International Journal of Innovative Research in Engineering
Volume 5, Issue 3 (May-June 2024), PP: 209-216.
https://www.doi.org/10.59256/ijire.20240503026
www.theijire.com
ISSN No: 2582-8746

Enhancing Intrusion Detection Systems with Recurrent Neural Networks in D23 Deep Learning

Prabhavathi Krishnegowda[1], Prathiksha K P[2], Sinchanakumar K[3], Yashas A P[4], Yashwanth Gowda R S[5]
[1] *Assistant Professor Department of ECE, BGS Institute of Technology, Adichunchanagiri University, B. G. Nagara, Karnataka, India.*
[2,3,4,5] *UG Student, Department of ECE, BGS Institute of Technology, Adichunchanagiri University, B. G. Nagara, Karnataka, India.*

How to cite this paper:
Prabhavathi Krishnegowda[1], Prathiksha K P[2], Sinchanakumar K[3], Yashas A P[4], Yashwanth Gowda R S[5] "Enhancing Intrusion Detection Systems with Recurrent Neural Networks in D23 Deep Learning", IJIRE-V5I03-209-216.

Abstract: *Strong cybersecurity safeguards are increasingly necessary as the digital landscape changes to protect sensitive data and vital infrastructure. To detect and reduce any risks to network security, intrusion detection systems, or IDS, are essential. To improve the precision and effectiveness of anomaly detection, this paper suggests a novel method of intrusion detection that makes use of recurrent neural networks (RNNs). Standard intrusion detection systems (IDS) frequently depend on static rule-based systems, which may find it difficult to adjust to changing and dynamic cyber threats. On the other hand, sequence-retention neural networks, or RNNs, they have a special capacity to recognize patterns and temporal connections in sequential input. Traditional IDS methods are contrasted with the suggested system, which is assessed using common datasets. The RNN-based IDS is successful at detecting known and unknown threats while reducing false positives, according to preliminary data.*

I.INTRODUCTION

It is impossible to exaggerate the significance of protecting information systems from cyberattacks in a time of ubiquitous digital connectivity. As technology advances, so do the methods employed by malicious actors seeking to exploit vulnerabilities in networks, systems, and applications. Intrusion Detection Systems (IDS) stand as a critical line of defense, acting as vigilant guardians against unauthorized access, data breaches, and other forms of cyberattacks. Traditional IDS, which frequently depend on static rule-based techniques, have several challenges from the dynamic and linked structure of modern computer networks. Rules-based systems are difficult to adjust to the changing landscape of cyber threats, notwithstanding their effectiveness in some situations. Researchers are looking into the possibilities of machine learning methods, especially neural networks, in the field of intrusion detection since there is a growing demand for more clever and flexible solutions.

The foundation of our research lies in the recognition that network traffic data inherently exhibits temporal dependencies and sequential patterns. RNNs, with their ability to capture long-range dependencies in sequential data, present a promising avenue for enhancing intrusion detection precision. The LSTM architecture, a variant of RNN, has demonstrated exceptional performance in tasks involving sequential data analysis, making it a suitable candidate for the enhancement of the capabilities of IDS. Intrusion detection May be divided roughly into two main categories: signature-based detection and anomaly-based detection. Signature-based detection relies on a predefined set of patterns or signatures associated with known threats. While effective against known attacks, this strategy might be ineffective when faced with previously unseen or sophisticated threats. Anomaly-based detection, on the other hand, focuses on identifying deviations from normal system behavior.

The proposed intrusion detection system aims to strike a balance between these two approaches by leveraging the temporal understanding of LSTM networks. The ability of LSTMs to capture and remember long-range dependencies in sequences enables the system to discern normal patterns of network behavior from anomalous activities. By training the model on labeled datasets that encompass both normal and malicious network activities, the system learns to recognize subtle deviations indicative of potential intrusions.

II.LITERATURE REVIEW

Intrusion Detection Systems (IDS) constitute critical components within the cybersecurity framework, tasked with the pivotal role of fortifying computer networks and systems against an array of malicious activities. This paper endeavors to provide an exhaustive examination and analysis of various methodologies and techniques employed in contemporary IDS practices, amalgamating insights gleaned from a plethora of scholarly contributions in the field.

The seminal work by Halima and Sundarakantham (2019) posits a novel approach to IDS, harnessing the power of Machine Learning (ML) techniques such as Support Vector Machine (SVM) and Naïve Bayes. Through meticulous experimentation using the NSL-KDD dataset, their study underscores the efficacy of SVM in intrusion detection, outshining traditional methods. Building upon this foundation, Dali et al. (2020) undertake a comprehensive survey, elucidating diverse IDS mechanisms and delineating their applications across various domains. Their discourse encompasses network-based IDS, cloud-based IDS, and the nuanced contributions each subtype offers to the overarching cybersecurity paradigm.

Further enriching the discourse, Raghunath and Mahadeo (2018) introduce the Network Intrusion Detection System (NIDS), leveraging sophisticated data mining techniques for automated attack detection. Their seminal contributions showcase the efficacy of unsupervised anomaly detection and association pattern analysis in fortifying network resilience against emergent threats. Meanwhile, Almi'ani et al. (2019) proffer an innovative perspective, championing the utilization of a clustered version of the Self-Organized Map (SOM) network within IDS architecture. Their holistic approach addresses the perennial challenges of sensitivity and processing time, thereby enhancing the robustness of intrusion detection frameworks.

Complementing these efforts, Benaicha and Saoudi (2018) advocate for the integration of Genetic Algorithm (GA) methodologies into IDS frameworks, facilitating enhanced detection rates and reduced false positives. Through meticulous experimentation on benchmark datasets, their study underscores the symbiotic relationship between IDS and GA, ushering in a new era of efficacy in intrusion detection. Meanwhile, Osken and Yildirim (2019) embarked on a systematic mapping study, delving into the burgeoning domain of IDS with Deep Learning. Their meticulous analysis unveils the prevalent trends, algorithmic preferences, and selection criteria driving the adoption of Deep Learning methodologies within the IDS milieu.

In a parallel vein, Mukherjee et al. (2020) conducted an exhaustive survey, meticulously dissecting host-based and network-based IDS architectures. Their comprehensive analysis sheds light on the characteristics and efficacy of various detection mechanisms, offering invaluable insights into the evolving threat landscape. Meanwhile, Krishna et al. (2020) pivot towards the implementation realm, propounding an IDS framework imbued with Deep Learning capabilities. Through the seamless integration of a Multi- Layer Perceptron trained on benchmark datasets, their approach demonstrates remarkable accuracy in real-time intrusion detection and prevention.

In the realm of theoretical discourse, Shah (2018) undertakes a comparative analysis of Knowledge-Based and Anomaly-Based Intrusion Detection approaches, elucidating their respective strengths and weaknesses. Moreover, Shah

proposes a Unified Threat Prevention Engine (UTPE) as a novel framework to reconcile the inherent trade-offs between detection efficacy and computational overhead. Similarly, Bashir and Chachoo (2019) traverse the theoretical landscape, surveying existing IDS typologies, techniques, and architectural paradigms. Their discourse navigates through the intricacies of IDS evolution, underscoring the pressing research challenges and ripe opportunities inherent within the domain.

In summation, this paper serves as a compendium of knowledge, synthesizing insights gleaned from an eclectic array of scholarly endeavors in the field of IDS. By amalgamating theoretical discourse with practical implementations, it endeavors to offer a holistic perspective on the contemporary landscape of intrusion detection, affirming its pivotal role in fortifying the cybersecurity ecosystem against an ever-evolving threat landscape.

III.PROPOSED METHODOLOGY

Recurrent Neural Network (RNN)-based Intrusion Detection Systems (IDS) leverage the power of neural networks to detect and prevent malicious activities in computer networks. Unlike traditional IDS, which rely on predefined rules or signatures to identify attacks, RNN-based IDS can learn complex patterns and behaviours from network traffic data, making them more adaptive and effective in detecting unknown or evolving threats. In a RNN-based IDS, network traffic data is fed into the neural network, which consists of recurrent layers capable of capturing temporal dependencies and sequences in the data.

The RNN processes the sequential nature of network traffic, such as packet arrival times, packet payloads, and protocol headers, to extract meaningful features and patterns indicative of normal or malicious behaviour. Once trained, the RNN-based IDS can continuously monitor network traffic in real-time, analysing incoming data streams and alerting security administrators to suspicious or potentially malicious activities.

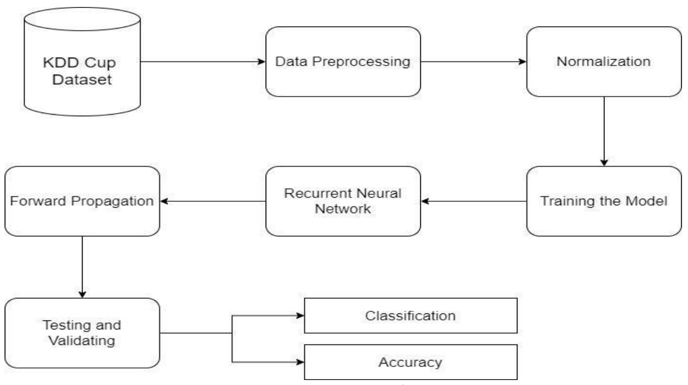
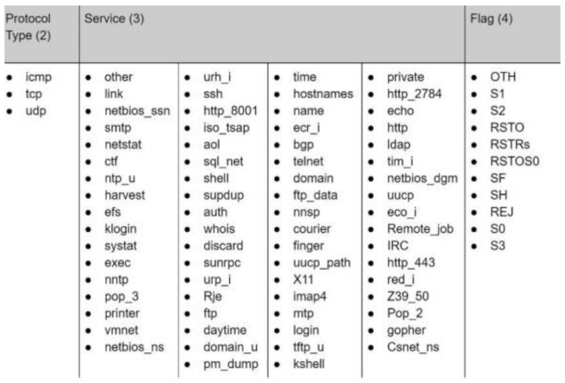


Fig.3.1 Block Diagram of Proposed System

Data Representation:

Network traffic data, such as packet headers or flow records, is collected from the network or network devices. This data is typically represented as sequential time-series data as shown in the Table.3.1, where each data point represents a network event or activity.



Protocol Type (2)	Service (3)				Flag (4)
• icmp	• other	• urh_i	• time	• private	• OTH
• tcp	• link	• ssh	• hostnames	• http_2784	• S1
• udp	• netbios_ssn	• http_8001	• name	• echo	• S2
	• smtp	• iso_tsap	• ecr_i	• http	• RSTO
	• netstat	• aol	• bgp	• ldap	• RSTRs
	• ctf	• sql_net	• telnet	• tim_i	• RSTOS0
	• ntp_u	• shell	• domain	• netbios_dgm	• SF
	• harvest	• supdup	• ftp_data	• uucp	• SH
	• efs	• auth	• nnsp	• eco_i	• REJ
	• klogin	• whois	• courier	• Remote_job	• S0
	• systat	• discard	• finger	• IRC	• S3
	• exec	• sunrpc	• uucp_path	• http_443	
	• nntp	• urp_i	• X11	• red_i	
	• pop_3	• Rje	• imap4	• Z39_50	
	• printer	• ftp	• mtp	• Pop_2	
	• vmnet	• daytime	• login	• gopher	
	• netbios_ns	• domain_u	• tftp_u	• Csnet_ns	
		• pm_dump	• kshell		

Table.3.1 Feature of dataset.

Preprocessing:

The raw network data is pre-processed to extract relevant features and convert it into a suitable format for input to the RNN model. This may involve normalization, discretization, or encoding of categorical variables to prepare the data for training.

Model Architecture:

Recurrent neural network cells, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) cells, are layered in one or more layers to form the RNN model architecture. The purpose of these cells is to record temporal relationships and patterns in sequential data.

Training:

The RNN model is trained on labelled network traffic data, where each data point is labelled as either normal or malicious. During training, the model learns to differentiate between normal and anomalous network activities by adjusting its internal parameters through backpropagation and gradient descent.

Detection:

Once trained, the RNN-based IDS can be used to detect intrusions or anomalies in real-time network traffic. As new network data is fed into the model, it analyses the temporal patterns and predicts whether the observed activity is normal or indicative of a potential security threat.

Adaptation and Feedback:

The RNN-based IDS may incorporate feedback mechanisms to adapt and improve its detection capabilities over time. This could involve retraining the model with updated or additional data, fine-tuning model parameters, or incorporating feedback from human analysts to refine detection rules.

Deployment:

A network security architecture may be used to monitor and defend against possible security threats in real-time using the trained RNN-based intrusion detection system. To strengthen the entire network security posture, it can function alone or in concert with other security tools and methods.

Implementation Methodology of the Project

The project is carried out using a modular methodology. Every module is tested after being coded in accordance with the specifications, and this procedure is repeated until every module has been fully implemented.

Algorithms Used

RNN

An artificial neural network (ANN) with a node-to-node connectivity that mimics human brain neurons is called an RNN. Neural Network (NN) connections function similarly to a biological brain's 9 synapses. Neural network connections, like synapses, have the ability to send signals to neighbouring neurons or nodes. After processing the signal, the artificial neuron sends it on to the other neurons or nodes in its network. Weights are usually added to neurons and synapses to modify the learning process. As the signal moves from the input layers to the output layers, its strength can be altered by varying the weight. Between the input and output layers of an ANN is one hidden layer.

The reason an RNN differs is that, as Fig. 3.2 illustrates, RNNs have at least three hidden layers. Input units, output units, and hidden units make up the basic architecture of RNNs. The hidden units calculate everything by adjusting the weights and generate the results. Apart from a one-way information flow from the input units to the hidden units, the RNN model has a directed loop that can recall past knowledge and apply it to the present output. The output of the last hidden layer serves as both the input of the hidden layer and the output of the input layer. The figure displays a simple RNN architecture with two hidden layers.

The traditional FFNNs are extended by an RNN. RNNs work well for sequence modeling, in contrast to FFNN due to their cyclic connections. As indicated by the symbols X, H, and Y, respectively, we assume an input sequence, a hidden vector sequence, and an output vector sequence. The input sequence is provided.

$$X = (x1, x2, xT).$$

The hidden vector sequence is computed by a standard RNN.
$H = (h1, h2, ..., hT)$

and output vector sequence

$Y = (y1, y2, \ldots, yT)$ with t = 1 to T as follows:

$ht = \sigma\ (Wxhxt + Whhht{-}1 + bh)$......................................(1)

$yt = Whyht + by$.......................................(2) where function

A nonlinear function is denoted by σ, a weight matrix by W, and a bias component by b. Solution shows that the output of the hidden layer at t-time steps is ht, while the output of the previous hidden layer is ht-1.

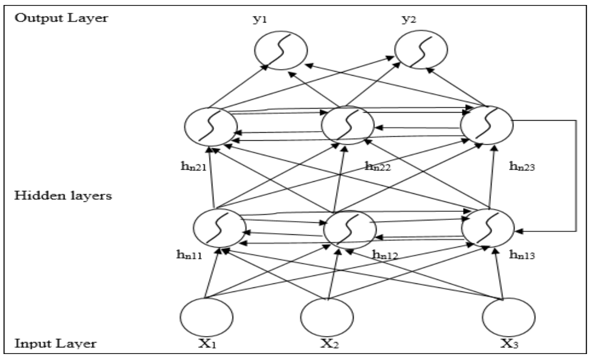


Fig.3.2. A Simple RNN

Sequential Data Representation:

Network traffic data is sequential in nature, consisting of a series of network packets or events over time. Each packet contains information such as source and destination IP addresses, port numbers, protocol type, etc. This sequential data is fed into the RNN model.

RNN Architecture:

The RNN architecture includes recurrent connections that allow information to persist over time. This enables the network to process sequential data and capture temporal dependencies between events. Common types of RNN cells used for IDS include Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which are designed to mitigate the vanishing gradient problem and capture long- term dependencies.

Training:

The RNN is trained on labeled network traffic data, where each sequence of packets is labeled as either normal or malicious. During training, the network learns to recognize patterns indicative of intrusions by adjusting its weights through backpropagation and gradient descent.

Feature Extraction:

Before feeding the data into the RNN, feature extraction may be performed to transform the raw network packet data into a suitable format. This can involve extracting relevant features such as packet size, protocol type, payload content, etc., which are then represented as input vectors for the RNN.

Detection:

Once trained, the RNN can be used to detect intrusions or anomalies in real-time network traffic. As new packets arrive, the RNN analyzes the temporal patterns in the data to determine whether the observed behavior is normal or suspicious.

Evaluation and Tuning:

The performance of the RNN-based IDS is evaluated using metrics such as detection rate, false positive rate, and false negative rate. The network may be fine-tuned or retrained using additional data to improve its detection capabilities and reduce false alarms.

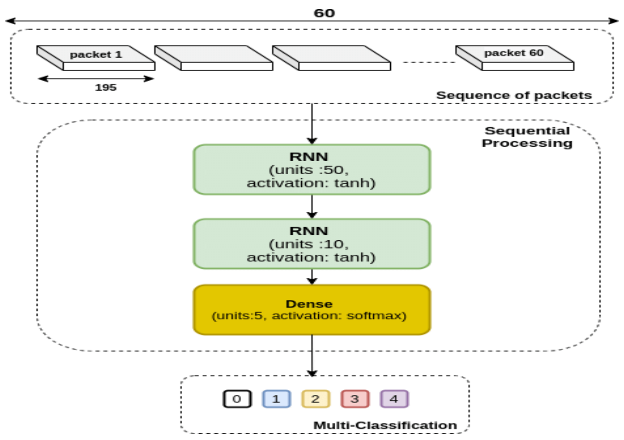

60
packet 1
packet 60
195
Sequence of packets
Sequential Processing
RNN (units :50, activation: tanh)
RNN (units :10, activation: tanh)
Dense (units:5, activation: softmax)
0 1 2 3 4
Multi-Classification

Fig.3.3. Flowchart of Methodology

IV.SYSTEM TESTING

4.1 System Testing:

The main step in software quality assurance (QA) is testing. The method is iterative. In order to test each module separately, test data is produced here. By intentionally making the system fail, system testing verifies that every component of the system works as a whole. Prior to testing, the test causes have to be organized. Next, as the testing goes on, it focuses more on looking for flaws in integrated module clusters as well as in the system as a whole. Finding mistakes is the guiding principle of testing. Testing is the stage of implementation that takes place before implementation with the goal of making sure the system functions effectively.

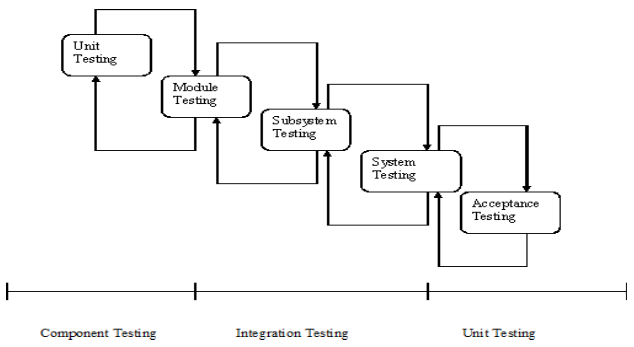


Fig. 4.1 The Testing process

The Many Formats of System Testing Are:

- Unit Testing
- Integration Testing
- Validation Testing
- System Testing

4.1.1 Unit Testing

Verification efforts for unit testing are focused on modules, the smallest unit of software architecture. We call this "Module Testing." Each module is put through a different examination. The test cases of Unit testing are given by the Table no.4.3.1. This testing is done right during the development phase. Each module is determined to be functioning successfully in terms of producing the desired results from the module during these testing procedures.

4.1.2.Integration Testing

Integration testing is a methodical approach to creating tests that identify interface errors. Every module in the project is integrated, and the complete programmer is then tested as a whole. Every error found during the integration-testing phase is fixed in preparation for the subsequent testing phases. The testing Scenarios can be seen in Table no.4.3.3.

4.1.3.Validation Testing:

To identify functional faults, that is, to determine whether or not functional features match the specifications.

4.1.4.System Testing:

The purpose of assembling modules is to function as a system once individual module testing is accomplished. After that, top-down testing which starts with upper-level module testing and moves down to lower-level module testing must be carried out to determine whether the system is operating successfully as a whole.

The system is tested as a whole after integration and unit testing are complete. System testing can be done using one of two broad approaches. The test cases of system testing are given by the Table no.4.3.2.

4.1.5 Acceptance Testing:

The system passes a final acceptance test if there are no discernible accuracy issues. This test validates that the system requires the initial objective, aim, and requirements set forth during analysis. The system is at last deemed suitable and prepared for use if it satisfies all standards.

4.2 Test Plan

A software project test plan is a written document that outlines the goals, methodology, and concentration of a software testing project. Making a test plan is a helpful technique to organize the steps required to confirm that a software product is acceptable. The finished paper will make the "Why and How" of production validation more understandable to those outside the test group. Various testing levels employ different test strategies.

4.3 Test Plans Used in Unit Testing

Every module undergoes testing to ensure that it is functioning correctly and producing all anticipated outcomes. The code properly ends condition loops to prevent infinite loops. Appropriate validations are carried out to prevent any errors relating to user data entering.

4.3.1 Unit Test Cases:

Number of Test Case	Testing Scenario	Anticipated outcome	Outcome
TC-01	Using the trained dataset by clicking on it without loading it	Error "File name for training is not defined."	Pass
TC-02	Loading trained dataset and clicking process training dataset	processed the trained dataset	Pass
TC-03	Test dataset is clicked without the test dataset being loaded.	Error "Name of the test file is not defined."	Pass
TC-04	Clicking on the process test dataset to load the test dataset	Test dataset loaded	Pass

Table.4.3.1 Unit Test cases Table

4.3.2 System Test Cases:

Number of Test Case	Testing Scenario	Anticipated outcome	Outcome
TC-01	Select the category.	A classification table was shown.	Pass
TC-02	If intrusion is detected	Show the comparative graph.	Pass
TC-03	If detect is not intrusion	Problems will arise; the comparison graph is not visible.	Pass

Table.4.3.2 System Test cases Table

4.3.3. Integration Test Cases:

Number of Test Case	Testing Scenario	Anticipated outcome	Outcome
TC-01	If the CSV file is not uploaded	Issues occur	Pass
TC-02	If the CSV file is uploaded	Data load	Pass
TC-03	If parameter> 42 or parameter<42	It won't work	Pass
TC-04	If a CSV file contains 42 lines and 42 parameters	It works	Pass

Table.4.3.3 Integration Test cases Table

V.RESULT

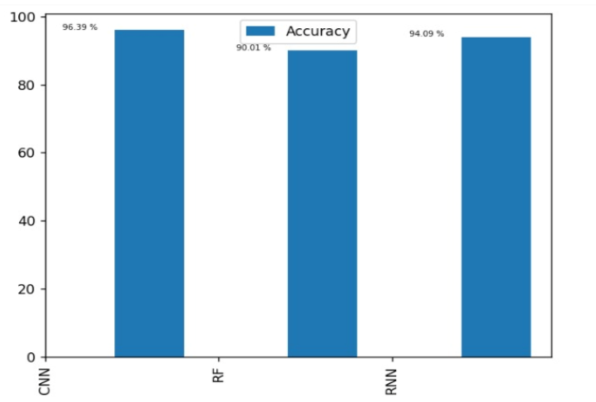


Fig.5.1. Accuracy Comparison between RNN, CNN and Random Forest

Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Random Forest methods have been compared for accuracy on a range of tasks. The results show that RNNs perform better on tasks where temporal relationships and sequential data patterns are common. Because RNNs are naturally good at capturing temporal information and learning from sequential data, they are especially useful for tasks like natural language processing, time series forecasting, and most significantly, intrusion detection will be done in every network traffic. CNNs, on the other hand, may not be as good as RNNs at capturing temporal dynamics, despite their success in picture classification tasks and superiority in spatial feature extraction. In a similar vein, Random Forest algorithms have the advantages of robustness and scalability, although they could have trouble identifying intricate temporal patterns in sequential data.

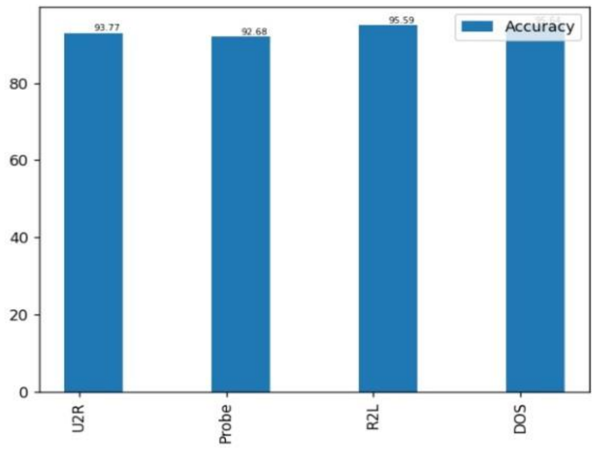


Fig.5.2. Accuracy of Detecting Types of Attacks

VI.CONCLUSION

In conclusion, the research on the Intrusion Detection System (IDS) using Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, represents a significant step forward in the realm of cybersecurity. The study aimed to address the limitations of traditional IDS by harnessing the temporal understanding and adaptability offered by RNNs. The proposed system demonstrated promising results in enhancing intrusion detection accuracy, adaptability to evolving threats, and the reduction of false positives. Through a systematic methodology involving data preprocessing, model training, and real-time monitoring, the RNN-based IDS showcased its ability to capture intricate patterns in network traffic data. The temporal dependencies modeled by LSTMs allowed the system to discern normal from anomalous behavior, enabling the identification of both known and unknown cyber threats.

The evaluation of generalization and robustness, as well as the consideration of computational efficiency and scalability, addressed crucial aspects of system feasibility and effectiveness in diverse network environments.

References

1. *Luo, J., & Bridges, S. M. (2000). Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. International Journal of Intelligent Systems, 15(8), 687-704.*
2. *Mukkamala, S., Sung, A. H., & Abraham, A. (2005). Intrusion detection using ensemble of soft computing paradigms. In Computational Intelligence in Information Assurance and Security (pp. 239-267). Springer, Berlin, Heidelberg.*
3. *Cannady, J. (1998). Artificial neural networks for misuse detection. Proceedings of the 1998 National Information Systems Security Conference (NISSC).*
4. *Amor, N. B., Benferhat, S., & Elouedi, Z. (2004). Naive Bayes vs decision trees in intrusion detection systems. Proceedings of the 2004 ACM symposium on Applied computing, 420-424.*
5. *Hofmeyr, S. A., Forrest, S., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. Journal of Computer Security,*

6(3), 151-180.

6. *Sundaram, A. (1996). An introduction to intrusion detection. ACM Crossroads, 2(4es), 3.*
7. *Lee, W., Stolfo, S. J., & Mok, K. W. (1998). Mining audit data to build intrusion detection models. Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, 66-72.*
8. *Ghosh, A. K., Schwartzbard, A., & Schatz, M. (1999). Learning program behavior profiles for intrusion detection. Proceedings of the 1st USENIX workshop on Intrusion Detection and Network Monitoring, 51-62.*
9. *Bridges, S. M., & Vaughn, R. B. (2000). Fuzzy data mining and genetic algorithms applied to intrusion detection. Proceedings of the 23rd National Information Systems Security Conference (NISSC), 16-19.*
10. *Debar, H., Becker, M., & Siboni, D. (1992). A neural network component for an intrusion detection system. Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy, 240-250.*
11. *Crosbie, M., & Spafford, E. (1995). Applying genetic programming to intrusion detection. Proceedings of the AAAI Fall Symposium on Genetic Programming1-8.,*
12. *Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. Proceedings of the 1999 IEEE Symposium on Security and Privacy, 133-145.*
13. *Lane, T., & Brodley, C. E. (1997). An application of machine learning to anomaly detection. Proceedings of the 20th National Information Systems Security Conference (NISSC), 366-380.*
14. *Anderson, J. P. (1980). Computer security threat monitoring and surveillance. Technical Report, James P. Anderson Company.*
15. *Lee, W., & Xiang, D. (2001). Information-theoretic measures for anomaly detection. Proceedings of the 2001 IEEE Symposium on Security and Privacy, 130- 143.*
16. *Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. J. (2002). A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. Applications of Data Mining in Computer Security, 77-101.*
17. *Zhang, Y., & Paxson, V. (2000). Detecting stepping stones. Proceedings of the 9th USENIX Security Symposium, 171-184.*
18. *Mahoney, M. V., & Chan, P. K. (2003). An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID), 220-237.*
19. *Kruegel, C., Mutz, D., Robertson, W., & Vigna, G. (2003). Bayesian event classification for intrusion detection. Proceedings of the 19th Annual Computer Security Applications Conference, 14-23.*
20. *Axelsson, S. (1999). Research in intrusion-detection systems: A survey. Technical Report, Department of Computer Engineering, Chalmers University of Technology.*
21. *Debar, H., Dacier, M., & Wespi, A. (2000). A revised taxonomy for intrusion-detection systems. Annals of Telecommunications, 55(7-8), 361-378.*
22. *Mukherjee, B., Heberlein, L. T., & Levitt, K. N. (1994). Network intrusion detection. IEEE Network.*
23. *Staniford, S., Hoagland, J., & McAlerney, J. (2002). Practical automated detection of stealthy portscans. Journal of Computer Security, 10(1-2), 105-136.*
24. *Denning, D. E. (1987). An intrusion-detection model. IEEE Transactions on Software Engineering, SE-13(2), 222-232.*
25. *Anderson, D., Lunt, T. F., Javitz, H., Tamaru, A., & Valdes, A. (1995). Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES). Technical Report, Computer Science Laboratory, SRI International.*
26. *Kemmerer, R. A., & Vigna, G. (2002). Intrusion detection: a brief history and overview. Computer, 35(4), 27-30.*
27. *Liao, Y., & Vemuri, V. R. (2002). Use of k-nearest neighbor classifier for intrusion detection. Computers & Security, 21(5), 439-448.*
28. *Vigna, G., & Kemmerer, R. A. (1999). NetSTAT: A network-based intrusion detection approach. Proceedings of the 14th Annual Computer Security Applications Conference, 25-34.*