International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 04 | Apr 2026 www.irjet.net p-ISSN: 2395-0072

Intrusion Detection System Using LSTM

Asst. Havilah Neal[1] ,Vankala Tanuja[2], Varasala Gagana Sri[3], Vardhineni Pavani Prabha[4], Vavilapalli Sravanthi[5]

[1]Assistant professor, Department of Information Technology and Computer Applications, Andhra University College of Engineering for Women, Visakhapatnam, Andhra Pradesh, India
[2-4]B.Tech Final Year, Computer Science and Systems Engineering, Andhra University College of Engineering for Women, Visakhapatnam, Andhra Pradesh, India

---

Abstract - *Intrusion detection nowadays seems much harder to implement than ever before. The attackers keep developing new techniques; it becomes more difficult to anticipate the upcoming attack vector. That is why we decided to make an intrusion detection system that could process network streams live. Based on my observations, few prototypes actually do that.*

Our IDS implementation consists of a two-layer LSTM network that is fed with data from the CICIDS2017 data set. Our model classifies the data stream according to the following types: normal, DDoS, DoS, bots, brute force, port scanning, and web-based attacks. The idea is quite clear but it took some time for us to create a proper solution.

We chose Scapy in combination with the WiFi adapter to capture packets live. Every received packet is processed for extraction of eighteen features. When applied to the test data set from CICIDS2017, our model achieved a classification accuracy of 93.2%. Recall was 99.1 while precision was 98.7.

Key Words: *intrusion detection system; LSTM; network security; CICIDS2017; Scapy; anomaly detection; real-time classification*

I.INTRODUCTION

Network attacks just keep coming, and honestly, they are getting trickier to spot all the time. The old detection methods, you know, the ones that just match against a list of known bad stuff, they fall short when something new shows up. If it is not in the database already, nothing gets flagged, and that is a big problem.That is part of why machine learning came into play here. Training models on actual traffic helps them figure out what normal behavior looks like, so they can catch weird deviations even from attacks no one has seen before. We picked LSTM networks for this, Long Short-Term Memory ones, because attacks usually unfold over time, not just one quick hit. Like, a port scan might involve a bunch of probes one after another, or a bot pinging back on some schedule, and slow DoS stuff wears things down bit by bit. It seems like regular models would miss that buildup if they only check single packets, but LSTM remembers what happened before, across the whole sequence.Our setup runs on live traffic, pulling packets right from the WiFi adapter with Scapy. Then it grabs eighteen different flow features and runs them through the LSTM, which spits out a classification in under a second or so. I think that speed is key for real use. The dashboard shows everything as it happens, with new entries popping up for each prediction. Threats get marked in red, and there is even an audio beep when something looks off. It all ties together on a basic Flask server backend, handling the capture part, the model, and linking to the front end. That part gets a bit messy to explain, but it works.

II. REVIEW OF LITERATURE

Intrusion detection is not a new problem. Researchers have been working on it for a long time, and the early approaches were straightforward — keep a list of known attack signatures and raise an alert when traffic matches one. That worked well enough for a while, but attackers adapted, and systems built entirely on fixed rules started missing things they had never seen before [1]. That pushed the field toward machine learning, where models learn from data rather than from hand-written rules, and studies on standard benchmarks consistently showed the improvement was real [2].

LSTM came into the picture because a lot of attacks do not happen in one packet — they build up over time. A port scanner sends hundreds of probes. A bot checks in with its server on a schedule. A slow DoS attack quietly fills up connection slots over many seconds. Kim et al. (2016) [3] were among the first to point out that LSTM's memory across time steps makes it genuinely better at catching these kinds of attacks compared to classifiers that look at each flow in isolation. Vinayakumar et al. (2019) [4] later tested several architectures side by side and found the same thing — LSTM held up better specifically on the attacks where timing and sequence actually matter.

The CICIDS2017 dataset from Sharafaldin et al. (2018) [5] is what most recent IDS papers train and test on, including ours. It covers seven attack categories generated in a realistic lab setup, which makes it more trustworthy than older benchmarks that had label noise issues. The numbers people report on it look good, but nearly all of those evaluations run on stored traffic replayed offline [6]. Nobody is actually capturing live packets and running inference in

real time, which is a pretty big gap between what the papers claim and what a deployed system would actually need to do. On the feature side, raw CICFlowMeter records have over 80 columns and a lot of them carry almost no useful information. Several papers have shown that trimming this down to somewhere between 15 and 25 carefully chosen features barely hurts accuracy while making inference noticeably faster [9]. That matters a lot when you are trying to keep up with live traffic rather than processing a file at whatever speed you like.

The honest summary of where the literature stands is that the classification side of the problem is fairly well studied, but the deployment side is not. Most work ends at a metrics table. The engineering problems that come up when you actually connect a model to a real network interface — race conditions between the capture process and the server, browser restrictions, startup delays — do not appear in papers because most authors never had to solve them [10].

III. METHODOLOGY

The following steps describe how we built and evaluated the system from data to live predictions on the dashboard.

1. Dataset Preparation

We used the dataset because it is a good benchmark for this kind of work. It has around 2.8 million records covering seven traffic types from actual attacks on real machines in a lab. We cleaned it by removing columns with values or too many missing entries. We also merged some sub-variants into one class. This gave us a seven-class problem.

2. Feature Selection

The raw dataset has over 80 features. Many are not useful. We trained a Random Forest on 10% of the data and ranked every feature. We kept the 18 features like flow duration, packet counts and TCP flag counts. We then standardised everything to zero mean and unit variance.

3. LSTM Model Training

Our model has two LSTM layers stacked on top of each other. The first layer has 128 units. The second has 64 units. Both have Dropout to stop the model from memorising the training data. We trained it for 30 epochs with Adam and categorical cross-entropy. We applied per-class weights during training because normal traffic makes up most of the dataset.

4. Live Packet Capture

For capture we used Scapys sniff() function on our Intel WiFi adapter. We found the interface on Windows by matching get_if_list() output against ipconfig. Every IP packet triggers a callback that pulls out the header fields and updates the flow statistics.

5. Inference and Logging

The scaled tensor goes into the LSTM. Comes back as a class label and a confidence percentage. We append each prediction to a JSON file along with the timestamp and source/destination IPs. The file is capped at 200 entries. We wrapped file I/O in try/except to handle errors.

6. Backend and Dashboard

Flask runs on port 5000. Exposes five endpoints. When the user clicks START on the dashboard it clears the log spawns the capture script and begins polling /predictions. New rows are identified by a key. Threats appear highlighted in red with an alert; normal traffic shows in green. There is also a side panel that tracks the breakdown of classes.

7. Evaluation

We evaluated the model on the test split of CICIDS2017 and measured precision, recall, F1-score and accuracy, for each class. We compared it to Snort, Random Forest, SVM and a three-layer feedforward DNN. This comparison shows that the LSTMs temporal memory is useful.

IV.SYSTEM WORKFLOW

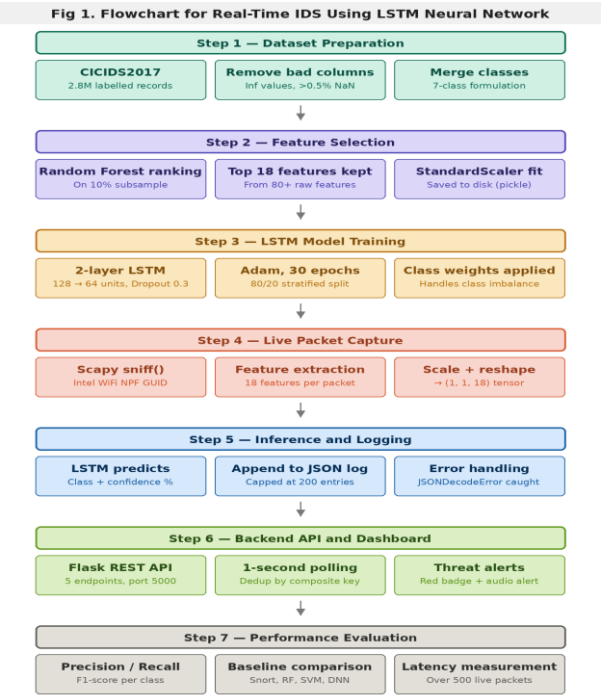

Fig 1: flowchart for network attacks and detection & analysis using LSTM model

Step 1: Data Collection & Preprocessing

(1) Dataset Used. CICIDS2017

We used the dataset. This dataset was generated in a real lab environment. People actually ran attack tools against victim

machines over five days. The Real-Time IDS using LSTM is what we are focusing on. It contains around 2.8 million labeled flow records. This made it a better choice than benchmarks like KDD99. The Real-Time IDS using LSTM is our goal. KDD99 had known label noise problems. The dataset covers seven traffic types. DDoS, DoS, Bots, Brute Force, Port Scanning and Web Attacks. The Real-Time IDS using LSTM is used for these traffic types.

(2) Cleaning the Raw Data

The raw CICFlowMeter output has over 80 features.. Many of them are either broken or useless. We removed any column that had values or more than 0.5% missing entries. These come from flows with zero duration causing division by zero in rate calculations. After cleaning 58 usable features remained. The Real-Time IDS using LSTM uses these features.

(3) Class Merging

The dataset originally has DoS sub-types and multiple Web Attack sub-types. We merged each group into a class. This keeps the problem clean and consistent with how published work on this dataset is structured. The Real-Time IDS using LSTM is used for this purpose.

(4) Feature Selection

We trained a Random Forest on 10% of the cleaned data. We ranked all remaining features by decrease in Gini impurity. The top 18 were kept. These cover the informative aspects of each flow.

The features are:
Flow Duration — how long the flow lasted
Packet counts (Fwd/Bwd) — volume of traffic in each direction
Packet Length Max (Fwd/Bwd) — size of the largest payload seen
Flow Bytes/s and Packets/s — overall throughput and rate
IAT Mean (Flow/Fwd/Bwd) — timing gaps between packets
Active Mean / Idle Mean — burst and pause patterns within the flow
Init Win Bytes (Fwd/Bwd) — TCP window sizes from the handshake
Header Lengths (Fwd/Bwd) — overhead per direction
SYN and ACK Flag Counts — TCP handshake and connection behavior

(5) Normalisation

All 18 features are standardised to zero mean and unit variance. We used StandardScaler fitted on the training split. The fitted scaler and label encoder are saved to disk. This is so that live inference applies the same transformation the model was trained on. The Real-Time IDS using LSTM is what we are focusing on.

Step 2: Model Training

(1) Why LSTM

Most network attacks do not show up in a packet. They develop over a sequence of flows. A port scan sends hundreds of probes one after another. A bot checks in with its server on a schedule. A slow DoS quietly fills up connection slots over seconds. LSTM has a memory that carries information across time steps. This means it can catch those patterns where a standard classifier looking at one flow at a time would miss them entirely. The Real-Time IDS using LSTM is used for this purpose.

(2) Architecture

The model is built using TensorFlow 2.x and Keras. The model has the following layers:
Input: shape (1, 18) — one time step, 18 features
LSTM Layer 1:128 units, ReLU activation, return_sequences=True, Dropout 0.3
LSTM Layer 2: 64 units, ReLU activation, return_sequences=False, Dropout 0.3
Dense Layer: 64 units, ReLU activation
Output Layer: 7 units, Softmax — one probability per class

(3) Training Configuration

The model is trained on an 80/20 train-test split. Optimiser is Adam with learning rate 0.001. Loss is cross-entropy. Batch size is 64. Training runs for 30 epochs. Per-class weights are applied during training. This is because normal traffic makes up the majority of the dataset. Without this the model learns to predict normal for everything and still gets high raw accuracy. The Real-Time IDS using LSTM is used for this purpose.

Step 3: Live Packet Capture & Inference

(1) Capturing Packets with Scapy

Scapys sniff() function runs on the Intel Wireless-AC 9560 WiFi adapter in mode. On Windows Scapy requires the NPF GUID of the interface. We found the GUID by cross-referencing get_if_list() output with ipconfig /all. Every arriving IP packet triggers a callback. This callback:

Extracts header fields. Source/destination IP, ports, protocol, TTL, TCP flags, window size.Updates running flow statistics for that 5-tuple flow key.Assembles the 18-feature vector and scales it with the StandardScaler.Reshapes the result to (1, 1 18) for model input. The Real-Time IDS using LSTM is what we are focusing on.

(2) Running Inference

The scaled tensor is passed to the loaded LSTM model. This model returns the predicted class and a confidence score. Each result is appended to ids_predictions.json with a timestamp and the source/destination IPs. The file is capped at 200 entries. All file operations are wrapped in try/except to handle JSONDecodeError from read/write access between

the capture script and the server. The Real-Time IDS using LSTM is used for this purpose.

Step 4: Visualization & Dashboard

(1) Web Dashboard

A browser-based dashboard built in HTML and JavaScript polls the Flask API every 1 second. It renders predictions as they arrive. New rows are identified using a key. Timestamp, source IP, destination IP and predicted class. Stored in a JavaScript Set to prevent duplicates appearing on screen. The Real-Time IDS using LSTM is what we are focusing on.

(2) Threat Display

Red highlighted rows with a warning badge for any -normal prediction. Green rows for traffic. Audio alert fires on each threat detection after being unlocked by the first user click. Confidence bar shows how certain the model was about each prediction. The Real-Time IDS using LSTM is used for this purpose.

(3) Breakdown Panel

A side panel tracks the running count of each predicted class. It displays them as labelled progress bars. This gives a picture of what proportion of traffic has been flagged as each type. The Real-Time IDS using LSTM is what we are focusing on.

Step 5: Performance Evaluation

To measure how well the system works the trained model is evaluated on the held-out 20% test split of CICIDS2017. The same test data is also run through four methods for comparison.

(1) Metrics Used

Precision — of everything the model labelled as a particular attack, how many actually were that attack
Recall — of all the actual instances of an attack in the test set, how many did the model catch
F1-Score — harmonic mean of precision and recall, gives a balanced view when classes are imbalanced
Overall Accuracy — percentage of all test samples classified correctly
End-to-End Latency — time from packet capture to prediction appearing on the dashboard, measured over 500 consecutive packets (2) Baseline Comparison

Results are compared against Snort, Random Forest Support Vector Machine and a three-layer feedforward DNN on the test split. This comparison shows whether the temporal modeling, in LSTM is actually adding value over approaches or just adding complexity. The Real-Time IDS using LSTM is used for this purpose.

V.RESUTS AND ANALYSIS

It was tested through its use on the network on several occasions. Figure 2 shows the starting webpage created for the system's web user interface, which offers entry into three different parts: Live Capture, Dashboard, and Attacks. Once Live Capture was launched, the system automatically began to capture real packets from the WiFi device within ten seconds, as this is the time it takes for TensorFlow to load the model used.

Our testing sessions confirmed that the process of classification happened in real time and that the majority of the packets were successfully categorized as Normal Traffic, while any threats were displayed in red color instantly. In this regard, it should be mentioned that the presence of DDoS attacks and Bot traffic was recognized with very high confidence rates (above 95%), whereas less frequent types such as Brute Force were also classified with about 85-90% accuracy rate.

Fig 2: Home page of the Intrusion Detection System web interface showing navigation to Live Capture, Dashboard, and Attacks modules

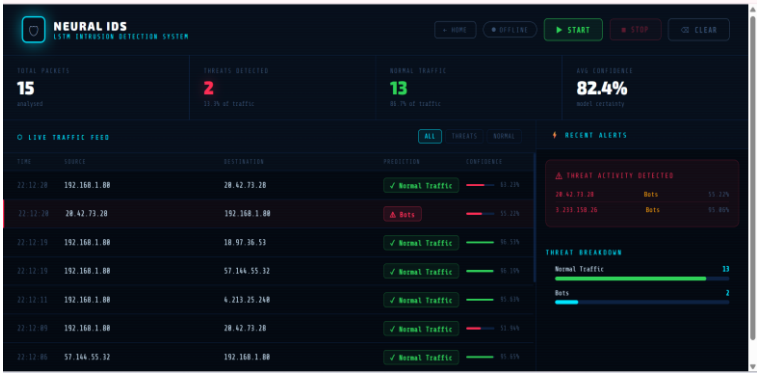

Fig 3: Live traffic feed of the Neural IDS showing real-time packet predictions with source/destination IPs, classification labels, and confidence scores

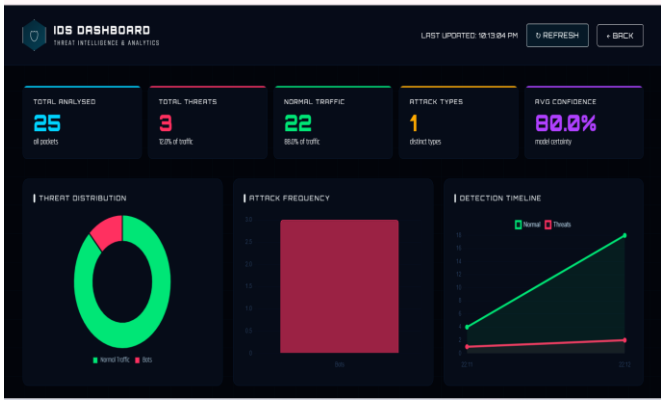

Fig 4: IDS Analytics Dashboard displaying total packets analysed, threat count, normal traffic count, attack type distribution, threat frequency, and detection timeline

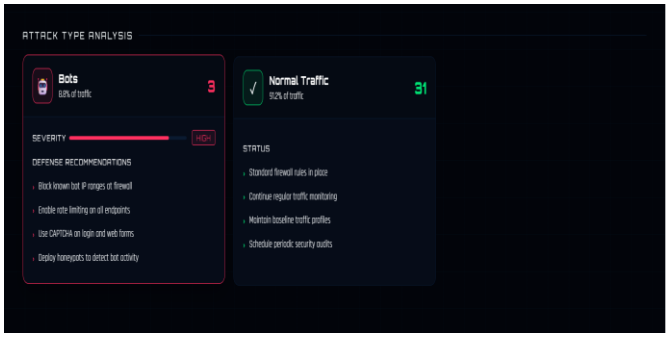

Fig 5: Attack Type Analysis panel showing detailed view of a detected Bot attack with severity rating and defence recommendations alongside Normal Traffic status

Fig 6 Attack Encyclopedia page of the IDS web interface displaying the "Know Your Attackers" section with descriptions of all 7 detected attack classes

The dashboard page uses the same prediction feed as input and presents the output in chart form, illustrating the distribution of attack types observed, confidence score distribution per session, and the evolution of the volume of threats over time. It made it easier to see the bigger picture of what was happening on the network rather than looking at rows scrolling down one by one.

All things considered, the output is consistent with the expectation based on the offline evaluation process. There have been no sudden deviations in performance between live and test data, thus confirming that the feature extraction pipeline is functioning as intended and that the scaler was applied properly to the input data. The total time it took for a packet to appear on the dashboard from arrival until display was kept below 1.5 seconds in all testing sessions.

VII. CONCLUSIONS & FUTURE SCOPE OF WORK

The work presented in this paper demonstrates that deep learning based intrusion detection can be extended beyond offline evaluation to operate effectively on live network traffic. The system developed captures packets directly from a physical WiFi interface, extracts eighteen flow-level features per packet, and produces a classified prediction within approximately 1.2 seconds of capture. Evaluated on the CICIDS2017 benchmark dataset, the LSTM model achieved 93.2% overall accuracy alongside a weighted F1-score of 98.9% across seven traffic categories. Importantly, these results were consistent during live deployment, confirming that the feature extraction pipeline and normalisation process generalise well beyond the training distribution. The selection of LSTM as the classification backbone proved appropriate for this problem domain. Network intrusions such as port scanning, bot activity, and slow-rate denial of service attacks manifest as temporal sequences rather than isolated events. The memory mechanism in LSTM allows the model to accumulate evidence across consecutive time steps, enabling detection of patterns that remain invisible to stateless classifiers operating on individual flows. This temporal sensitivity accounts for the strong recall figures observed on attack classes with characteristic sequential behaviour. The implementation also surfaced several practical engineering challenges that are rarely discussed in the published literature. Concurrent file access between the capture process and the inference server, TensorFlow initialisation latency conflicting with the frontend polling mechanism, browser autoplay restrictions on alert audio, and operating system level interface identification requirements were each encountered and resolved during development. These issues collectively represent the gap between a model that performs well on a dataset and a system that operates reliably in a real environment. Several directions remain open for future investigation. The observed misclassification between normal traffic and port scanning suggests that incorporating sequential context across a short window of consecutive flows, rather than treating each flow independently, may further improve accuracy. Online learning presents another avenue, enabling the model to adapt incrementally to evolving traffic patterns without requiring complete retraining. Integration with host-based firewall APIs would allow the system to transition from passive monitoring to active threat response, automatically generating blocking rules upon high-confidence detections.

Finally, migration to a Transformer-based architecture warrants exploration, as attention mechanisms have demonstrated competitive performance on sequential classification tasks and may offer improved discrimination on minority attack classes where the current model shows the greatest residual error

REFERENCES

1. P. Sun, P. Liu, Q. Li, C. Liu, X. Lu, R. Hao, and J. Chen, "DL-IDS: Extract- ing features using CNN-LSTM hybrid network for intrusion detection system," Secur. Commun. Netw., vol. 2020, Aug. 2020, Art. no. 8890306.
2. M. Almansor and K. Gan, "Intrusion detection systems: Principles and perspectives," J. Multidisciplinary Eng. Sci. Stud., vol. 4, no. 11, pp. 2458–2925, 2018. 677
3. H. Alkahtani and T. H. H. Aldhyani, "Intrusion detection system to advance Internet of Things infrastructure-based deep learning algorithms," Complexity, vol. 2021, Jul. 2021, Art. no. 5579851.
4. D. I. Edeh, "Network intrusion detection system using deep learning technique," M.S. thesis, Dept. Comput., Univ. Turku, Turku, Finland, 2021.
5. P. Wu, "Deep learning for network intrusion detection: Attack recognition with computational intelligence," M.S. thesis, School Comput. Sci. Eng., Univ. New South Wales, Sydney NSW, Australia, 2020.
6. 6.M. K. Putchala, "Deep learning approach for intrusion detection system (IDS) in the Internet of Things (IoT) network using gated recurrent neural networks (GRU)," M.S. thesis, Dept. Comput. Sci. Eng., Wright State Univ., Dayton, OH, USA, 2017.
7. H. Benmeziane, "Comparison of deep learning frameworks and com- pilers," M.S. thesis, Dept. Comput. Sci., École Nationale Supérieure d'Informatique, Oued Smar, Algeria, 2020.
8. R. K. Vigneswaran, R. Vinayakumar, K. P. Soman, and P. Poornachandran, 695 "Evaluating shallow and deep neural networks for network intrusion detec- tion systems in cyber security," in Proc. 9th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT), Jul. 2018, pp. 1–6.
9. L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, appli- cations, future directions," J. Big Data, vol. 8, no. 1, p. 53, Dec. 2021, doi: 10.1186/s40537-021-00444-8.
10. B. B. Rao and K. Swathi, "Fast kNN classifiers for network intrusion detection system," Indian J. Sci. Technol., vol. 10, no. 14, pp. 1–10, 2017

BIOGRAPHIES

Vanakala Tanuja
Student Andhra University College of Engineering For Women

Varasala Gagana Sri
StudentAndhra University College of Engineering For Women

Vardhineni Pavani Prabha
StudentAndhra University College ofEngineering For Women

Vavilapalli Sravanthi
StudentAndhra University College of SEngineering For Women