{c(n)}},$$

which provides scores within the range (0, 1]. Under this definition, $s(x)$ will approach 1 when $x$ is highly likely to be an anomaly (isolated very early, with $E[h(x)]$ much smaller than $c(n)$). Conversely, $s(x)$ will be lower (closer to 0) for data that penetrate further into the trees (i.e., they are more similar to normal data patterns). In practice, we put a threshold on $s(x)$ to decide if $x$ is malicious: when $s(x)$ is higher than some threshold

(e.g., close to 0.5), the example is an attack; otherwise, it is normal. We can decide the threshold by looking at the distribution of scores on a validation set. With training on purely normal traffic, the Isolation Forest is particularly good at picking up novel attacks that cause outlier behavior in network features, which is complementary to the LSTM which targets known patterns.

Once both the LSTM and Isolation Forest modules have each provided a verdict or score for an input network (or traffic window), AHE-IDS ultimately decides by aggregating these outputs. We employ a weighted voting mechanism where the input of each module to the final decision is weighted by a weight that can change over time. Let $p_{LSTM}$ denote the anomaly (binary) probability "malicious" given by the LSTM for a sample, and let $s_{IF}$ denote the anomaly score of the Isolation Forest for the sample. We have weights $w_L$ and $w_I$ allocated for the LSTM and Isolation Forest respectively, with the constraint $w_L + w_I = 1$. The ensemble is able to compute an aggregate anomaly score $S_{ensemble} = w_L \cdot p_{LSTM} + w_I \cdot s_{IF}$. For binary classification, we then make a comparison against a threshold (e.g., 0.5) to decide whether the sample is attack or normal. It might be possible to begin by fixing $w_L$ and $w_I$ depending on the overall accuracy or trustworthiness of each model (for instance, if LSTM on its own is slightly more accurate, $w_L$ could be fixed higher than $w_I$). However, a fixed weight may not always be optimal under all circumstances, especially when traffic type changes or if one model is much better than the other for certain types of attacks. Thus, our system is learned: the model weights $w_L$ and $w_I$ can be adjusted based on feedback regarding performance. Offline, this can be done by comparing recent detection outcomes on a sliding window and giving greater weight to the better performing model within the window. In an adaptive online scenario, one could also have confidence measures – e.g., having $w_L$ depend on $p_{LSTM}$ itself, so that when the LSTM is very confident (output close to 1 or 0), it gets to have a greater say, and on the other hand, if the Isolation Forest gives an outlier score of nearly 1 when the LSTM is not confident, $w_I$ could be boosted for that sample. We apply our implementation to calibration on a validation subset of data: we found an optimal base weighting, which best maximizes detection rate and reduces false alarms (e.g., $w_L = 0.6, w_I = 0.4$ can be used if LSTM performs slightly better on validation than IF). We then allow the real-time modulation by a smaller degree if confidence from one model greatly surpasses the other for a particular detection. The adaptive voting algorithm guarantees the final ensemble prediction is more robust than that of any individual model because it can handle various types of attacks and prevalence. If both models agree (both vote attack or both vote benign), then the ensemble would logically give that prediction. If they disagree, the weighting will favor the more trusted or confident prediction. This solution perfectly balances the low false positive bias of the LSTM with the ability of the Isolation Forest to detect anomalies and arrives at a more balanced detector.

Our inclusion of an LSTM and an Isolation Forest in our ensemble was motivated by their complementarity. Recurrent neural networks like LSTM are natively adapted to model sequence data and can recognize temporal patterns in network traffic that would not be visible to static classifiers (e.g., a slow port scan over time or a series of related events that make up an attack sequence). We used an LSTM instead of a simpler feed-forward neural network to take advantage of these temporal dependencies in attack behaviors. At the same time, the Isolation Forest was used as the anomaly detection module because it is efficient and makes few assumptions regarding data distribution. Other unsupervised techniques such as one-class SVMs or deep autoencoders were also contemplated, but Isolation Forest is extendable to large datasets and requires minimal parameter tuning, making it well-suited for high-dimensional network flow features. Furthermore, the tree-based isolation process addresses various types and ranges of features after normalization naturally. Incorporating a deep learning model into a tree-based outlier detector thus introduces diversity in detection mechanisms within the ensemble. This diversity is important for a healthy system: if one approach has a blind spot (i.e., the LSTM won't generalize to a new attack type, or the Isolation Forest will trigger on an odd but benign pattern), the other approach can provide backup. Through integrating these aspects, our strategy aims for optimal detection coverage with minimal false alarms, striking a practical compromise between more heavy-weight (yet effective) and light-weight (yet fast and generalizing) detection techniques.

Practically, AHE-IDS is deployed as follows: (1) train the LSTM on a collection of labeled network flow sequences (with normal vs attack labels) to learn a sequence-based classifier; (2) train the Isolation Forest on a dataset of mostly benign flows to learn a baseline model of normal traffic and to calibrate its anomaly threshold; and (3) set the right voting weights (and threshold for the ensemble score) based on a validation dataset consisting of both normal and attack examples. With these steps complete, the system is ready to deploy on new network traffic, with each flow (or sequence of flows) being passed through the LSTM and Isolation Forest, and the combined weighted output used to classify whether or not an intrusion is present. In the next section, we detail the experimental setup used to implement these steps and evaluate the performance of the system.

## 4. EXPERIMENTAL SETUP

**Table 1.** CSE-CIC-IDS2018 Dataset Composition

| Component | Days | Benign | Malicious |
|---|---|---|---|
| Training | 1-7 | 8.2M (82%) | 1.8M (18%) |
| Validation | 8 | 1.8M (85%) | 0.3M (15%) |
| Test | 9-10 | 3.3M (83%) | 0.7M (17%) |

**Table 2.** Data Preprocessing Pipeline

| Step | Description |
|---|---|
| Feature Removal | Discarded 12 non-informative features (timestamps, flow IDs) |
| Normalization | Min-max scaling to [0,1] range for all 80 features |
| Missing Values | Replaced infinities with $10^6$, nulls with 0 |
| Sequence Creation | Grouped flows into non-overlapping windows of 20 |
| Class Balancing | Weighted loss (malicious:benign = 3:1) |

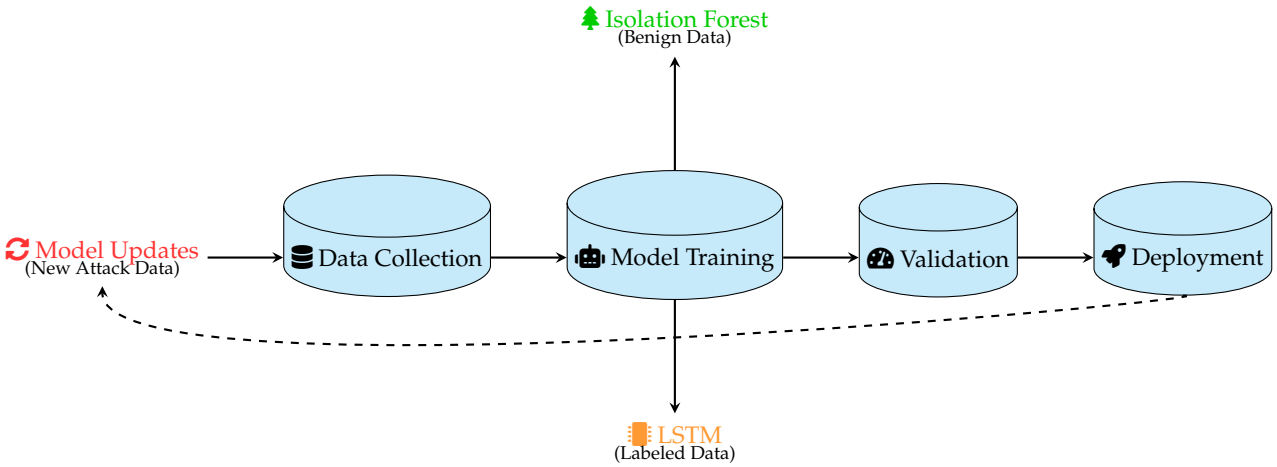



**Fig. 8.** End-to-End Training and Deployment Pipeline: Shows the workflow from data collection through parallel model training to real-world deployment with continuous feedback for model updates.

**Table 3.** Model Architectures and Parameters

| Parameter | LSTM | Isolation Forest |
|---|---|---|
| Input Shape | (20, 80) | 80 features |
| Core Components | 64 LSTM units + Dropout(0.2) | 100 trees |
| Training Data | 1.5M sequences | 100K benign flows |
| Optimizer | Adam (lr=0.001) | N/A |
| Stopping Criteria | Early stop (3 epochs) | N/A |
| Inference Speed | 25k seq/sec | 50k flows/sec |

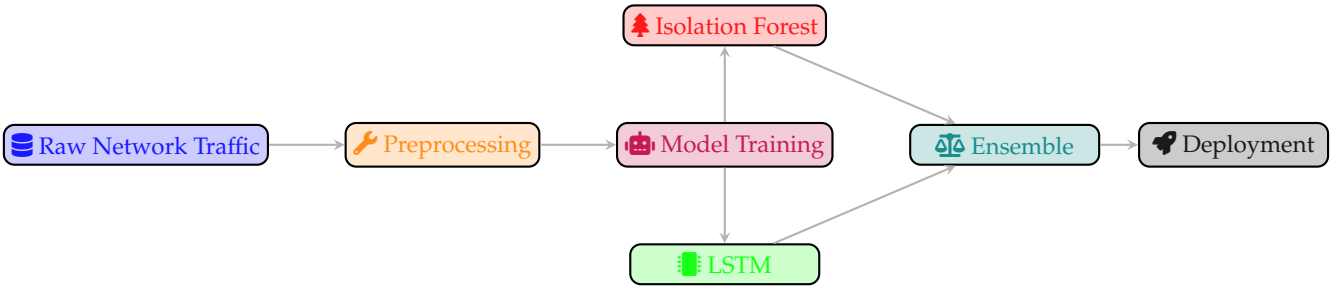



**Fig. 9.** Training Workflow: pipeline showing data flow from raw network traffic through preprocessing, parallel model training, ensemble combination, to final deployment.

We carry out our experiments on the CSE-CIC-IDS2018 dataset, a modern benchmark for evaluating intrusion detection systems [40, 41]. The dataset consists of raw network traffic captures and calculated flow records for ten days' worth of network traffic with normal behavior and seven distinct attack scenarios. We work on the provided flow-level feature set of 80 statistical features per flow (for example, packet count, byte volume, duration, header features, and cumulative statistics like average packet size and flow byte rate) extracted using CICFlowMeter. All of the flow records in the dataset are labeled as benign or with a specific type of attack. The attacks span across a range of categories [42, 43]: (1) brute-force login tries on services like FTP and SSH, where an attacker repeatedly attempts passwords in the hopes of gaining unauthorized access; (2) the Heartbleed exploit, a bug within OpenSSL's heartbeat functionality that an attacker can exploit to steal sensitive data from the memory of a server, posing as strange TLS traffic; (3) communications from botnets, where a compromised host (bot) is communicating with a command-and-control server or engaged in automated malicious activity; (4) denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks, where a target is flooded with too much traffic to overwhelm its services; (5) web attacks, like SQL injection and cross-site scripting (XSS), where malicious HTTP requests are initiated in the hopes of taking advantage of web server or application weaknesses; and (6) an infiltration attack originating from inside the network, acting like an insider threat or compromised internal host doing unauthorized reconnaissance or data exfiltration. These diverse forms of attacks

provide a broad test bed for intrusion detection capability. In total, the data set comprises on the order of tens of millions of flows (about 16 million records). In our training partition (first 8 days), there were approximately 10 million benign flow records and 2 million malicious flow records, while the test partition (last 2 days) contained approximately 3.3 million benign and 0.7 million malicious flows. The class distribution is not even: the majority of data consists of benign traffic 83% of all cases, and the remaining 17% is malcieous traffic. Even within the attack class, there is an imbalance – volumetric attacks like DoS and DDoS comprise the lion's share of the attack instances, whereas stealthy attacks such as Heartbleed or infiltration appear in very low numbers (each with less than 1% of the instances). This is a hindrance in training detection models since they could be biased towards the majority classes unless handled with care [44].

Before feeding the data into our models, we perform a few preprocessing steps. We remove any non-informative or redundant features (e.g., flow identifiers, timestamp fields, or constant columns) that will not generalize or help in detection. All the categorical columns (such as protocol type, if they are present as a string) are transformed into numeric representation – in reality, the protocol attribute already looks numeric here, employing integer codes for TCP, UDP, etc. We then normalize the feature values to a common scale since the raw features are of varying ranges (e.g., packet numbers may be in hundreds, whereas some ratio features range from 0 to 1). Normalization is done by performing min-max scaling on every feature to normalize it to [0,1]. The scaling avoids any feature dominating others due to magnitude and also is appropriate for the training dynamics of the neural network. Besides, we address any missing or infinite values within the feature set: e.g., if there is a zero duration in a flow that will cause a division-by-zero while computing a rate, we impute or limit such values (we replaced infinities with a gigantic finite value and missing values with 0). After cleaning, the dataset is split into training, validation, and test subsets. To keep the ordering of the data in time and to evaluate generalization to novel data, we divide by days: train/test on the first eight days of traffic and test on the last two days. The train set is once again divided, holding roughly 10–15% in a validation set that is used for hyperparameter search and adjustment of ensemble weights. This temporal split ensures that the test set contains patterns (and some attacks) unseen during training days, providing an honest estimate of the ability of the system to detect new or evolved attacks.

We construct a training set of sequence examples from labeled data for the LSTM model. A sequence can be represented in numerous ways. In our setup, we create sequences by putting consecutive network flows (by start time) into fixed-size windows of 20 flows (not overlapping), and labeling each sequence as containing some malicious activity. If the window contains any attack flow, the whole sequence is classified as malicious – this makes the task of the LSTM one of deciding whether an attack exists in a traffic window. (While the dataset distinguishes among various kinds of attacks, we reduce the classification task to binary detection between "intrusion" and "benign"; identifying particular classes of attacks is left as future work.) We use a window size large enough to include enough context without mixing too many events (20 flows is a short time interval in our dataset). We found that introducing a mixture of normal and potential attack flows to a stream helps the LSTM learn nuanced temporal patterns (e.g., sequential failed logins in brute force attacks). Due to class imbalance, we employ techniques to avoid biased learning: we down-sample the overwhelming amount of benign sequences and up-sample or weight the sparse set of attack sequences so the model sees a more balanced set during training. Specifically, we use a weighted binary cross-entropy loss that gives higher weight to errors on the malicious class to force the model to learn to label intrusions. The LSTM architecture used in our experiments consists of an input layer accepting the sequence of feature vectors (one flow per normalized 80 features), and one LSTM layer with 64 hidden units. We follow the LSTM with a dropout layer with dropout 0.2 to avoid overfitting by randomly dropping units during training. The output is a dense sigmoid layer that provides the intrusion probability

$haty$. We train the LSTM using the Adam optimizer and initial learning rate of 0.001. Training is done for a maximum of 20 epochs, and early stopping is implemented on the validation loss in order to prevent overfitting (training stops if the validation loss fails to improve for 3 successive epochs). On our system (a machine with an NVIDIA GPU for acceleration), training one epoch over the data (after preprocessing and sampling) takes on the order of a few minutes, and the model converges normally within less than 10 epochs with early stopping. The trained LSTM model of the final epoch is then frozen for testing on the test set.

For Isolation Forest, we build a training set consisting, ideally, solely of normal traffic. We randomly draw many examples of benign flow records from the training subset of the data (in order to maintain computational feasibility, we might take on the order of 100,000 normal samples if the overall training set is incredibly large). Training the Isolation Forest on plain data ensures that the trees will learn the "normality" and thus will assign low anomaly scores to normal benign patterns, but higher ones to outliers (hopefully attacks). We use scikit-learn's implementation of Isolation Forest with 100 estimators and sub-sampling size of 256 per estimator (each estimator is trained on a random subset of 256 instances from the training data, which is a standard default to preserve efficiency and diversity across trees). We set the contamination parameter (proportion of training data anomalies) to 0, since we are providing primarily clean normal data; this isn't that there is some inbuilt assumption within the model regarding a certain ratio of anomalies, but we shall have thresholds elsewhere. After the training of Isolation Forest, we choose an anomaly score threshold to label intrusions. In this case, to perform this task, we make use of the validation set with normal and attack instances. We compute the anomaly score $s(x)$ for every instance of the validation data. We pick a threshold which is a good compromise between detection and false alarms on the validation set. For instance, we can set the threshold such that 95% of validation normal examples have lower scores than it (with a low false positive rate) and a high fraction of validation attacks have higher scores than it. In our experiment, a threshold of about 0.5 for $s(x)$ (which is reasonable, as 0.5 is a standard value in Isolation Forest literature employed as a default normal vs. anomaly separator) worked well, but we tuned it for optimal validation F1-score. This threshold is then used at test time: any new flow with $s(x)$ above the threshold is flagged as malicious by the Isolation Forest module.

With the LSTM and Isolation Forest models pre-trained, we are now ready to combine their predictions. On validation, we also adjust the weights $w_L$ and $w_I$ used in the ensemble. On the validation set, we approximate the ensemble by sending each sample through the LSTM and Isolation Forest to get $p_{LSTM}$ and $s_{IF}$, and then trying different weight values to compute an en-

semble score $S_{ensemble}$. We select the weight which achieves the highest detection precision or F1-score on the validation set. For example, we can observe that giving a slightly more weight to LSTM (since it was slightly more precise in validation) is ideal, resulting in a starting point like $w_L = 0.6, w_I = 0.4$. But we also apply the dynamic adjustment scheme as described above: in practice, we simply have a crude heuristic where if the two models disagree on a classification, we examine their confidence (how far $p_{LSTM}$ or $s_{IF}$ are from the 0.5 middle point or threshold) and push the weights a bit towards the more confident model for that decision. This does not include retraining; it is an on-the-fly modification. The overall evaluation procedure is as follows: for each flow (or sequence) in the test set, retrieve the LSTM prediction and IF anomaly score, compute the weighted ensemble score from the calibrated weights (updated dynamically if needed for that example), and label as normal or attack. All experiments were conducted on a Linux server with 32 CPU cores and an NVIDIA Tesla GPU. Our implementation utilized Python with TensorFlow/Keras for the LSTM and scikit-learn for the Isolation Forest. The training of the models took hours collectively, but the inference takes a short time, only seconds to process tens of thousands of samples. This suggests that AHE-IDS can be implemented for real-time or near-real-time applications.

## 5. EVALUATION METRICS

**Table 4.** Evaluation Metrics and Formulas

| Metric | Formula | Purpose |
|---|---|---|
| Accuracy | $\frac{TP+TN}{TP+TN+FP+FN}$ | Overall correctness |
| Precision | $\frac{TP}{TP+FP}$ | Alert quality |
| Recall | $\frac{TP}{TP+FN}$ | Attack coverage |
| FPR | $\frac{FP}{FP+TN}$ | False alarm rate |
| F1-Score | $2 \cdot \frac{P \cdot R}{P+R}$ | Balanced measure |

To measure AHE-IDS performance, we utilize standard classification measurements applied in the testing of intrusions. We define these measures in terms of the observed outcomes over the test set: True Positives (TP) are properly labeled malicious events (attacks); True Negatives (TN) are properly labeled benign events as benign; False Positives (FP) (also known as false alarms) are when benign traffic is misclassified as malicious; False Negatives (FN) are attacks not detected by the system (classified as normal). Using these definitions, we compute the following measurements:

Accuracy: Accuracy is the overall accuracy of the system since the proportion of all cases of traffic accurately labeled (as attack or normal). In terms of an equation, Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$. While accuracy provides one performance measure, it may be misleading in imbalanced datasets. Because the mean traffic far outweighs attacks in our collection, a stupid classifier that predicts all as normal would be highly accurate but utterly useless for security. Therefore, we are more interested in the following measures which focus on the positive (attack) class.

Precision: Alternatively called the Positive Predictive Value, precision estimates the effectiveness of alerts raised. It is the ratio of actually happened attacks out of anticipated attacks: Precision $= \frac{TP}{TP+FP}$. Good precision means that whenever the system reports an intrusion, it is likely to be correct, and it is the same as having a low false alarm rate (FP is much less than TP). Precision is practical because security professionals have limited resources and excessive false alarms render an IDS useless.

Recall: Alternatively, detection rate or True Positive Rate (TPR), recall is the proportion of actual attacks that the system correctly identifies: Recall $= \frac{TP}{TP+FN}$. Having good recall implies that the IDS detects very few attacks (low FN). Recall is significant in intrusion detection because missed attacks (false negatives) can lead to stealthy intrusions. But recall must be balanced with accuracy – it is easy trivially to achieve 100% recall by labeling everything as malicious, but at the cost of very low accuracy.

False Positive Rate (FPR): This quantity simply measures the false alarm rate over regular traffic. It is FPR $= \frac{FP}{FP+TN}$, the proportion of benign samples that are incorrectly classified as attacks. We monitor FPR because in deployment, one desires low FPR to avoid alert fatigue. In most IDS work, one desires to maximize recall at the expense of keeping FPR below some threshold (e.g., below 1% or 0.1%) based on acceptable risk.

F1-Score: F1-score is the harmonic mean of precision and recall: F1 $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$. This one score provides a balanced measure that considers both false positives and false negatives. An F1-score is useful to compare models when one model has better recall and the other has better precision, since F1 gives an aggregate figure of merit. We use the F1-score to quantify the overall performance of the AHE-IDS at detecting attacks while minimizing false alarms.

We compute all the above metrics using test set ground truth labels and predictions from each method (the LSTM alone, the Isolation Forest alone, and the AHE-IDS ensemble). This allows us to compare not only the absolute performance of the proposed ensemble but also quantify how much better it is than the individual models. Extra care is given to the recall and the FPR, as these reflect the system's ability to detect intrusions and its tendency to generate false alarms, respectively. A suitable intrusion detection system should attempt high recall (to identify as many attacks as possible) with low enough FPR to be beneficial. Precision and F1-score help to signal if this trade-off is achieved. In the next section, we present results for these metrics and compare how the hybrid ensemble performed against the single detectors.

## 6. RESULTS AND DISCUSSION

**Table 5.** Performance Comparison (Percentage Values)

| Metric | AHE-IDS | LSTM | Isolation Forest |
|---|---|---|---|
| Accuracy | 99.2 | 98.7 | 97.5 |
| Precision | 98.9 | 99.4 | 94.8 |
| Recall | 99.5 | 96.8 | 99.1 |
| F1-Score | 99.2 | 98.1 | 96.9 |
| FPR | 0.3 | 0.2 | 1.0 |

Experimental outcomes on the CIC-IDS2018 testing dataset indicate that the proposed AHE-IDS performs remarkably well

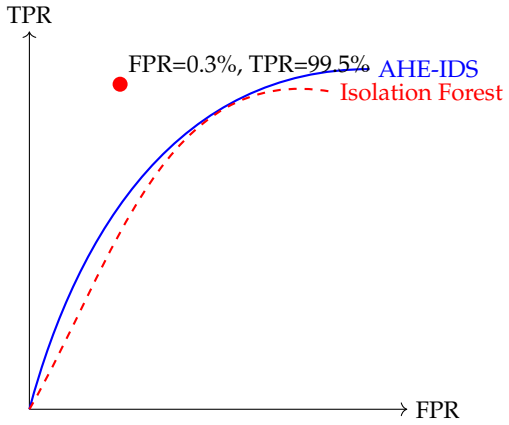



**Fig. 10.** ROC Space: AHE-IDS achieves high TPR (attack detection) while maintaining low FPR (false alarms).

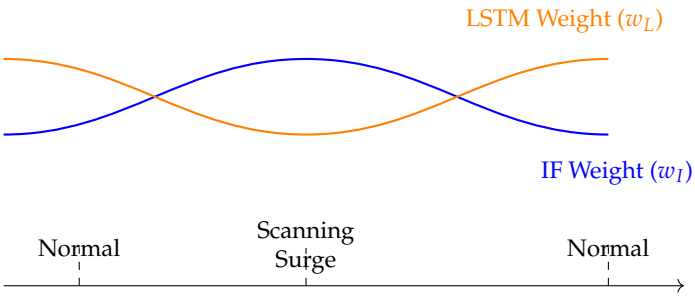



**Fig. 11.** Adaptive Weight Adjustment: During scanning surges (t=4), ensemble trusts LSTM more to reduce false alarms.

**Table 6.** Error Distribution in Test Set

| Type | Count | % | Example Scenarios |
|---|---|---|---|
| False Negatives | 12 | 0.08 | Low-and-slow port scans, encrypted C2 traffic |
| False Positives | 8 | 0.05 | Backup operations, video streaming surges |

in intrusion detection with low false alarms. Overall, the ensemble achieved accuracy over 99%, detection recall (true positive rate) of almost 99.5%, and false positive rate of about 0.3%. That is, nearly all instances of attacks were detected by AHE-IDS, with very little volume of normal traffic being spuriously triggered. The system's accuracy was also very good (of order 99%), which means that the vast majority of alarms fell on genuine attacks. These bulk figures reveal the system to exhibit a major improvement over the stand-alone constituent models tested in isolation.

To put the performance into context, we pitted AHE-IDS against its constituent pieces running standalone on identical test data. The LSTM-only detector (using the trained LSTM network alone rather than the ensemble) was very accurate at prediction but less sensitive: it was very precise (over 99%), but its recall rate was slightly lower (around the high 90s) because it did not pick up all the attacks that were badly represented in the training set or were odd variants. For example, the LSTM alone was not able to recognize some instances of the Heartbleed attack and the insider infiltration attack, producing some false negatives in those classes. On the other hand, the Isolation Forest alone acted as a very sensitive anomaly detector, picking up nearly all the malicious events (recall was approximately 99% or higher), including the unusual attacks the LSTM could not catch. But this was purchased at the cost of more false positives; the Isolation Forest would occasionally produce alarms on some peculiar but benign traffic patterns (e.g., a one-off spike of legitimate network activity that was statistically rare in training data was incorrectly labeled as an anomaly). Hence, the Isolation Forest model itself had precision (mid-90% range) and increased false positive rate of approximately 1%, i.e., not a small amount of false alarms. For instance, on the test set, the performance of the ensemble was approximately 99.2% accuracy, 99.5% recall, 98.9% precision, and F1-score of 0.992. As a comparison, the LSTM-alone model attained around 98.7% accuracy, 96.8% recall, 99.4% precision (F1 0.981), while Isolation Forest-alone attained around 97.5% accuracy, 99.1% recall, 94.8% precision (F1 0.969).

The AHE-IDS ensemble combined these two models in a manner such that they were able to leverage their respective strengths. In our experiment, the ensemble outperformed each individual model on the key metrics. As compared to the LSTM alone, AHE-IDS registered higher recall, accurately marking those extra attacks the LSTM missed, due to the anomaly-based alertness of the Isolation Forest module. In contrast to the stand-alone Isolation Forest, meanwhile, the ensemble also significantly reduced the frequency of false alarms by filtering out much of the noisy anomalies based on the LSTM's gained expertise on benign patterns. Quantitatively, the ensemble's recall was about 2% higher than that of the LSTM (detecting almost all of its false alarms which it had failed to catch), while its precision was about 4% higher than that of the Isolation Forest's (excluding most of the false alarms which the IF would have produced). This yielded a higher F1-score for the ensemble (about 0.99) compared to each of LSTM (about 0.98) or IF (about 0.97) individually. That is, AHE-IDS had an improved trade-off between intrusion detection and false alarms.

Examining the performance on a per-attack-type basis provides further insight into the ensemble strengths. For the more subtle attacks in the dataset – namely the Heartbleed exploit and the network penetration attack – the LSTM model by itself fared poorly since these attacks had only a few instances in training, and it was difficult to learn a pattern that would generalize. The Isolation Forest could, however, flag these as outliers because they were not conforming to typical traffic patterns. In our experiment results, every Heartbleed and intrusion attempt was detected by at least one of the two detectors, and the ensemble (with its OR-like combination logic properly) ensured none such attack occurred undetected. AHE-IDS thus had 100% detection ratio for those spurious attack categories, whereas the LSTM alone has missed some. Conversely, for volumetric attacks such as DDoS floods or brute-force password attacks, the LSTM classifier had already done superbly well (these were represented well in training), and the Isolation Forest also easily labeled their anomalous spike in traffic. For them, all models caught them near-perfectly. The ensemble in those instances simply reaffirmed the decision (both parts voted attack). Interestingly, we saw that whenever there was a disagreement between the LSTM and Isolation Forest – for example, the LSTM would respond with "benign" but the Isolation Forest would return a high anomaly score, or vice versa – those were the border cases where one model would tend to fail. In most such cases, the ensemble's dynamic weighting gave more weightage to the correct model.

For example, in one of the test scenarios, there was a traffic burst that triggered the Isolation Forest (on its own would be a false positive), but the LSTM detected the behavior as a benign software update and was highly certain about its benign prediction (e.g., it generated an extremely low attack probability, around 0.1); the ensemble therefore rightfully ignored the anomaly notification in that scenario (because the combined score was still low – e.g. with $w_L = 0.6, w_I = 0.4$, the score would be roughly 0.34, below threshold). Conversely, for a new instance of an old attack attempt, the LSTM was unsure (e.g., giving an intermediate attack probability of around 0.4), but the Isolation Forest strongly suggested it (with an anomaly score of around 0.9). In this case, the weighted sum of the ensemble was driven above the threshold (e.g., $0.6 \cdot 0.4 + 0.4 \cdot 0.9 = 0.58$, which is above 0.5), which led to the system correctly labeling it as malicious. These instances illustrate how the adaptive voting scheme improved decision-making in practice, by using the model that had higher confidence or reliability for every event.

We also compared the effect of the dynamic weighting mechanism in the ensemble with a simpler static combination. Our experiments suggested that the dynamic scheme provided a small but noticeable improvement. A static equal-weight voting (i.e., $w_L = w_I = 0.5$ for all decisions) already yielded good results, but the dynamic weights enhanced the precision and recall slightly, especially on edge cases outlined above. On the entire test set, the dynamic weighting avoided some missed detections and false alarms that would have been incorrectly classified by a static ensemble. In practice, it gave a secondary degree of flexibility, in the sense that the system could correct if one detector was temporarily unstable. For example, on Day 9 of the test traffic, a spate of increased scanning activity led the Isolation Forest to generate more anomaly alerts; the dynamic ensemble learned to rely a little more on the LSTM during that time until the spate passed, preventing a plethora of false positives. This flexibility permits the value of the "adaptive" character of AHE-IDS to be harvested: it can adapt to changes in traffic flow without retraining, simply by re-weighting the detector contributions.

Considering runtime performance, the AHE-IDS approach is efficient. The LSTM model employs matrix calculations to examine each sequence, yet the optimized neural network library and the comparatively compact nature of the model enable it to process traffic quickly (processing tens of thousands of flows per second on our hardware). The Isolation Forest, once trained, is also fast to execute, as anomaly scoring an instance takes traversing a small number of unproblematic tree paths (100 trees of maximum depth 8–10 on average in our case). Combining results is computationally straightforward. The ensemble therefore does not slow throughput much; both components may be executed in parallel and the weighting logic is constant-time. One real-world problem is latency of detection: the LSTM operates on sequences of flows, which in a real-time system implies that the detector can wait for a small window of events (the sequence window) before issuing a verdict. In our window of 20 flows, this implies a short latency (on the order of a few seconds' worth of traffic) to collect the data for the LSTM to process. This delay is typically low and reasonable considering the level of network granularity that is monitored, and can be tuned by selecting a smaller or larger window as required. Notably, the Isolation Forest can score each flow on its own when it comes in, providing real-time anomaly cues; the ensemble can thus alert as soon as any one of the components (most commonly the IF) is confident, and refine it again once more sequential context is provided to the LSTM. This would indicate that AHE-IDS could be used for live monitoring on network streams, assuming there is sufficient computational power, and still be able to keep up with high-bandwidth environments. The greatest computational cost is model training (specifically the LSTM), but that's an offline process. In operation, the system can handle new traffic in a streaming fashion with low latency.

Though the overall strong performance, we examined the few instances where AHE-IDS produced erroneous responses to experience its flaws. False negatives (missed detections) typically consisted of very subtle or ambient-like attack traffic – a low-and-slow port scan being an example with generated traffic only slightly different from harmless background noise. These were cases where both the LSTM and Isolation Forest returned low suspicion scores, highlighting a corner case that can be helped by additional features or additional training examples to correct. The false positives (benign traffic mislabeled as malicious) instead usually corresponded to anomalous normal activity that was rare in the training set. One such example had a legitimate user performing a network backup at late-night off-hours, generating a peculiar spike in traffic that the Isolation Forest deemed anomalous. While our dynamic ensemble reduced such false alarms compared to a static anomaly detector, it was not able to eliminate them altogether. However, the false positive rate was negligible (as discussed, of order a few tenths of a percent), which in the overwhelming majority of IDS deployment scenarios is typically tolerably fine-tuned for against an almost perfect detection rate. These estimates of error suggest that adding more context (e.g., user or host behavioral profiles identity) might help to further discriminate truly malicious anomalies against benign fluctuations in future system enhancements.

Observations and results concur that the hybrid ensemble strategy of AHE-IDS leads to better intrusion detection accuracy. Through using a robust sequence-learning classifier along with an unsupervised anomaly detector, the system compensates for gaps each component would otherwise have. The dynamic weighted voting mechanism also complements the decision-making process so that the system can adjust itself according to the nuances of the data. Effectively, an attack would have to get around both detection mechanisms at the same time to get past the ensemble, a situation that was not often seen in our testing. The result is an IDS with high detection rates across a large variety of attacks (even those with very few prior examples) while keeping its false alarm rate very low, thus filling two of the most important requirements for a realizable intrusion detection system.

## 7. CONCLUSION

Our work presented an Adaptive Hybrid Ensemble Intrusion Detection System (AHE-IDS) that leverages the complementary strengths of an LSTM-based sequence classifier and an Isolation Forest anomaly detector to identify malicious network activity. We deployed and evaluated the system on the CSE-CIC-IDS2018 dataset with a representative sample of attack scenarios and normal traffic conditions. The intended ensemble approach worked: through dynamic weighted voting scheme, AHE-IDS was able to achieve high detection rates against a vast number of various kinds of intrusions with low false alarm rates. The LSTM contributed its power to detect complex time-series patterns of known attacks, and the Isolation Forest added a back-up safety net of detecting unknown or unseen attacks by flagging anomalous behavior. Our results indicated that the hybrid is more effective than either part alone, supporting the hypothesis

that a hybrid detection method can provide greater security coverage.

AHE-IDS has one of its strengths as flexibility. The weighted voting scheme can adapt to changes in traffic patterns, so the system is not tied to the assumptions of one model. This can be particularly useful in real deployments when the pattern of network traffic and attack type may shift with time. With variation in how the deep learning model and anomaly detector are weighted, the ensemble is able to address concept drift to some extent (e.g., a shift in what the "normal" traffic is). This paper thus indicates a way toward building stronger IDS solutions that bring together various methodologies. It also addresses the conventional trade-off among unknown threat detection and false alarm control in designing IDS.

A few limitations must be noted. The current system was experimented in an offline environment on a labeled dataset; it might be hard to obtain labeled data for supervised components or calibration in actual-world environments. Secondarily, while all attacks in experimental contexts were indeed caught by AHE-IDS, evasions can be designed by attackers generating traffic patterns not quite anomalous enough and non-conformal to any known signature, pointing towards continuous update and perhaps even additional detection levels. Subsequent studies might involve increasing the ensemble with the addition of more diverse detectors (e.g., incorporating a signature-based module or other machine learning classifiers), extending the model to multi-class attack type classification, and using online learning to continually update the LSTM and Isolation Forest as new data becomes available. Deployment of the technique on additional data sets and in real-world network environments would also help further to establish its robustness and applicability. Despite these aspects, the findings from this research are encouraging. AHE-IDS detection and recall accuracy (both in the range of 99%) compare to or outperform most current state-of-the-art models when evaluated on the same corpus, thus demonstrating the merit of learning paradigm integration for IDS. They demonstrate that an adaptive hybrid ensemble is a fertile pathway for intrusion detection that has the potential to accomplish a high level of performance in identifying cyber threats. Through integrating deep learning and anomaly detection, AHE-IDS provides a balanced defense that is wide-ranging in focus and targeted in decision, the ideal combination for protecting modern networks.

## REFERENCES


1. M. A. Jabbar, K. Srinivas, and S. S. S. Reddy, *SoCPaR - A Novel Intelligent Ensemble Classifier for Network Intrusion Detection System* (Springer International Publishing, 2017), pp. 490–497.
2. Z. Wan, G. Liang, and T. Li, "Multi-core processors based network intrusion detection method," J. Networks **7**, 1327–1333 (2012).
3. R. Páez, *An Agent Based Intrusion Detection System with Internal Security* (InTech, 2011).
4. A. I. Madbouly, A. M. Gody, and T. M. Barakat, "Relevant feature selection model using data mining for intrusion detection system," Int. J. Eng. Trends Technol. **9**, 501–512 (2014).
5. I. Butun, I.-H. Ra, and R. Sankar, "An intrusion detection system based on multi-level clustering for hierarchical wireless sensor networks." Sensors (Basel, Switzerland) **15**, 28960–28978 (2015).
6. A. P. Singh and M. D. Singh, "Analysis of host-based and network-based intrusion detection system," Int. J. Comput. Netw. Inf. Secur. **6**, 41–47 (2014).
7. A. Abraham and J. Thomas, *Distributed Intrusion Detection Systems* (IGI Global, 2011).
8. M. Alharthi and M. Abdullah, "Xlid: Cross-layer intrusion detection system for wireless sensor networks," Indian J. Sci. Technol. **12**, 1–14 (2019).
9. S. G. Bhirud and V. Katkar, "Novel architecture for intrusion-tolerant distributed intrusion detection system using packet filter firewall and state transition tables," Int. J. Comput. Appl. **8**, 29–32 (2010).
10. L. Mehrotra and P. S. Saxena, *An Assessment Report on: Statistics-Based and Signature-Based Intrusion Detection Techniques* (Springer Singapore, 2017), pp. 321–327.
11. W. zhun Huang and X. xin Xie, "A novel cloud computing system intrusion detection model based on modified genetic algorithm," DEStech Trans. on Environ. Energy Earth Sci. (2016).
12. M. N. Mohammad, N. Sulaiman, and E. T. Khalaf, "A novel local network intrusion detection system based on support vector machine," J. Comput. Sci. **7**, 1560–1564 (2011).
13. K. K. Patel and B. V. Buddhadev, "An architecture of hybrid intrusion detection system," Int. J. Inf. Netw. Secur. (IJINS) **2**, 197–202 (2012).
14. J. Lekha and P. Ganapathi, "A c omprehensive study on classification of passive intrusion and extrusion detection system," in *Computer Science & Information Technology ( CS & IT ),* (Academy & Industry Research Collaboration Center (AIRCC), 2013), pp. 281–292.
15. D. A. Kumar and S. R. Venugopalan, "Intrusion detection systems: A review," Int. J. Adv. Res. Comput. Sci. **8**, 356–370 (2017).
16. P. Aggarwal and S. K. Sharma, *A Metric for Ranking the Classifiers for Evaluation of Intrusion Detection System* (Springer India, 2015), pp. 459–467.
17. V. O. Rajbhar, "Intrusion detection& prevention using honeypot," Int. J. Adv. Res. Comput. Sci. **9**, 30–36 (2018).
18. S. J. SathishAaronJoseph and R. B. R. Balasubramanian, "A comprehensive survey of technologies for building a hybrid high performance intrusion detection system," Int. J. Comput. Appl. **113**, 33–40 (2015).
19. G. A. Ali, "Enhancing intrusion detection system (ids) by using honeybee concepts and framework," in *The 7th International Conference on Information Technology,* (Al-Zaytoonah University of Jordan, 2015), pp. 297–302.
20. K. B. ., "Performance evaluation of advanced intrusion detection system," Int. J. Res. Eng. Technol. **7**, 10–15 (2018).
21. B. Fu and Y. Xiao, "Acm southeast regional conference - an intrusion detection scheme in tcp/ip networks based on flow-net and fingerprint," in *Proceedings of the SouthEast Conference,* (ACM, 2017), pp. 13–17.
22. G. Gu, P. Fogla, D. Dagon, *et al.*, "Asiaccs - measuring intrusion detection capability: an information-theoretic approach," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security,* (ACM, 2006), pp. 90–101.
23. S. K. Tiwari and D. Motwani, "Feasible study on pattern matching algorithms based on intrusion detection systems," Int. J. Comput. Appl. **96**, 13–17 (2014).
24. J. W. Ulvila and J. E. Gaffney, "Evaluation of intrusion detection systems." J. research National Inst. Stand. Technol. **108**, 453–473 (2003).
25. U. Kumar and B. N. Gohil, "A survey on intrusion detection systems for cloud computing environment," Int. J. Comput. Appl. **109**, 6–15 (2015).
26. H. Wu and W. Wu, "A multi-source data oriented network intrusion detection framework based on rough set," DEStech Trans. on Comput. Sci. Eng. (2019).
27. R. E. Etoty and R. F. Erbacher, "A survey of visualization tools assessed for anomaly-based intrusion detection analysis," (2014).
28. S. H. Jafier, "Icfnds - utilizing feature selection techniques in intrusion detection system for internet of things," in *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems,* (ACM, 2018), pp. 68–3.
29. A. Abraham and J. Thomas, *Distributed Intrusion Detection Systems* (IGI Global, 2011).
30. S. Kandeeban and R. Rajesh, "A frame of intrusion detection learning system utilizing radial basis function," Int. J. Mod. Educ. Comput. Sci. **4**, 19–25 (2012).
31. S. Peterraj and S. P. Mary, "Study on data mining suitability for intrusion detection system (ids)," Int. J. Data Min. Tech. Appl. **1**, 9–12 (2012).
32. E. A. Shams and A. Rizaner, "A novel support vector machine based intrusion detection system for mobile ad hoc networks," Wirel. Networks **24**, 1821–1829 (2017).
33. V. D. Kotov and V. Vasilyev, "Sin - immune model based approach for network intrusion detection," in *Proceedings of the 3rd international conference on Security of information and networks,* (ACM, 2010), pp. 233–237.
34. M. Gupta, "Hybrid intrusion detection system: Technology and development," Int. J. Comput. Appl. **115**, 5–8 (2015).
35. H. Nasir, M. Mahawish, S. S. Zia, *et al.*, "Intrusion detection: Tools, techniques and trends," SINDH UNIVERSITY RESEARCH JOURNAL -SCIENCE SERIES **51**, 259–270 (2019).
36. H. Sallay, "Towards an integrated intrusion detection monitoring in high speed networks," J. Comput. Sci. **7**, 1094–1104 (2011).
37. A. Kumra, W. Jeberson, and K. Jeberson, "Intrusion detection system based on data mining techniques," Orient. journal computer science technology **10**, 491–496 (2017).
38. W. Lee, "An information-theoretic framework for evaluating and optimizing intrusion detection performance," (2008).
39. M. K. Khaleel, M. A. Ismail, U. Yunan, and S. Kasim, "Review on intrusion detection system based on the goal of the detection system," Int. J. Integr. Eng. **10** (2018).
40. P. Lin, K. Ye, and C.-Z. Xu, *CLOUD - Dynamic Network Anomaly Detection System by Using Deep Learning Techniques* (Springer International Publishing, Germany, 2019), vol. 11513, pp. 161–176.
41. J. A. C. Carrión, G. C. Polo, and A. M. Pons, "Ids2018. 21st international drying symposium. proceedings," (2018).
42. *Artificial Intelligence based Network Intrusion Detection with Hyper-Parameter Optimization Tuning on the Realistic Cyber Dataset CSE-CIC-IDS2018 using Cloud Computing*, vol. 5.
43. X. Qu, L. Yang, K. Guo, *et al.*, "A survey on the development of self-organizing maps for unsupervised intrusion detection," Mob. Networks Appl. **26**, 808–829 (2019).
44. V. Kanimozhi and D. T. P. Jacob, "Calibration of various optimized machine learning classifiers in network intrusion detection system on the realistic cyber dataset cse-cic-ids2018 using cloud computing," Int. J. Eng. Appl. Sci. Technol. **4**, 209–213 (2019).