IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 March 2024, accepted 3 April 2024, date of publication 16 April 2024, date of current version 1 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3389096

RESEARCH ARTICLE

A Comparative Study of Using Deep Learning Algorithms in Network Intrusion Detection

SALWA ELSAYED, KHALIL MOHAMED, AND MOHAMED ASHRAF MADKOUR
Systems and Computers Engineering Department, Faculty of Engineering, Al-Azhar University, Nasr City, Cairo 11765, Egypt
Corresponding author: Salwa Elsayed (salwa.elsayed.stu.4@azhar.edu.eg)

ABSTRACT This study introduces a deep learning approach for network intrusion detection (NIDS), which excels in both binary and multi-classification tasks. This approach combines the strengths of six distinct deep learning algorithms: DNN, CNN, RNN, LSTM, GRU, and a Hybrid CNN-LSTM architecture. The NSL-KDD dataset, a widely recognized benchmark for intrusion detection research, was utilized for implementation and evaluation. In binary classification, the approach demonstrates exceptional capabilities, with the GRU approach outperforming others. Similarly, the DNN, LSTM, CNN, and RNN approaches exhibit robust performance, showcasing their efficacy in detecting anomalies within network data. In the multi-classification setting, the DNN approach stands out with outstanding performance. While other approaches, including RNN, CNN, LSTM, GRU, and the Hybrid CNN-LSTM approach, also maintain commendable results, the DNN approach proves to be the most effective in handling complex network patterns. This research provides valuable insights into the application of deep learning approaches using the NSL-KDD dataset for network anomaly detection, emphasizing their versatility and reliability across different classification scenarios. The findings lay the groundwork for further exploration and utilization of deep learning methodologies in enhancing network security.

INDEX TERMS Network security, anomaly detection, NIDS, deep learning algorithms, NSL-KDD dataset, binary classification, multi-classification.

I. INTRODUCTION

With the increasing reliance on computer networks for business, communication, and information sharing, network security has emerged as a critical concern [1]. The escalating number and sophistication of cyber-attacks have underscored the importance of intrusion detection systems (IDS) as essential security measures for safeguarding computer networks [2].

IDS systems play a crucial role in protecting networks from malicious activities by continuously monitoring network traffic and system logs, either independently or in conjunction with firewalls [3]. These systems utilize three primary deployment methods: host-based (HIDS), network-based (NIDS), and hybrid IDS, which combines HIDS and NIDS for comprehensive protection [4].

The associate editor coordinating the review of this manuscript and approving it for publication was Chuan Heng Foh.

Despite the prevalence of signature-based techniques in existing Network Intrusion Detection System (NIDS) solutions, their limitations necessitate a shift towards anomaly detection techniques. Signature-based systems exhibit a low false error rate but can only detect known intrusions, and there is a significant delay in incorporating newly identified intrusion attacks into the signature base [5]. Relying solely on these techniques eventually leads to ineffective and inaccurate detection.

In contrast, anomaly detection systems have the ability to detect zero-day attacks by analyzing attack behavior. However, they often suffer from a high false error rate, which translates to increased costs. Additionally, acquiring reliable training data poses challenges, and capturing the behavioral dynamics of the system requires extensive data [6].

The challenge is to develop an anomaly detection technique that can effectively address the limitations arising from the evolving landscapes of modern networks [7]. These

limitations contribute to the complexity and difficulty of distinguishing between normal and abnormal behavior, and three key factors play a role [7].

The exponential growth in network data volume, driven by the widespread adoption of the Internet of Things and cloud-based services, poses a significant challenge. Analyzing such vast amounts of data necessitates techniques that are rapid, efficient, and effective.

Achieving in-depth monitoring and granularity is crucial for enhancing the effectiveness and accuracy of NIDS analysis. The ability to attribute behavioral changes to specific network elements, such as individual users, operating system versions, or protocols, is essential.

The multitude of different protocols and the diverse nature of data traversing modern networks amplify the difficulty in establishing a reliable NIDS. This situation widens the scope for potential exploitation and zero-day attacks, further underscoring the need for an accurate detection system.

An NIDS encompasses the analysis, identification, and response to malicious activities in a computer network, with the objective of improving the accuracy of classifiers for intrusive behavior [8], [9]. Machine learning and deep learning techniques are being increasingly employed in NIDS systems to enhance their accuracy and effectiveness [8], [9]. While traditional machine learning methods such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Random Forest (RF) have demonstrated success in improving detection accuracy, the emergence of deep learning has introduced a new and promising approach to NIDS [10].

Deep learning, a subset of machine learning, utilizes artificial neural networks to capture the intricate relationships between inputs and outputs. In the context of Network Intrusion Detection Systems (NIDS), Deep learning offers notable advantages over traditional machine learning methods. in the context of NIDS. First, deep learning models can have the capability to autonomously learn complex features from raw data, eliminating the need for manual feature engineering [8]. This characteristic proves especially beneficial for NIDS applications, as they often deal with high-dimensional data. Secondly, deep learning models exhibit enhanced robustness to noise and outliers in the data, which is crucial for NIDS systems that frequently encounter noisy and incomplete data. Lastly, deep learning enables the development of novel NIDS techniques, including anomaly detection and hybrid NIDS systems [8].

While deep learning-based models demonstrate promising results, certain challenges necessitate attention. These challenges encompass the scarcity of labeled data required for training deep learning models and the interpretability of such models. The collection of labeled data for network intrusion detection is a complex undertaking that demands expertise and resources. Additionally, interpretability plays a pivotal role in comprehending the reasoning behind the detection outcomes produced by deep learning models. Nevertheless, the integration of deep learning techniques in network intrusion detection exhibits significant potential in enhancing network security [11], [12].

This paper presents a deep learning-based approach for network intrusion detection (NID) that integrates various architectures, including Deep Neural Network (DNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and a Hybrid model combining CNN and LSTM (Hybrid CNN-LSTM), to address both Binary and Multi classification tasks. The model's evaluation on the NSL-KDD dataset demonstrates its remarkable accuracy, precision, recall, and F1-score. This approach excels in capturing spatial and temporal features from network traffic data, enabling a comprehensive understanding of network behavior and detection of intricate intrusion patterns. Comparative analysis reveals its superiority over traditional machine learning methods and other deep learning-based NIDS approaches. The proposed deep learning approach for NIDS brings forth significant contributions:

Enhanced NIDS Performance: By leveraging six distinct deep learning algorithms, the approach achieves superior accuracy, precision, recall, and F1-score compared to existing NID methods.

Comprehensive Feature Extraction: The approach effectively captures spatial and temporal features from network traffic data, providing a holistic representation of network behavior.

Adaptability to Binary and Multi-Classification Tasks: The approach can be applied to both binary (normal vs. anomalous) and multi-classification (Probe, Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Normal) scenarios.

The paper is structured as follows: Section II comprehensively reviews related research in intrusion detection, focusing on the role of deep learning methods. Section III provides an overview of the present study. Section IV explores the NSL-KDD Dataset and details the data preprocessing techniques employed. Section V introduces various Deep Learning Models. Section VI presents the experimental results, including the environment, evaluation metrics, performance of tuned NIDS Models for both binary and multi-classification tasks, and a comparative study with previous approaches. Finally, Section VII discusses the conclusions drawn from the study. Additionally, two appendices are included: Appendix A details the features of the NSL-KDD Dataset and its attack categories, and Appendix B provides the architectural details of deep learning models for both binary and multi-classification.

II. LITERATURE REVIEW

The past decade has witnessed a surge in research aimed at tackling the complex challenge of network intrusion detection. Various approaches have emerged, with some studies leveraging established machine learning algorithms, while others pioneering the adoption of deep learning models.

Extensive exploration has shed light on the techniques and challenges inherent in applying traditional machine learning to intrusion detection systems. Consequently, the existing body of work relevant to our investigation in network anomaly detection can be categorized into three distinct areas:

A. STATISTICAL-BASED DETECTION

Statistical-based detection methods for NIDS have laid the groundwork for anomaly identification by establishing baselines of ''normal'' network behavior. These methods analyze traffic features like packet size, protocol distribution, and connection frequency, using statistical models like univariate and multivariate analysis to build profiles and flag deviations as potential intrusions.

PAYL (Wang and Stolfo [13]) detects network traffic anomalies by modeling normal application payloads (byte frequency & std. dev.) & comparing new data (Mahalanobis distance). Effective on DARPA IDS'99 & Columbia CS data. Kruegel et al. [14] proposed a statistical intrusion detection scheme based on Bayesian networks that significantly reduces the false alarm rate by incorporating model confidence and dependencies between models.

While statistical methods provide interpretable results and low processing overhead, their effectiveness can be limited by: (i) The static nature of their models, struggling to adapt to evolving attack patterns, and (ii) Their inability to capture complex relationships between features, potentially missing sophisticated attacks. Although these methods are still valuable for initial filtering and anomaly detection, machine learning architectures offer a more dynamic and adaptable approach to NIDS.

B. MACHINE LEARNING-BASED DETECTION

While traditional statistical methods laid the groundwork for anomaly detection, machine learning (ML) methods for network intrusion detection (NIDS) offer a significant leap forward. It uses machine learning algorithms to learn the normal behavior of network traffic and identify anomalies. The system is typically trained on a large dataset of historical network traffic data. Once the system is trained, it can be used to detect anomalies in real time. Wang et al. [15] surveys ML models (supervised, unsupervised, etc.) for network anomaly detection, highlighting challenges and potential.

Dang and Quang-Vinh. [16] XGBoost shines in intrusion detection, even with limited data, but simpler options like Naive Bayes hold untapped potential. Random Forest tops anomaly detection on key datasets (CICIDS-2017, UNSW-NB15, ICS) while Naive Bayes lags [17].

According to [18], Open issues in anomaly detection persist, but promising pathways exist. Systematic technique evaluation and standardized datasets key for future progress.

Gadal et al. [19], propose a new anomaly detection method based on K-Mean Array and SMO algorithms. It is designed to handle high traffic volumes without affecting performance. It was evaluated on the NSL-KDD dataset and achieved a high accuracy of 97.4%.

Compared to statistical methods, ML boasts advantages like adaptability to evolving threats and the ability to identify complex attack signatures. However, their effectiveness hinges on accurate and comprehensive training data, and their interpretability can be limited, leaving a gap for advanced, non-linear attack detection. This gap paves the way for the exploration of deep learning architectures in this paper, promising improved accuracy, adaptability, and the potential for anomaly detection beyond known attack patterns.

C. DEEP LEARNING-BASED DETECTION

Deep learning architectures are revolutionizing network intrusion detection (NIDS) by tackling limitations inherent in traditional methods. These powerful models, driven by artificial neural networks, automatically extract intricate features from raw network traffic, capturing complex relationships and nuances beyond the reach of statistical or rule-based approaches. This enables deep learning to detect not just known attack signatures, but also subtle anomalies and even zero-day threats before they wreak havoc. Therefore, several recent studies have explored innovative approaches for network intrusion detection using deep learning techniques. Studies like [20], Network IDS: Detects unauthorized network usage to secure systems. Attacks can be monitored on network traffic or individual hosts. Network-based IDS focuses on traffic, not host behavior. Using six features from the NSL KDD dataset, Kamili [21] achieved a 76% detection rate with a DNN model for SDN intrusion detection.

Utilizing a mutual information-based feature selection, [22] tested optimal features on an LSSVM-IDS over NSL-KDD, achieving good results (DR =98.76%, FAR =0.28%) with only 18 features. Reference [23] proposed an IDS for network anomalies using recursive feature elimination with random forest on CICIDS2017, followed by a DMLP model, achieving 91% accuracy.

Ludwig [24] built an ensemble model with AE, DBN, DNN, and ELM, achieving 97.95% DR and 14.72% FAR on the NSL-KDD dataset.

Menon and Radhika [25] focused on maximizing detection rate (DR) by introducing a combined DBN-SVM model for intrusion detection. Reference [26] demonstrate the potential of DL for accurate anomaly detection. Wang and Stolfo NSL-KDD-based DL model [13] highlights the need for real-time applicability and integration with more diverse DL techniques. Additionally, class imbalance in datasets can skew results.

Promisingly, [27] explores multiclass classification with DNN, RNN, and DBN models, achieving promising results while also comparing traditional methods like decision forests. Employing an optimization algorithm for hyperparameter selection further empowers NIDS performance.

Ieracitano et al. [28] combined statistical analysis with an autoencoder (AE) to extract optimal features from the

NSL-KDD dataset, achieving improved system performance. Similarly, [29] proposed a two-stage deep learning framework using GRU and DAE models, combined with supervised and semi-supervised anomaly detection approaches. Furthermore, [30] and [31] provide comprehensive reviews of intrusion detection research, analyzing progress, bottlenecks, and evaluation methods, while [4] and [32] specifically focus on NIDS classification using ML and DL models, comparing their accuracy and complexity. These studies showcase the potential of deep learning for intrusion detection while highlighting the need for further research and optimization.

Deep learning continues to revolutionize network intrusion detection. D-PACK [33] leverages a CNN and Autoencoder combo for early anomaly detection, exhibiting impressive results in identifying malicious traffic. Similarly, a 1-D CNN architecture proves effective in [34], splitting the UNSW-NB15 dataset by protocol for targeted analysis. DOC-IDS [35] tackles obstacles like feature engineering and labeling by adopting anomaly detection, achieving outstanding performance with an AUC of 0.996 and 0.889. Finally, a dynamic anomaly detection system using LSTM+ Attention [36] boasts a 96.2% overall accuracy and 98% recall rate for various anomaly categories, showcasing the growing sophistication of deep learning-powered defenses.

Xu et al. [37] propose a 5-layer Autoencoder for NSL-KDD data, achieving impressive accuracy (90.61%) and F1-score (92.26%) in anomaly detection. This highlights the effectiveness of autoencoders in extracting crucial features for accurate threat identification.

Reference [38] explores a two-phase deep learning method using Bi-directional LSTMs on the UNSW-NB15 dataset. Their findings reveal that BLSTMs excel at predicting flow-based data but struggle with packet-based data, suggesting the need for tailored approaches for different data types.

References [39] and [40] showcase the potential of hybrid deep learning architectures. They combine CNNs and LSTMs on NSL-KDD and TEST datasets, respectively, demonstrating improved accuracy in anomaly packet recognition. This highlights the synergy between CNNs' ability to capture spatial patterns and LSTMs' prowess in handling sequential data.

Cao et al. [41] address the issue of imbalanced datasets in intrusion detection. They propose a hybrid sampling algorithm combining ADASYN and RENN, achieving remarkable accuracy and precision on NSL-KDD, UNSW-NB15, and CIC-IDS2017 datasets. This paves the way for robust detection even when dealing with skewed data distributions.

These studies underscore the vast potential of deep learning in fortifying network security, emphasizing the continuous development and refinement of these methodologies. However, a critical gap in NIDS research is identified, characterized by a limited focus on multiclass classification—a notably more challenging task than binary classification. Additionally, another significant gap pertains to the optimization of hyperparameters, playing a pivotal role in empowering NIDS performance.

To address these research gaps, our study proposes a dual approach encompassing both binary and multiclass classification for NIDS. Furthermore, we commit to the investigation of advanced optimization algorithms for hyperparameter selection. This comprehensive effort is crucial to meticulously fine-tune deep learning models, ultimately aiming for optimal performance in the intricate task of intrusion detection. By bridging these identified gaps, our research endeavors to contribute to the holistic understanding and advancement of deep learning methodologies in the realm of network intrusion detection.

III. OVERVIEW OF THE PRESENT STUDY

The present study compares six distinct deep learning architectures, to achieve enhanced intrusion detection through both binary and multi-classification tasks. As shown in Fig. 1, the compared architectures are: DNN, CNN, RNN, LSTM, GRU, and a Hybrid CNN-LSTM.

This selection aims to stress on the individual strengths of each architecture. DNNs provide a robust framework for learning high-level patterns, while CNNs extract local dependencies within traffic sequences. RNNs, LSTM, and GRU handle temporal dynamics and long-term relationships, enabling the model to recognize evolving patterns and complex intrusion signatures. The Hybrid CNN-LSTM architecture further enhances feature extraction by combining the strengths of CNNs and LSTMs.

The effectiveness of our approach is rigorously evaluated using the widely used NSL-KDD dataset, a benchmark for NIDS research. We leverage robust performance metrics such as accuracy, precision, recall, and F1-score to assess the models' ability to distinguish between normal and anomalous network traffic, paving the way for further development and potential real-world deployment.

IV. THE NSL_KDD DATASET

This section encompasses the NSL-KDD dataset introduction and preprocessing procedures.

A. NSL-KDD INTRODUCTION

The NSL-KDD dataset is a widely used publicly available'' benchmark dataset for intrusion detection research [42]. It was refined from the KDD'99 dataset to address its shortcomings, such as redundant and duplicate records [43]. This makes the number of records in the training and testing sets more reasonable, and it also prevents the classifier from favoring more frequent records. Although the NSL-KDD dataset still has the limitations discussed in [43], it is still widely used by intrusion detection researchers. According to [4],most current research uses KDD99 and NSL-KDD as datasets for evaluating performance, with NSL-KDD accounting for the majority of research. NSL-KDD is often regarded as the most widely used latest network intrusion dataset, and it can be applied as an effective benchmark to

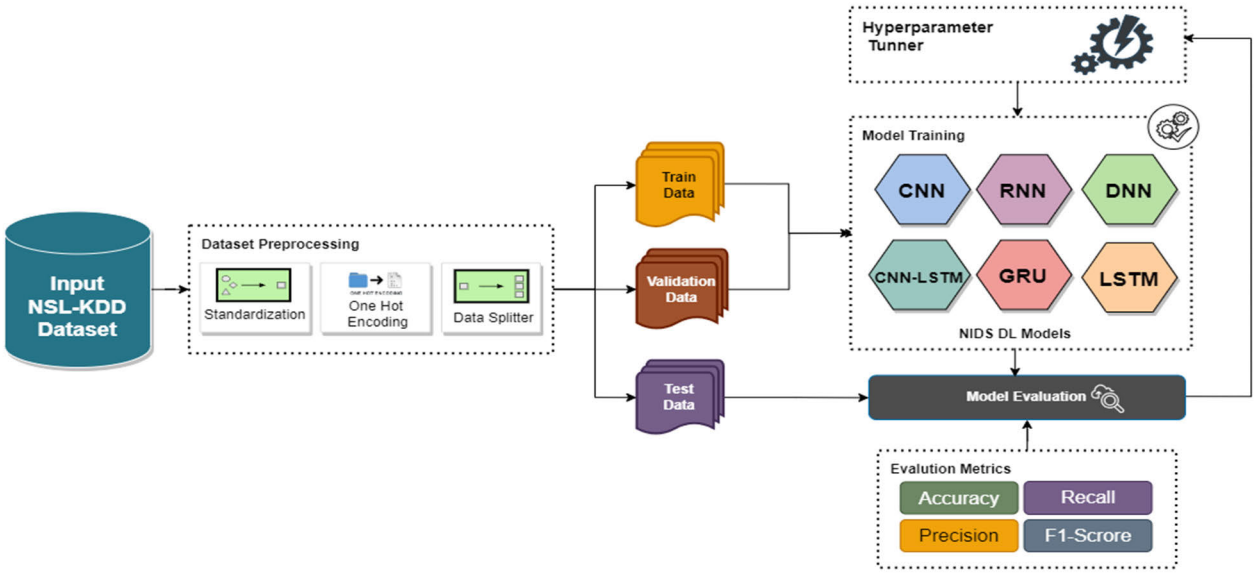


FIGURE 1. The overall NIDS development approach using six DL models.

compare different intrusion detection methods. Therefore, this paper uses the NSL-KDD dataset as the benchmark dataset.

The NSL KDD dataset consists of the KDDTrain+ dataset as the training set and the KDDTest+ and KDDTest-21 datasets as the testing set. It contains normal traffic and four different attack types: denial-of-service (DoS), root to local (R2L), user to root (U2R), and probing attacks(Probe), as shown in Table 1.

There are 41 features, including 38 numeric (e.g., int64 or float64) and 3 categorical values (e.g., object), and 1 class label for every traffic record. These features are divided into four categories: basic, content, time-based traffic, and connection-based traffic, as shown in Table 10 [Appendix A]. According to the characteristics of attacks in the dataset, there are around 20 attack types in both the training and testing sets, 2 attack types in the training set only, and 17 attack types in the testing set only that are not present in the training set. This allows the dataset to provide a more realistic theoretical basis for intrusion detection, as shown in Table 11 [Appendix A].

TABLE 1. The number of records in each subset of the NSL-KDD.

Label		KDDTrain^{+}	KDDTest^{+}	KDDTest^{-21}
Attack	DoS	45927	7458	4342
	Probe	11656	2421	2402
	R2L	995	2887	2754
	U2R	52	67	200
Normal		67343	9711	2152
Total		**125973**	**22544**	**11850**

B. NSL_KDD PREPROCESSING

To effectively utilize the NSL-KDD dataset for intrusion detection, a comprehensive data preprocessing stage is proposed. This stage encompasses three critical sub-stages:

1) STANDARDIZATION

Numerical features in the dataset, such as 'duration', 'Src_bytes', and 'Dst_bytes', are identified and standardized using the StandardScaler. Standardization transforms numerical features to have zero mean and unit variance, ensuring that all features are on a similar scale. This transformation improves the performance and convergence of various analytical and modeling techniques by preventing any single feature from dominating the analysis process. The formula for standardization is:

$$\mathbf{Z} = (\mathbf{x} - \mu)/\sigma \tag{1}$$

where Z is the standardized value, x is the original value, μ is the mean of the data.

2) ONE HOT ENCODING

As shown in Table 10 [Appendix A], the NSL-KDD dataset contains three categorical features: 'Protocol_type', 'Service', and 'Flag'. These features were transformed into numerical representations using one-hot encoding, a technique that assigns a unique binary vector to each distinct value within a categorical feature. For instance, the 'Protocol_type' feature, with its three distinct values (''tcp'', ''udp'', and ''icmp''), was expanded into three separate features. Each of these features is represented by a three-dimensional binary vector: [1,0,0], [0,1,0], and [0,0,1], respectively. Similarly, the 'Service' and 'Flag' features, with 70 and 11 unique

attributes, respectively, were expanded into corresponding sets of binary vectors.

This transformation introduced a total of 84 new features, increasing the dataset's dimensionality from 41 to 122. The resulting 122-dimensional representation comprises 38 original continuous features and 84 newly introduced features obtained through one-hot encoding.

3) DATA SPLITTING

The preprocessed dataset is split into training and test sets using the train_test_split function. The dataset is divided such that 80% of the data is allocated for training, with the remaining 20% allocated for testing. The split is performed randomly, and a fixed random state is set for reproducibility. Then, of the training data, 20% is further set aside as a validation set to monitor the model's performance on unseen data during training, helping to detect overfitting or other issues.

V. DEEP LEARNING MODELS

Deep learning algorithms have emerged as powerful tools for detecting anomalies in network traffic, offering a promising avenue for enhancing network security measures. In this paper, we employ six deep learning models (DNN, RNN, LSTM, GRU, CNN, Hybrid CNN-LSTM) for both binary and multi-classification detection. The optimal number of filters, kernel size, and hidden units were selected through a meticulous hyperparameter tuning process, where various configurations were evaluated to identify the optimal balance between model complexity and generalization performance. The following discussion explains the reasons behind exploring the use of deep learning algorithms in NIDSs.

The deep learning approach showed exceptional performance in capturing temporal and special features from the training data in many artificial intelligence applications, such as computer vision [44] and natural language understanding [45]. However, deep learning algorithms suffer from the vanishing gradient problem which is a phenomenon that occurs during the training of deep neural networks, where the gradients that are used to update the network become extremely small as they are backpropagated from the output layers to the earlier layers. Regarding network intrusion detection, the problem is to identify the anomalous behavior of the attacking entities. The analysis of typical network traffic data reveals that it contains both sequential and spatial information. Consequently, deep learning algorithms are well-suited for capturing temporal and special dependencies in network traffic patterns.

The RNN model and its variants LSTM and GRU are designed to handle sequential data including text, speech, and time series. Their internal memory mechanisms allow them to effectively capture temporal patterns in network traffic data. LSTM has a better performance than RNN because of its ability to overcome the vanishing gradient problem. GRU shares the LSTM's ability to handle long-term dependencies with greater computational efficiency. In essence, GRU reduces the computational overhead of LSTM but maintains the effective learning of long-term dependencies in network traffic data.

In addition to temporal information, deep learning models can also capture spatial features from network traffic data. The CNN model, for instance, is particularly adept at extracting local patterns and spatial relationships. By applying convolutional operations, CNNs can effectively identify spatial anomalies within the network traffic data. This ability to capture spatial features provides valuable insights into the nature of network intrusions.

The Hybrid CNN-LSTM model combines the strengths of both CNNs and LSTMs, allowing it to capture both spatial and temporal information simultaneously. This hybrid architecture enables the model to detect anomalies based on both local patterns and temporal dependencies in network traffic data, providing a comprehensive approach to network intrusion detection.

A. DNN

Deep neural networks (DNNs) are a class of artificial neural networks (ANNs) with multiple layers of neurons, enabling them to learn complex nonlinear relationships in data. In the context of network anomaly detection, DNNs can effectively extract and analyze features from raw network traffic data to identify anomalies. Their ability to model intricate patterns and relationships makes them well-suited for detecting subtle anomalies that might go unnoticed by simpler models.

B. RNN

Recurrent neural networks (RNNs) are a type of neural network designed to handle sequential data, such as network traffic data that unfolds over time. RNNs have internal memory cells that allow them to retain information from previous inputs, enabling them to capture temporal dependencies in network traffic patterns. This makes RNNs particularly effective at detecting anomalies that span multiple time steps, such as traffic spikes or unusual sequences of network events.

C. CNN

Convolutional neural networks (CNNs) are a type of neural network particularly adept at processing image and time series data. In the context of network anomaly detection, CNNs can effectively extract local patterns and sequential features from network traffic data, making them suitable for detecting anomalies based on specific data patterns. Their ability to identify local anomalies within the broader context of network traffic provides valuable insights into the nature of these anomalies.

D. LSTM

Long short-term memory (LSTM) networks are a variant of RNNs specifically designed to overcome the vanishing gradient problem, which can hinder the ability of RNNs to learn long-term dependencies. LSTMs incorporate gating mechanisms that allow them to selectively retain and update

information over extended periods, making them well-suited for detecting anomalies that occur over long-time frames.

E. GRU

Gated recurrent units (GRUs) are another variant of RNNs that share LSTM's ability to handle long-term dependencies but offer greater computational efficiency. GRUs employ a simpler gating mechanism compared to LSTMs, reducing the computational overhead while maintaining effective learning of long-range dependencies in network traffic data.

F. HYBRID CNN_LSTM

Hybrid CNN-LSTM networks combine the strengths of CNNs and LSTMs to create a powerful architecture for network anomaly detection. This hybrid approach leverages CNNs' ability to extract local patterns and LSTMs' ability to capture long-range dependencies, enabling the model to detect anomalies based on both local and temporal characteristics of network traffic data. This hybrid architecture provides a comprehensive approach to anomaly detection, capturing both spatial and temporal information from network traffic.

VI. ARCHITECTURE DETAILS AND EXPERIMENTAL RESULTS

The experimental work involved utilizing a Tensor Processing Unit (TPU) for model training on the Google Colaboratory platform, with Python 3.10.12 as the programming language. TensorFlow and Keras 2.14.0 were employed for model development, while NumPy, Pandas, and Scikit-Learn were used for data preprocessing. The computer hardware configuration included an Intel (R) Core (TM) i7-8650U CPU @ 1.90GHz (2.11 GHz), 16 GB of RAM, Intel (R) UHD Graphics 620, a 512 GB SSD, and the Windows 10 Pro 64-bit operating system (Version: 19045.3693).

The used evaluation metrics are explained in this section followed by the algorithms and the architectural details of the considered six deep learning architectures. The six NIDS models are thoroughly tuned, and their performance is presented with a comparative analysis of the obtained results against other relevant studies.

A. EVALUATION METRICS

The efficacy of Network Intrusion Detection Systems (NIDS) is evaluated using a variety of metrics, with Accuracy (AC), Precision (P), Recall (R), and F1-score (F) being among the most significant. These metrics should ideally attain high values, particularly Accuracy, as it is the foundation upon which NIDS dependability is built. Another crucial evaluation tool is the confusion matrix, which provides a detailed breakdown of various parameters, including True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

1) ACCURACY: A MEASURE OF OVERALL CORRECTNESS

Accuracy represents the proportion of network activities that are correctly classified, encompassing both intrusions and normal traffic. A high Accuracy value indicates that the NIDS is effectively distinguishing between malicious and benign network activity, minimizing the occurrence of false alarms.

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (2)$$

2) PRECISION: ENSURING THE RELEVANCE OF INTRUSION DETECTION

Precision measures the proportion of detected intrusions that are actually true intrusions. A high Precision value indicates that the NIDS is not generating a large number of false alarms, ensuring that security resources are not unnecessarily allocated to investigate non-threatening events.

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (3)$$

3) RECALL: CAPTURING THE FULL SPECTRUM OF ATTACKS

Recall measures the proportion of actual intrusions that are correctly detected. A high Recall value indicates that the NIDS is not missing a significant number of intrusions, ensuring that potential security threats are identified and addressed promptly.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \quad (4)$$

4) F1-SCORE: STRIKING A BALANCE BETWEEN PRECISION AND RECALL

F1-score is a harmonic mean of precision and recall, providing a balanced measure of the NIDS's effectiveness in both identifying and classifying intrusions. A high F1-score indicates that the NIDS is achieving a good balance between precision and recall, ensuring that it is effectively detecting and classifying intrusions without causing excessive false alarms or missing genuine intrusions.

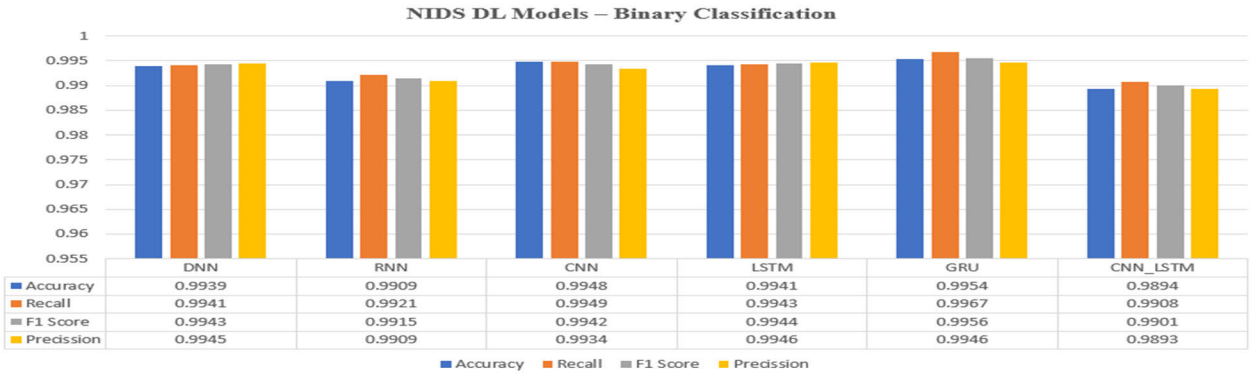
$$\text{F1} = 2 * (\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall}) \quad (5)$$

5) CONFUSION MATRIX: A DETAILED BREAKDOWN OF PERFORMANCE

The confusion matrix provides a more granular view of the NIDS's performance by categorizing the TP, FP, TN, and FN values. This breakdown can help identify areas where the NIDS is making mistakes and can guide the development of better intrusion detection rules and algorithms.

B. PERFORMANCE OF TUNED NDS MODELS

In our study, we systematically fine-tuned DNN, RNN, LSTM, GRU, CNN, and Hybrid CNN-LSTM models using Keras Tuner [46], [47] to optimize their performance beyond default settings. The best-performing models were then uniformly trained on the NSL-KDD dataset and rigorously evaluated for both binary and multi-class classification tasks using dedicated testing sets. We applied a standardized training protocol with 100 epochs and a batch size of 5000 to



	DNN	RNN	CNN	LSTM	GRU	CNN_LSTM
Accuracy	0.9939	0.9909	0.9948	0.9941	0.9954	0.9894
Recall	0.9941	0.9921	0.9949	0.9943	0.9967	0.9908
F1 Score	0.9943	0.9915	0.9942	0.9944	0.9956	0.9901
Precission	0.9945	0.9909	0.9934	0.9946	0.9946	0.9893

FIGURE 2. The results for all binary classification models.

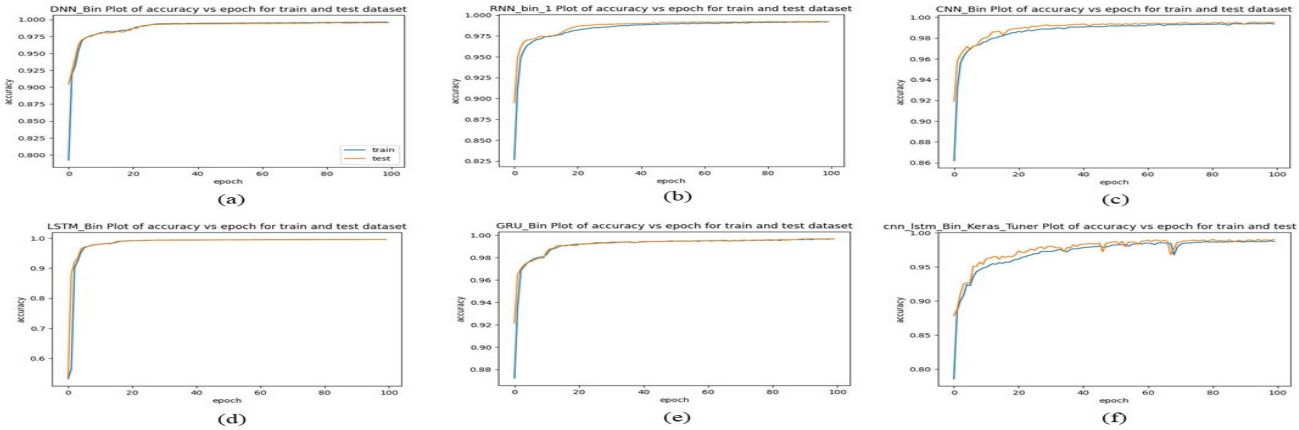


FIGURE 3. The plot of Accuracy vs. epoch for train and test dataset, for all binary classification models (a) DNN, (b) RNN, (c) CNN, (d) LSTM, (e) GRU, (f) Hybrid CNN-LSTM.

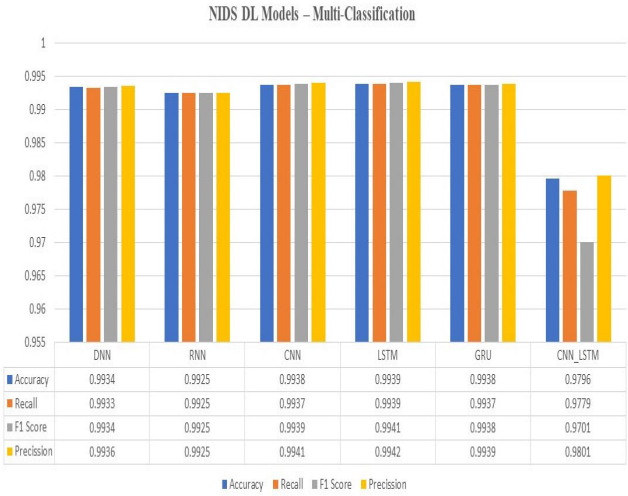
ensure consistent conditions for comprehensive performance assessment. Evaluation metrics, including accuracy, recall, F1 score, and precision, were meticulously considered to gauge the models' generalization performance on unseen testing sets. Furthermore, accuracy and loss plots were generated to visually represent the learning progress over epochs. Rigorous checks were implemented to verify the models' generalization to unseen data, thus fortifying the reliability of our anomaly detection system. Graphical analyses of accuracy and loss metrics further affirmed the absence of overfitting, contributing to the robustness of ourapproach as shown in Fig. 3, for binary and Fig. 5, for multi-classification models.

1) BINARY CLASSIFICATION

All models underwent training using binary cross-entropy loss, Adam optimizer, and a sigmoid activation function in the output layer.

a: DNN BINARY CLASSIFICATION MODEL

The DNN model architecture as shown in Fig 6 [Appendix B], is meticulously designed within a hyperparameter tuning framework, representing a sophisticated approach to optimizing neural network design. The model structure consists of an input layer, two hyperparameter-tuned hidden layers utilizing rectified linear units (ReLU) activation functions, and an output layer. The hyperparameters include the number of neurons in each hidden layer (units_1 and units_2), selected from the predefined set [8, 16, 25, 32, 50, 64, 128] as shown



	DNN	RNN	CNN	LSTM	GRU	CNN_LSTM
Accuracy	0.9934	0.9925	0.9938	0.9939	0.9938	0.9796
Recall	0.9933	0.9925	0.9937	0.9939	0.9937	0.9779
F1 Score	0.9934	0.9925	0.9939	0.9941	0.9938	0.9701
Precission	0.9936	0.9925	0.9941	0.9942	0.9939	0.9801

FIGURE 4. The results for all multi-classification models.

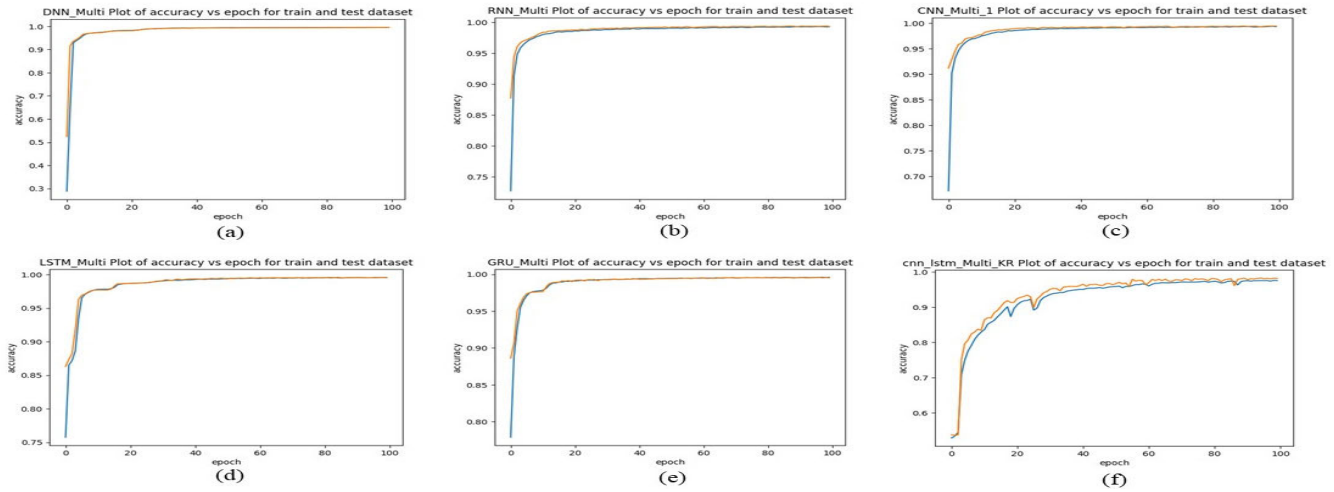


FIGURE 5. The plot of accuracy vs. epoch for train and test dataset, for all multi-classification models (a) DNN, (b) RNN, (c) CNN, (d) LSTM, (e) GRU, (f) Hybrid CNN-LSTM.

Input: Training Dataset X_train ($X_i \mid X_i \in \mathbb{R}^{41}$,i = 1,2,.,n)
Output: Categorical Y_train (member of {normal, anomaly})
Validation: Testing Dataset X_test {$X_k \mid X_k \in \mathbb{R}^{41}$,k=1,2,.,m}

```
# Step 1: Preprocessing
change_label(dataset)
standardization(dataset, numeric_col)
One-hot encode categorical features ("protocol_type", "service",
"flag")
train_test_split(dataset_bin, test_size=0.20, random_state=42)

# Step 2: Build the DNN model with Keras Tuner
def build_model_DNN(hp):
   model = Sequential()
   model.add(Dense(Dense(units=hp.Choice('units_1', [8, 16, 25,
32, 50, 64, 128]) , input_dim= X_train.shape[1],
activation='relu'))
   model.add(Dense(units= hp.Choice('units_2', [8, 16, 25, 32, 50,
64, 128]), activation='relu'))
   model.add(Dense(units=1, activation='sigmoid'))
   model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
   return model

tuner = KerasTuner.RandomSearch(build_model_DNN,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild DNN with best-hp and Train it
dnn_model_with_tuner =
build_model_DNN(best_hyperparameters)
 dnn_model_with_tuner.fit(X_train, Y_train, epochs=100,
batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the DNN model on the testing dataset
dnn_model_with_tuner.evaluate(X_test_bin, Y_test_bin)
```

LISTING 1. DNN algorithm for binary classification.

in Listing 1 for a DNN pseudo-implementation. Through an exhaustive tuning process, the optimal hyperparameter values of 32 for both units_1 and units_2 are identified.

Subsequently, the best model, manually instantiated with the previously determined optimal hyperparameter values, is systematically trained and evaluated. This refined model achieves a remarkable accuracy of 99.39%, along with high recall (0.994), F1 score (0.9943), and precision score (0.9945).

TABLE 2. The confusion matrix for all binary classification models.

DL Model	Confusion Matrix-Parameters			
	TN	TP	FN	FP
DNN	11699	13342	80	74
RNN	11651	13316	106	122
CNN	11684	13354	68	89
LSTM	11701	13345	77	72
GRU	11700	13378	44	73
Hybrid CNN_LSTM	11630	13299	123	143

The confusion matrix shown in Table 2, further demonstrates the model's robustness, indicating minimal misclassifications with only 74 false positives and 80 false negatives. These outcomes underscore the effectiveness of the hyperparameter-tuned neural network architecture in capturing intricate patterns, resulting in an exceptionally accurate and reliable model for binary classification tasks.

b: RNN BINARY CLASSIFICATION MODEL

The RNN model architecture as shown in Fig 7 [Appendix B], is meticulously designed within a hyperparameter tuning framework, with a primary focus on identifying optimal configurations for key parameters. The model structure consists of three SimpleRNN layers, each featuring a variable number of units and a strategically placed dropout layer to mitigate overfitting and preserve temporal dependencies. Notably, the first two SimpleRNN layers are configured to return sequences, allowing the model to capture temporal information inherent in network behavior. The output from

these sequential layers is then flattened and passed through a dense layer with a tunable number of units. The final layer of the model utilizes a sigmoid activation function, aligning seamlessly with the binary classification task.

The hyperparameter tuning process employs a random search strategy, systematically exploring diverse configurations for critical hyperparameters. These include the number of units in each SimpleRNN layer (units_1, units_2, and units_3), dropout rates (dropout_1 and dropout_2), and the number of units in the Dense layer (dense_units). The choices for units and dense_units range across [8, 16, 25, 32, 50, 64, 128], providing a comprehensive search space. Similarly, the dropout rates are selected from the set [0.1: 0.5], offering flexibility in fine-tuning the model's regularization strategy, through an exhaustive tuning process, the optimal hyperparameter values are identified as 50 for units_1 with a dropout rate of 0.3, 32 for units_2 with a dropout rate of 0.1, 8 for units_3, and 16 for dense_units, as shown in Listing 2.

```
# Step 2: Build the RNN model with Keras Tuner
def build_model_RNN(hp):
  model = Sequential()
  model.add(SimpleRNN(units= hp.Choice('units_1', [8, 16, 25,
32, 50, 64, 128]),return_sequences=True, input_shape=(1,
X_train.shape[2])))
  model.add(SimpleRNN(units= hp.Choice('units_2', [8, 16, 25,
32, 50, 64, 128]),return_sequences=True))
 model.add(Dropout(rate=hp.Choice("dropout_2", [0.1, 0.2, 0.3,
0.4, 0.5])))
 model.add(SimpleRNN(units= hp.Choice('units_3', [8, 16, 25, 32,
50, 64, 128]),return_sequences=True))
 model.add(Flatten())
 model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
 model.add(Dense(units= 1, activation='sigmoid'))
 model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
  return model
tuner = KerasTuner.RandomSearch(build_model_RNN,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild RNN with best-hp and Train it
rnn_model_with_tuner=
build_model_RNN(best_hyperparameters)
rnn_model_with_tuner.fit(X_train, Y_train, epochs=100,
batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the RNN model on the testing dataset
rnn_model_with_tuner.evaluate(X_test_bin, Y_test_bin)
```

LISTING 2. RNN algorithm for binary classification.

Subsequently, the best model, manually instantiated with the previously determined optimal hyperparameter values, underwent systematic training and evaluation. This refined model achieved an impressive accuracy of 99.095%, affirming its adeptness in discerning intricate patterns indicative of anomalies within network data. Notably, the model exhibited high recall (0.9921), F1 score (0.9915), and precision score (0.9909), indicating its robust generalization to unseen data and underscore its capability to effectively identify anomalous network behavior.

The confusion matrix shown in Table 2, further validated the model's resilience, revealing minimal misclassifications with only 122 false positives and 106 false negatives. These outcomes emphasize the model's effectiveness in capturing subtle deviations in network patterns, establishing it as a reliable tool for network anomaly detection, even in the face of dynamic and evolving network conditions.

c: CNN BINARY CLASSIFICATION MODEL

The Convolutional Neural Network (CNN) model for network anomaly detection as shown in Fig 8 [Appendix B], is meticulously designed within a hyperparameter tuning framework to optimize its performance in capturing complex patterns inherent in network data. The architecture comprises two convolutional layers, each characterized by a variable number of filters and kernel sizes, strategically applied dropout layers to mitigate overfitting, and max-pooling layers for spatial down-sampling. The choice of filters and kernel sizes, including [8, 16, 32, 64, 128] and [3], [5], [7], [9] respectively, reflects a comprehensive exploration of potential feature extraction configurations. The first convolutional layer is followed by a max-pooling layer with a pooling size of 4, enhancing the model's ability to capture salient features. Dropout regularization, with rates chosen from [0.1: 0.5], is strategically introduced after each convolutional-max-pooling block to prevent overfitting. The final layer of the model includes a flattening operation, followed by a Dense layer with a tunable number of units, chosen from [8, 16, 25, 32, 50, 64, 128]. The output layer utilizes a sigmoid activation function, aligning with the binary classification task. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 64 for filters_1, 7 for kernel_size_1 with a dropout rate of 0.1, 32 for filters_2, 7 for kernel_size_2 with a dropout rate of 0.1, and 50 for dense_units, as shown in Listing 3.

Subsequently, the best model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined model demonstrated outstanding performance with an accuracy of 99.376%, showcasing its proficiency in discerning intricate patterns indicative of anomalies within network data. Remarkably, the model exhibited exceptional recall (0.9949), F1 score (0.9942), and precision score (0.9934), underlining its robust generalization to unseen data and reinforcing its capacity to effectively identify anomalous network behavior.

The confusion matrix shown in Table 2, provided further validation of the model's resilience, revealing minimal misclassifications with only 89 false positives and 68 false negatives. These outcomes highlight the model's effectiveness in capturing subtle deviations in network patterns, establishing it as a highly reliable and accurate tool for

```
# Step 2: Build the CNN model with Keras Tuner
def build_model_CNN(hp):
   model = Sequential()
   model.add(Conv1D(filters =hp.Choice('filters_1', [8, 16, 32, 64,
128]), kernel_size =hp.Choice('kernel_size_1', [3, 5, 7, 9]),
padding="same",input_shape = (X_train.shape[1], 1),
activation='relu'))
   model.add(MaxPool1D(pool_size=(4)))
   model.add(Dropout(rate=hp.Choice("dropout_1", [0.1: 0.5])))
   model.add(Conv1D(filters =hp.Choice('filters_2', [8, 16, 32, 64,
128]), kernel_size =hp.Choice('kernel_size_2', [3, 5, 7, 9]),
padding="same", activation='relu'))
   model.add(MaxPool1D(pool_size=(4)))
   model add(Dropout(rate=hp.Choice("dropout_2", [0.1:0.5])))
   model.add(Flatten())
   model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
   model.add(Dense(units=1 ,activation='sigmoid'))
   model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

tuner = KerasTuner.RandomSearch(build_model_CNN,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild CNN with best-hp and Train it
cnn_model_with_tuner =
build_model_CNN(best_hyperparameters)
 cnn_model_with_tuner.fit(X_train, Y_train, epochs=100,
batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the CNN model on the testing dataset
cnn_model_with_tuner.evaluate(X_test_bin, Y_test_bin)
```

LISTING 3. CNN algorithm for binary classification.

network anomaly detection. Even in the face of dynamic and evolving network conditions, the CNN model consistently demonstrated its ability to discern anomalies, making it a valuable asset in safeguarding network integrity.

d: LSTM BINARY CLASSIFICATION MODEL

The LSTM model for network anomaly detection as shown in Fig 9 [Appendix B], is intricately designed within a hyperparameter tuning framework to optimize its performance in capturing intricate temporal patterns within network data. The architecture comprises three LSTM layers, each featuring a variable number of units and strategically placed dropout layers to mitigate overfitting. The first two LSTM layers are configured to return sequences, allowing the model to capture and retain temporal dependencies present in network behavior. The third LSTM layer operates without returning sequences, serving as a feature aggregator.

The model incorporates dropout regularization after each LSTM layer, with dropout rates systematically chosen from the set [0.1: 0.5]. A Flatten operation is introduced after the third LSTM layer, followed by a Dense layer with a tunable number of units, chosen from the set [8, 16, 25, 32, 50, 64, 128]. The output layer utilizes a sigmoid activation function, aligning with the binary classification task. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 50 for units_1 with a dropout rate of 0.2, 16 for units_2 with a dropout rate of 0.1, 16 for units_3 and 32 for dense_units, as shown in Listing 4.

```
# Step 2: Build the LSTM model with Keras Tuner
def build_model_LSTM(hp):
   model = Sequential()
    model.add(LSTM(units= hp.Choice('units_1', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True,input_shape = (1,
X_train.shape[2])))
   model.add(Dropout(rate=hp.Choice("dropout_1", [0.1 : 0.5])))
   model.add(LSTM(units= hp.Choice('units_2', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
   model.add(Dropout(rate=hp.Choice("dropout_2", [0.1 : 0.5])))
   model.add(LSTM(units= hp.Choice('units_3', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
   model.add(Flatten())
   model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
   model.add(Dense(units=1 ,activation='sigmoid'))
   model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

tuner = KerasTuner.RandomSearch(build_model_LSTM,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild LSTM with best-hp and Train it
lstm_model_with_tuner =
build_model_LSTM(best_hyperparameters)
lstm_model_with_tuner.fit(X_train, Y_train, epochs=100,
batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the LSTM model on the testing dataset
lstm_model_with_tuner.evaluate(X_test_bin, Y_test_bin)
```

LISTING 4. LSTM algorithm for binary classification.

Subsequently, the best-performing LSTM model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined LSTM model demonstrated exceptional performance, achieving an accuracy of 99.4086%. This outcome underscores its proficiency in discerning intricate patterns indicative of anomalies within network data. Notably, the LSTM model exhibited outstanding recall (0.9943), F1 score (0.9944), and precision score (0.9946), highlighting its robust generalization to unseen data and emphasizing its effectiveness in accurately identifying anomalous network behavior.

The confusion matrix shown in Table 2, provided further validation of the LSTM model's resilience, revealing minimal misclassifications with only 72 false positives and 77 false negatives. These outcomes underscore the LSTM model's efficacy in capturing subtle deviations in network patterns, establishing it as a highly reliable and accurate tool for network anomaly detection. Even in the face of dynamic and

evolving network conditions, the LSTM model consistently demonstrated its ability to discern anomalies, making it a valuable asset in safeguarding network integrity.

e: GRU BINARY CLASSIFICATION MODEL

The GRU model for network anomaly detection as shown in Fig 10 [Appendix B], is intricately designed within a hyperparameter tuning framework to optimize its performance in capturing complex temporal patterns within network data. The architecture consists of three GRU layers, each featuring a variable number of units and strategically placed dropout layers to mitigate overfitting. The first two GRU layers are configured to return sequences, enabling the model to capture and retain temporal dependencies present in network behavior. The third GRU layer operates without returning sequences, serving as a feature aggregator.

The model incorporates dropout regularization after each GRU layer, with dropout rates systematically chosen from the set [0.1: 0.5]. A Flatten operation is introduced after the third GRU layer, followed by a Dense layer with a tunable number of units, chosen from the set [8, 16, 25, 32, 50, 64, 128] (also applicable to units_1, units_2, units_3, and dense_units). The output layer utilizes a sigmoid activation function, aligning with the binary classification task. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 128 for units_1 with a dropout rate of 0.2, 128 for units_2 with a dropout rate of 0.4, 16 for units_3 and 25 for dense_units, as shown in Listing 5.

Subsequently, the best-performing GRU model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined GRU model demonstrated exceptional performance, achieving an accuracy of 99.54%. This outcome underscores its proficiency in discerning intricate patterns indicative of anomalies within network data. Notably, the GRU model exhibited outstanding recall (0.9967), F1 score (0.9956), and precision score (0.9946), highlighting its robust generalization to unseen data and emphasizing its effectiveness in accurately identifying anomalous network behavior.

The confusion matrix shown in Table 2, provided further validation of the GRU model's resilience, revealing minimal misclassifications with only 73 false positives and 44 false negatives. These outcomes underscore the GRU model's efficacy in capturing subtle deviations in network patterns, establishing it as a highly reliable and accurate tool for network anomaly detection. Even in the face of dynamic and evolving network conditions, the GRU model consistently demonstrated its ability to discern anomalies, making it a valuable asset in safeguarding network integrity.

f: HYBRID CNN_LSTM BINARY CLASSIFICATION MODEL

The Hybrid model for NIDS is designed as a combination of CNN and LSTM architectures as shown in Fig 11 [Appendix B], aiming to leverage the strengths of both in capturing spatial and temporal patterns within network

```
# Step 2: Build the GRU model with Keras Tuner
def build_model_GRU(hp):
   model = Sequential()
    model.add(LSTM(units= hp.Choice('units_1', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True,input_shape = (1,
X_train.shape[2])))
   model.add(Dropout(rate=hp.Choice("dropout_1", [0.1 : 0.5])))
   model.add(LSTM(units= hp.Choice('units_2', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
   model.add(Dropout(rate=hp.Choice("dropout_2", [0.1 : 0.5])))
   model.add(LSTM(units= hp.Choice('units_3', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
   model.add(Flatten())
   model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
   model.add(Dense(units=1 ,activation='sigmoid'))
   model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model


tuner = KerasTuner.RandomSearch(build_model_LSTM,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild GRU with best-hp and Train it
gru_model_with_tuner =
build_model_GRU(best_hyperparameters)
gru_model_with_tuner.fit(X_train, Y_train, epochs=100,
batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the GRU model on the testing dataset
gru_model_with_tuner.evaluate(X_test_bin, Y_test_bin)
```

LISTING 5. GRU algorithm for binary classification.

data. The model begins with a convolutional layer featuring a variable number of filters and kernel size, chosen from the sets [8, 16, 32, 64] and [3], [5], [7], [9], respectively. This layer employs the Rectified Linear Unit (ReLU) activation function and is followed by a max-pooling layer to downsample the spatial dimensions.

Subsequently, a LSTM layer is incorporated with a tunable number of units selected from [8, 16, 32, 64, 128], accompanied by a dropout layer with a dropout rate chosen from [0.1: 0.5]. The return_sequences parameter is set to False, indicating that the LSTM layer does not return sequences, facilitating the extraction of higher-level features. The model concludes with a dense layer utilizing a sigmoid activation function, aligning with the binary classification task, as shown in Listing 6. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 64 for filters, 7 for kernel size, 128 for units with a dropout rate of 0.2.

Subsequently, the hybrid CNN-LSTM model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined hybrid model demonstrated notable performance, achieving an accuracy of 98.944%. This outcome highlights its capability in discerning

```
# Step 2: Build the Hybrid CNN_LSTM model with Keras Tuner
def build_model_Hybrid(hp):
    model = Sequential()
    model.add(Conv1D(filters =hp.Choice('filters', [8, 16, 32, 64]),
kernel_size =hp.Choice('kernel_size', [3, 5, 7, 9]) ,
padding="same",input_shape = (X_train.shape[1], 1),
activation='relu'))
   model.add(MaxPool1D(pool_size =(4)))
   model.add(LSTM(units= hp.Choice('units', [8, 16, 32, 64,
128]),dropout= hp.Choice('dropout', [0.1: 0.5]),
return_sequences=False))
   model.add(Dense(units=1 ,activation='sigmoid'))
   model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

tuner = KerasTuner.RandomSearch(build_model_Hybrid,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild Hybrid CNN_LSTM with best-hp and Train it
hybrid_model_with_tuner =
build_model_Hybrid(best_hyperparameters)
hybrid _model_with_tuner.fit(X_train, Y_train, epochs=100,
batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the Hybrid CNN_LSTM model on the testing
dataset
hybrid _model_with_tuner.evaluate(X_test_bin, Y_test_bin)
```

LISTING 6. Hybrid CNN-LSTM algorithm for binary classification.

intricate patterns indicative of anomalies within network data. The hybrid model exhibited commendable recall (0.9908), F1 score (0.9901), and precision score (0.9894), showcasing its robust generalization to unseen data and emphasizing its effectiveness in accurately identifying anomalous network behavior.

The confusion matrix shown in Table 2, provided further validation of the hybrid model's resilience, revealing minimal misclassifications with only 143 false positives and 123 false negatives. These outcomes underscore the hybrid model's efficacy in capturing subtle deviations in network patterns, establishing it as a reliable and accurate tool for network anomaly detection. Even in the face of dynamic and evolving network conditions, the hybrid CNN-LSTM model consistently demonstrated its ability to discern anomalies, making it a valuable asset in safeguarding network integrity.

The results underscore the effectiveness of various neural network architectures in binary classification for detecting anomalies within network data shown in Fig. 2, The GRU model emerges as the top performer, achieving superior accuracy, recall, F1 score, and precision. Specifically, the GRU model demonstrated outstanding binary classification performance with a remarkable accuracy of 99.54%, recall of 99.67%, F1 score of 99.56%, and precision of 99.46%. Additionally, the LSTM, CNN, and DNN models showcased robust binary classification performance, exhibiting high accuracy, recall, F1 score, and precision values. The LSTM model, for instance, achieved 99.41% accuracy, 99.43% recall, 99.44% F1 score, and 99.46% precision. While the hybrid CNN-LSTM model proved effective, it exhibited a slightly lower performance in binary classification compared to other architectures. The hybrid model achieved an accuracy of 98.94%, a recall of 99.08%, an F1 score of 99.01%, and a precision of 98.93%.

In summary, these findings highlight the binary classification capabilities of these neural network architectures, with each model demonstrating distinct strengths in effectively detecting anomalies within network behavior.

2) MULTI-CLASSIFICATION

All models are trained using categorical cross-entropy loss, Adam optimizer, and a softmax activation function in the output layer.

a: DNN MULTI-CLASSIFICATION MODEL

The DNN model architecture as shown in Fig 12 [Appendix B], is meticulously designed within a hyperparameter tuning framework, representing a sophisticated approach to optimizing neural network design. The model structure consists of an input layer, two hyperparameter-tuned hidden layers utilizing rectified linear units (ReLU) activation functions, and an output layer. The hyperparameters include the number of neurons in each hidden layer (units_1 and units_2), selected from the predefined set [8, 16, 25, 32, 50, 64, 128]. Through an exhaustive tuning process, the optimal hyperparameter values were identified as 64 for units_1 and 8 for units_2, as shown in Listing 7.

Subsequently, the best model, manually instantiated with the previously determined optimal hyperparameter values, is systematically trained and evaluated. This refined model achieves a remarkable accuracy of 99.34%, along with high recall (0.993), F1 score (0.9934), and precision score (0.9936).

b: RNN MULTI-CLASSIFICATION MODEL

The RNN model architecture as shown in Fig 13 [Appendix B], is meticulously designed within a hyperparameter tuning framework, with a primary focus on identifying optimal configurations for key parameters. The model structure consists of three SimpleRNN layers, each featuring a variable number of units and a strategically placed dropout layer to mitigate overfitting and preserve temporal dependencies. Notably, the first two SimpleRNN layers are configured to return sequences, allowing the model to capture temporal information inherent in network behavior. The output from these sequential layers is then flattened and passed through a dense layer with a tunable number of units. The final layer of the model utilizes a softmax activation function, aligning seamlessly with the multi-classification task.

The hyperparameter tuning process employs a random search strategy, systematically exploring diverse configura-

```
Input: Training Dataset X_train (X_i | X_i ∈ ℝ^41 ,i = 1,2,.,n)
Output: Categorical Y_train (member of {normal, DoS, Probe, R2L, U2R})
Validation: Testing Dataset X_test  {Xk |Xk ∈ ℝ^41,k=1,2,.,m}

# Step 1: Preprocessing
change_label(dataset)
standardization(dataset, numeric_col)
One-hot encode categorical features ("protocol_type", "service", "flag")
train_test_split(dataset_multi, test_size=0.20, random_state=42)

# Step 2: Build the DNN model with Keras Tuner
def build_model_DNN(hp):
   model = Sequential()
   model.add(Dense(Dense(units=hp.Choice('units_1', [8, 16, 25, 32, 50, 64, 128]) , input_dim=X_train.shape[1], activation='relu'))
   model.add(Dense(units= hp.Choice('units_2', [8, 16, 25, 32, 50, 64, 128]), activation='relu'))
    model.add(Dense(units=5, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
   return model

tuner = KerasTuner.RandomSearch(build_model_DNN, objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild DNN with best-hp and Train it
dnn_model_with_tuner = build_model_DNN(best_hyperparameters)
dnn_model_with_tuner.fit(X_train_multi, Y_train_multi, epochs=100, batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the DNN model on the testing dataset
dnn_model_with_tuner.evaluate(X_test_multi, Y_test_multi)
```

LISTING 7. DNN algorithm multi-classification.

tions for critical hyperparameters. These include the number of units in each SimpleRNN layer (units_1, units_2, and units_3), dropout rates (dropout_1 and dropout_2), and the number of units in the Dense layer (dense_units). The choices for units and dense_units range across [8, 16, 25, 32, 50, 64, 128], providing a comprehensive search space. Similarly, the dropout rates are selected from the set [0.1: 0.5], offering flexibility in fine-tuning the model's regularization strategy, through an exhaustive tuning process, the optimal hyperparameter values are identified as 64 for units_1 with a dropout rate of 0.1, 32 for units_2 with a dropout rate of 0.2, 64 for units_3, and 25 for dense_units, as shown in Listing 8.

Subsequently, the best model, manually instantiated with the previously determined optimal hyperparameter values, underwent systematic training and evaluation. This refined model achieved an impressive accuracy of 99.253%, affirming its adeptness in discerning intricate patterns indicative of anomalies within network data. Notably, the model exhibited high recall (0.9924), F1 score (0.9925), and precision score (0.9925), indicating its robust generalization to unseen data and underscore its capability to effectively identify anomalous network behavior.

```
# Step 2: Build the RNN model with Keras Tuner
def build_model_RNN(hp):
   model = Sequential()
   model.add(SimpleRNN(units= hp.Choice('units_1', [8, 16, 25, 32, 50, 64, 128]),return_sequences=True,input_shape=(1, X_train.shape[2])))
   model.add(SimpleRNN(units= hp.Choice('units_2', [8, 16, 25, 32, 50, 64, 128]),return_sequences=True))
  model.add(Dropout(rate=hp.Choice("dropout_2", [0.1, 0.2, 0.3, 0.4, 0.5])))
  model.add(SimpleRNN(units= hp.Choice('units_3', [8, 16, 25, 32, 50, 64, 128]),return_sequences=True))
  model.add(Flatten())
  model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32, 50, 64, 128])))
   model.add(Dense(units=5, activation='softmax'))
   model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
   return model

tuner = KerasTuner.RandomSearch(build_model_RNN, objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild RNN with best-hp and Train it
rnn_model_with_tuner = build_model_RNN(best_hyperparameters)
rnn_model_with_tuner.fit(X_train_multi, Y_train_multi, epochs=100, batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the RNN model on the testing dataset
rnn_model_with_tuner.evaluate(X_test_multi, Y_test_multi)
```

LISTING 8. RNN algorithm multi-classification.

c: CNN MULTI-CLASSIFICATION MODEL

The Convolutional Neural Network (CNN) model for network anomaly detection as shown in Fig 14 [Appendix B], is meticulously designed within a hyperparameter tuning framework to optimize its performance in capturing complex patterns inherent in network data. The architecture comprises two convolutional layers, each characterized by a variable number of filters and kernel sizes, strategically applied dropout layers to mitigate overfitting, and max-pooling layers for spatial down-sampling. The choice of filters and kernel sizes, including [8, 16, 32, 64, 128] and [3, 5, 7, 9] respectively, reflects a comprehensive exploration of potential feature extraction configurations.

The first convolutional layer is followed by a max-pooling layer with a pooling size of 4, enhancing the model's ability to capture salient features. Dropout regularization, with rates chosen from [0.1: 0.5], is strategically introduced after each convolutional-max-pooling block to prevent overfitting. The final layer of the model includes a flattening operation, followed by a Dense layer with a tunable number of units, chosen from [8, 16, 25, 32, 50, 64, 128]. The output layer utilizes a softmax activation function, aligning with the multi-classification task. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 64 for

```
# Step 2: Build the CNN model with Keras Tuner
def build_model_CNN(hp):
   model = Sequential()
   model.add(Conv1D(filters =hp.Choice('filters_1', [8, 16, 32, 64,
128]), kernel_size =hp.Choice('kernel_size_1', [3, 5, 7, 9]),
padding="same",input_shape = (X_train.shape[1], 1),
activation='relu'))
   model.add(MaxPool1D(pool_size=(4)))
   model.add(Dropout(rate=hp.Choice("dropout_1", [0.1: 0.5])))
   model.add(Conv1D(filters =hp.Choice('filters_2', [8, 16, 32, 64,
128]), kernel_size =hp.Choice('kernel_size_2', [3, 5, 7, 9]),
padding="same", activation='relu'))
   model.add(MaxPool1D(pool_size=(4)))
   model add(Dropout(rate=hp.Choice("dropout_2", [0.1:0.5])))
   model.add(Flatten())
   model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
  model.add(Dense(units=5, activation='softmax'))
  model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
   return model

tuner = KerasTuner.RandomSearch(build_model_CNN,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]
# Step 3: Rebuild CNN with best-hp and Train it
cnn_model_with_tuner = build_model_CNN(best_hyperparameters)
cnn_model_with_tuner.fit(X_train_multi, Y_train_multi,
epochs=100, batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the CNN model on the testing dataset
cnn_model_with_tuner.evaluate(X_test_multi, Y_test_multi)
```

LISTING 9. CNN algorithm multi-classification.

filters_1, 7 for kernel_size_1 with a dropout rate of 0.1, 32 for filters_2, 7 for kernel_size_2 with a dropout rate of 0.1, and 50 for dense_units, as shown in Listing 9.

Subsequently, the best model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined model demonstrated outstanding performance with an accuracy of 99.385%, showcasing its proficiency in discerning intricate patterns indicative of anomalies within network data. Remarkably, the model exhibited exceptional recall (0.9937), F1 score (0.9939), and precision score (0.994), underlining its robust generalization to unseen data and reinforcing its capacity to effectively identify anomalous network behavior.

d: LSTM MULTI-CLASSIFICATION MODEL

The LSTM model for network anomaly detection as shown in Fig 15 [Appendix B], is intricately designed within a hyperparameter tuning framework to optimize its performance in capturing intricate temporal patterns within network data. The architecture comprises three LSTM layers, each featuring a variable number of units and strategically placed dropout layers to mitigate overfitting. The first two LSTM layers are configured to return sequences, allowing the model to capture and retain temporal dependencies present in network behavior. The third LSTM layer operates without returning sequences, serving as a feature aggregator.

The model incorporates dropout regularization after each LSTM layer, with dropout rates systematically chosen from the set [0.1: 0.5]. A Flatten operation is introduced after the third LSTM layer, followed by a Dense layer with a tunable number of units, chosen from the set [8, 16, 25, 32, 50, 64, 128]. The output layer utilizes a softmax activation function, aligning with the multi-classification task. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 64 for units_1 with a dropout rate of 0.1, 32 for units_2 with a dropout rate of 0.5, 50 for units_3 and 64 for dense_units, as shown in Listing 10. Subsequently, the best-performing LSTM model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined LSTM model demonstrated exceptional performance, achieving an accuracy of 99.39%. This outcome underscores its proficiency in discerning intricate patterns indicative of anomalies within network data. Notably, the LSTM model exhibited outstanding recall (0.9939), F1 score (0.994), and precision score (0.9941), highlighting its robust generalization to unseen data and emphasizing its effectiveness in accurately identifying anomalous network behavior.

```
# Step 2: Build the LSTM model with Keras Tuner
def build_model_LSTM(hp):
   model = Sequential()
   model.add(LSTM(units= hp.Choice('units_1', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True,input_shape = (1,
X_train.shape[2])))
   model.add(Dropout(rate=hp.Choice("dropout_1", [0.1 : 0.5])))
   model.add(LSTM(units= hp.Choice('units_2', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
   model.add(Dropout(rate=hp.Choice("dropout_2", [0.1 : 0.5])))
   model.add(LSTM(units= hp.Choice('units_3', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
   model.add(Flatten())
   model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
  model.add(Dense(units=5, activation='softmax'))
  model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
   return model

tuner = KerasTuner.RandomSearch(build_model_LSTM,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild LSTM with best-hp and Train it
lstm_model_with_tuner =
build_model_LSTM(best_hyperparameters)
lstm_model_with_tuner.fit(X_train_multi, Y_train_multi,
epochs=100, batch_size=5000, validation_split=0.2)
# Step 4: Evaluate the LSTM model on the testing dataset
lstm_model_with_tuner.evaluate(X_test_multi, Y_test_multi)
```

LISTING 10. LSTM algorithm multi-classification.

e: GRU MULTI-CLASSIFICATION MODEL

The GRU model for network anomaly detection as shown in Fig 16 [Appendix B], is intricately designed within a hyperparameter tuning framework to optimize its performance in capturing complex temporal patterns within network data. The architecture consists of three GRU layers, each featuring a variable number of units and strategically placed dropout layers to mitigate overfitting. The first two GRU layers are configured to return sequences, enabling the model to capture and retain temporal dependencies present in network behavior. The third GRU layer operates without returning sequences, serving as a feature aggregator.

The model incorporates dropout regularization after each GRU layer, with dropout rates systematically chosen from the set [0.1: 0.5]. A Flatten operation is introduced after the third GRU layer, followed by a Dense layer with a tunable number of units, chosen from the set [8, 16, 25, 32, 50, 64, 128] (also applicable to units_1, units_2, units_3, and dense_units). The output layer utilizes a softmax activation function, aligning with the multi-classification task. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 128 for units_1 with a dropout rate of 0.2, 50 for units_2 with a dropout rate of 0.1, 25 for units_3 and 16 for dense_units, as shown in Listing 11.

```
# Step 2: Build the GRU model with Keras Tuner
def build_model_GRU(hp):
  model = Sequential()
  model.add(LSTM(units= hp.Choice('units_1', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True,input_shape = (1,
X_train.shape[2])))
  model.add(Dropout(rate=hp.Choice("dropout_1", [0.1 : 0.5])))
  model.add(LSTM(units= hp.Choice('units_2', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
  model.add(Dropout(rate=hp.Choice("dropout_2", [0.1 : 0.5])))
  model.add(LSTM(units= hp.Choice('units_3', [8, 16, 25, 32, 50,
64, 128]),return_sequences=True))
  model.add(Flatten())
  model.add(Dense(units=hp.Choice("dense_units", [8, 16, 25, 32,
50, 64, 128])))
  model.add(Dense(units=5, activation='softmax'))
  model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
  return model

tuner = KerasTuner.RandomSearch(build_model_GRU,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild GRU with best-hp and Train it
gru_model_with_tuner =
build_model_GRU(best_hyperparameters)
gru_model_with_tuner.fit(X_train_multi, Y_train_multi,
epochs=100, batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the GRU model on the testing dataset
gru_model_with_tuner.evaluate(X_test_multi, Y_test_multi)
```

LISTING 11. GRU algorithm multi-classification.

Subsequently, the best-performing GRU model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined GRU model demonstrated exceptional performance, achieving an accuracy of 99.38%. This outcome underscores its proficiency in discerning intricate patterns indicative of anomalies within network data. Notably, the GRU model exhibited outstanding recall (0.9937), F1 score (0.9938), and precision score (0.9939), highlighting its robust generalization to unseen data and emphasizing its effectiveness in accurately identifying anomalous network behavior.

f: HYBRID CNN_LSTM MULTI-CLASSIFICATION MODEL

The Hybrid model for NIDS as shown in Fig 17 [Appendix B], is designed as a combination of CNN and LSTM architectures, aiming to leverage the strengths of both in capturing spatial and temporal patterns within network data. The model begins with a convolutional layer featuring a variable number of filters and kernel size, chosen from the sets [8, 16, 32, 64] and [3, 5, 7, 9], respectively. This layer employs the Rectified Linear Unit (ReLU) activation function and is followed by a max-pooling layer to downsample the spatial dimensions.

Subsequently, a LSTM layer is incorporated with a tunable number of units selected from [8, 16, 32, 64, 128], accompanied by a dropout layer with a dropout rate chosen from [0.1: 0.5]. The return_sequences parameter is set to False, indicating that the LSTM layer does not return sequences, facilitating the extraction of higher-level features. The model concludes with a dense layer utilizing a softmax activation function, aligning with the multi-classification task, as shown in Listing 12. Through an exhaustive tuning process, the optimal hyperparameter values are identified as 8 for filters, 5 for kernel size, 64 for units with a dropout rate of 0.2.

Subsequently, the hybrid CNN-LSTM model, manually instantiated with previously determined optimal hyperparameter values, underwent systematic training and evaluation for network anomaly detection. This refined hybrid model demonstrated notable performance, achieving an accuracy of 97.96%. This outcome highlights its capability in discerning intricate patterns indicative of anomalies within network data. The hybrid model exhibited commendable recall (0.9779), F1 score (0.9790), and precision score (0.9801), showcasing its robust generalization to unseen data and emphasizing its effectiveness in accurately identifying anomalous network behavior.

In the realm of multi-classification for network anomaly detection, our experimental results shed light on the performance of various neural network architectures shown in Fig. 4, The DNN model demonstrated notable accuracy, recall, F1 score, and precision, achieving scores of 99.34%, 99.33%, 99.34%, and 99.36%, respectively. Similarly, the RNN, CNN, LSTM, and GRU models exhibited strong multi-classification capabilities, with accuracy hovering around the

```
# Step 2: Build the Hybrid CNN_LSTM model with Keras Tuner
def build_model_Hybrid (hp):
  model = Sequential()
  model.add(Conv1D(filters =hp.Choice('filters', [8, 16, 32, 64]),
kernel_size =hp.Choice('kernel_size', [3, 5, 7, 9]) ,
padding="same",input_shape = (X_train.shape[1], 1),
activation='relu'))
  model.add(MaxPool1D(pool_size =(4)))
  model.add(LSTM(units= hp.Choice('units', [8, 16, 32, 64,
128]),dropout= hp.Choice('dropout', [0.1: 0.5]),
return_sequences=False))
  model.add(Dense(units=5, activation='softmax'))
  model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
  return model


tuner = KerasTuner.RandomSearch(build_model_Hybrid,
objective='val_loss, max_trials=5)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
best_hyperparameters = tuner.get_best_hyperparameters()[0]

# Step 3: Rebuild Hybrid CNN_LSTM with best-hp and Train it
hybrid _model_with_tuner = build_model_ Hybrid
(best_hyperparameters)
hybrid _model_with_tuner.fit(X_train_multi, Y_train_multi,
epochs=100, batch_size=5000, validation_split=0.2)

# Step 4: Evaluate the Hybrid CNN_LSTM model on the testing
dataset
hybrid _model_with_tuner.evaluate(X_test_multi, Y_test_multi)
```

LISTING 12. Hybrid CNN-LSTM algorithm multi-classification.

TABLE 3. DNN evaluation results for multi-class classification.

Class	Precision	Recall	F1-Score
DoS	1.00	1.00	1.00
Probe	0.99	0.98	0.99
R2L	0.92	0.87	0.89
U2R	0.33	0.09	0.14
Normal	0.99	1.00	0.99

99.25% mark and consistently high recall, F1 score, and precision values.

The LSTM model, in particular, showcased a commendable accuracy of 99.39%, coupled with a recall of 99.39%, F1 score of 99.41%, and precision of 99.42%. Meanwhile, the GRU model closely trailed, attaining an accuracy of 99.38% alongside robust recall, F1 score, and precision metrics. While these models excelled in multi-classification, the hybrid CNN-LSTM model demonstrated slightly diminished performance with an accuracy of 97.96%. Nevertheless, it maintained respectable recall (97.79%), F1 score (97.01%), and precision (98.01%) values. It is crucial to note that the hybrid model, despite a lower accuracy, still exhibited proficiency in identifying anomalies across multiple classes.

In summary, our experimental findings highlight the efficacy of diverse neural network architectures in multi-classification for network anomaly detection. Each model showcased strong performance, with the LSTM model standing out for its well-rounded capabilities. The hybrid CNN-LSTM model, although exhibiting a marginally reduced accuracy, maintained competitive performance, emphasizing the adaptability of these architectures in the challenging task of multi-class network anomaly detection.

TABLE 4. RNN evaluation results for multi-class classification.

Class	Precision	Recall	F1-Score
DoS	1.00	1.00	1.00
Probe	0.99	0.98	0.98
R2L	0.89	0.87	0.88
U2R	0.60	0.27	0.37
Normal	0.99	0.99	0.99

TABLE 5. CNN evaluation results for multi-class classification.

Class	Precision	Recall	F1-Score
DoS	1.00	1.00	1.00
Probe	0.99	0.98	0.99
R2L	0.90	0.83	0.87
U2R	0.60	0.27	0.37
Normal	0.99	1.00	0.99

TABLE 6. LSTM evaluation results for multi-class classification.

Class	Precision	Recall	F1-Score
DoS	1.00	1.00	1.00
Probe	0.99	0.99	0.99
R2L	0.87	0.87	0.87
U2R	0.50	0.18	0.27
Normal	1.00	0.99	0.99

TABLE 7. GRU evaluation results for multi-class classification.

Class	Precision	Recall	F1-Score
DoS	1.00	1.00	1.00
Probe	0.99	0.99	0.99
R2L	0.88	0.87	0.87
U2R	0.57	0.36	0.44
Normal	0.99	0.99	0.99

TABLE 8. Hybrid CNN-LSTM evaluation results for multi-class classification.

Class	Precision	Recall	F1-Score
DoS	1.00	0.99	0.99
Probe	0.95	0.96	0.96
R2L	0.60	0.60	0.60
U2R	0.40	0.16	0.25
Normal	0.97	0.99	0.98

As illustrated in Tables 3–8, this study extensively evaluated diverse deep learning algorithms for multi-class classification, specifically focusing on the identification of various attack types, including DoS, Probe, R2L, U2R, and Normal traffic. Remarkably, all algorithms demonstrated near-perfect precision, recall, and F1-scores for DoS, Probe,

TABLE 9. Hybrid CNN-LSTM evaluation results for multi-class classification.

Ref.	Method	Classification Type	Dataset	Accuracy
[48]	DNN	Multi	CSE-CIC-IDS2018	98.83%
	CNN			98.83%
	RNN			98.80%
	LSTM			98.83%
	CNN + RNN			98.84%
	CNN + LSTM			98.84%
	DNN	Binary	CSE-CIC-IDS2018	98.83%
	CNN			98.82%
	RNN			98.82%
	LSTM			98.83%
	CNN + RNN			98.84%
	CNN + LSTM			98.85%
[39]	With Training and Testing	Not Mentioned	NSL-KDD	
	CNN			82.4 %
	LSTM			82.7 %
	RNN			83.2 %
	CNN-LSTM			86.5 %
	With 10-fold cross validation	Not Mentioned	NSL-KDD	
	CNN			85.74%
	LSTM			86.72%
	RNN			85.22%
	CNN-LSTM			91.02%
[49]	CNN	Binary	NSL-KDD	98.63 %
	DNN			98.53 %
	RNN			98.13 %
	LSTM			98.04 %
	GRU			97.78 %
Our Method	**DNN**	**Binary**	**NSL-KDD**	**99.39%**
	RNN			**99.09%**
	CNN			**99.38%**
	LSTM			**99.41%**
	GRU			**99.54%**
	CNN-LSTM			**98.94%**
	DNN	**Multi**	**NSL-KDD**	**99.34%**
	RNN			**99.25%**
	CNN			**99.38%**
	LSTM			**99.39%**
	GRU			**99.38%**
	CNN-LSTM			**97.96%**

and normal traffic. Notably, DNN, RNN, and CNN emerged as particularly robust performers in R2L detection. Despite these successes, U2R detection remained a challenge across all models, highlighting the need for further research in this area. Exploring alternative architectures, feature engineering, and data balancing strategies could potentially enhance overall performance and improve U2R detection.

3) COMPARATIVE STUDY

In this section, we delve into a comparative analysis of our proposed approach and its results against other relevant studies, assessing their performance across diverse datasets. In [48], which explores the efficacy of DNN, CNN, RNN, LSTM, CNN + RNN, and CNN + LSTM on the CSE-CIC-IDS2018 dataset, demonstrating high accuracies in both multi-class (98.80% to 98.84%) and binary (98.82% to 98.85%) classifications. For the NSL-KDD dataset, in [39] compare CNN, LSTM, and RNN with hybrid CNN-LSTM, reporting a 10-fold cross-validation approach with higher accuracies than compared to the training and testing from 82.4% to 91.02% with a 10-fold cross-validation approach. Finally, [49] introduces CNN, DNN, RNN, LSTM, and GRU models for binary classification on NSL-KDD, achieving accuracies between 97.78% and 98.63%.

Compared with previous results and methods, our proposed approach stands out with exceptional accuracies (97.96% to 99.54%) for both binary and multi-class classifications on NSL-KDD, highlighting its robust performance in detecting network anomalies. The results underscore the importance of model selection tailored to specific datasets,

with our approach showcasing remarkable discriminatory power in practical network anomaly detection scenarios. We envision future optimization through hyperparameter tuning and resource efficiency to pave the way for real-world application and robust network anomaly detection. A comparison of the studies is presented in Table 9.

VII. CONCLUSION

In conclusion, this research offers a thorough comparative investigation of various established deep learning models for network anomaly detection, leveraging the widely recognized NSL-KDD dataset as a benchmark. Although the paper does not propose a new algorithm, yet it extensively compares the performance of six different existing models (DNN, RNN, LSTM, GRU, CNN, and Hybrid CNN-LSTM) on binary and multi-class classification tasks highlighting the strengths and limitations of different architectures. The paper identified GRU as the best model and to obtain best possible performance it fine tunes all hyperparameters by careful optimization.

Specifically, the GRU model demonstrates exceptional performance in binary classification, excelling in identifying normal and anomalous network behavior. Conversely, the DNN model proves most effective in navigating the intricacies of complex network patterns within multi-class classification scenarios. A notable aspect of this research is the collaborative strength derived from a diverse ensemble of deep learning algorithms. By considering models such as GRU and DNN, each excelling in specific scenarios, we emphasize the importance of a multifaceted approach to network security. This collaborative paradigm not only enhances the accuracy of anomaly detection but also underscores the adaptability required to address the evolving nature of cybersecurity threats.

The findings presented in this research serve as a guiding beacon for future endeavors in network security research. The exceptional performance of deep learning models, as demonstrated on the NSL-KDD dataset, encourages further exploration and refinement of methodologies. Future research avenues may include studying the performance of recent deep learning models such as the Transformer model [50] for network intrusion detection, optimizing model hyperparameters, exploring additional datasets, and adapting these approaches to real-world network environments.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

REFERENCES

[1] A. S. Eesa, Z. Orman, and A. M. A. Brifcani, "A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems," *Exp. Syst. Appl.*, vol. 42, no. 5, pp. 2670–2679, Apr. 2015, doi: 10.1016/j.eswa.2014.11.009.

[2] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," *Int. J. Secur. Appl.*, vol. 9, no. 5, pp. 205–216, May 2015, doi: 10.14257/ijsia.2015.9.5.21.

[3] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, Dec. 2019, doi: 10.1186/s42400-019-0038-7.

[4] M. S. Habeeb and T. R. Babu, "Network intrusion detection system: A survey on artificial intelligence-based techniques," *Exp. Syst.*, vol. 39, no. 9, Nov. 2022, Art. no. e13066, doi: 10.1111/exsy.13066.

[5] M. A. Salama, H. F. Eid, R. A. Ramadan, A. Darwish, and A. E. Hassanien, "Hybrid intelligent intrusion detection scheme," in *Soft Computing in Industrial Applications*. Berlin, Germany: Springer, 2011.

[6] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Comput. Netw.*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007, doi: 10.1016/j.comnet.2007.02.001.

[7] T. Nguyen, V. Dinh, N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. (2018). *A Deep Learning Approach to Network Intrusion Detection*. [Online]. Available: http://researchonline.ljmu.ac.uk/

[8] E. Akleman, "Deep learning," *Computer*, vol. 53, no. 9, p. 17, Sep. 2020, doi: 10.1109/MC.2020.3004171.

[9] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI Int. Conf. Bio-Inspired Inf. Commun. Technol.*, 2016, pp. 21–26, doi: 10.4108/eai.3-12-2015.2262516.

[10] P. Wang, S. Li, F. Ye, Z. Wang, and M. Zhang, "PacketCGAN: Exploratory study of class imbalance for encrypted traffic classification using CGAN," 2019, *arXiv:1911.12046*.

[11] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang, "An LSTM-based deep learning approach for classifying malicious traffic at the packet level," *Appl. Sci.*, vol. 9, no. 16, p. 3414, Aug. 2019, doi: 10.3390/app9163414.

[12] E. Mushtaq, A. Zameer, M. Umer, and A. A. Abbasi, "A two-stage intrusion detection system with auto-encoder and LSTMs," *Appl. Soft Comput.*, vol. 121, May 2022, Art. no. 108768, doi: 10.1016/j.asoc.2022.108768.

[13] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2004, pp. 203–222.

[14] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Bayesian event classification for intrusion detection," in *Proc. 19th Annu. Comput. Secur. Appl. Conf.*, 2003, pp. 14–23.

[15] S. Wang, J. F. Balarezo, S. Kandeepan, A. Al-Hourani, K. G. Chavez, and B. Rubinstein, "Machine learning in network anomaly detection: A survey," *IEEE Access*, vol. 9, pp. 152379–152396, 2021, doi: 10.1109/ACCESS.2021.3126834.

[16] Q. V. Dang, "Studying machine learning techniques for intrusion detection systems," in *Proc. Int. Conf. Future Data Security Eng.*, 2019, pp. 411–426. [Online]. Available: https://hal.science/hal-02306521

[17] N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of machine learning algorithms for anomaly detection," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services*, 2020, pp. 1–8. [Online]. Available: https://www2.le.ac.uk/offices/itservices/ithelp/services/hpc

[18] G. Fernandes, J. J. P. C. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommun. Syst.*, vol. 70, no. 3, pp. 447–489, Mar. 2019, doi: 10.1007/s11235-018-0475-8.

[19] S. Gadal, R. Mokhtar, M. Abdelhaq, R. Alsaqour, E. S. Ali, and R. Saeed, "Machine learning-based anomaly detection using K-mean array and sequential minimal optimization," *Electronics*, vol. 11, no. 14, p. 2158, Jul. 2022, doi: 10.3390/electronics11142158.

[20] B. Alsughayyir, A. M. Qamar, and R. Khan, "Developing a network attack detection system using deep learning," in *Proc. Int. Conf. Comput. Inf. Sci. (ICCIS)*, Apr. 2019, pp. 1–5.

[21] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, Oct. 2016, pp. 258–263.

[22] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 2986–2998, Oct. 2016, doi: 10.1109/TC.2016.2519914.

[23] S. Ustebay, Z. Turgut, and M. A. Aydin, "Intrusion detection system with recursive feature elimination by using random forest and deep learning classifier," in *Proc. Int. Congr. Big Data, Deep Learn. Fighting Cyber Terrorism*, 2018, pp. 71–76.

[24] S. A. Ludwig, "Intrusion detection of multiple attack classes using a deep neural net ensemble," in *Proc. IEEE Symp. Ser. Comput. Intell.*, Mar. 2017, pp. 1–7.
[25] D. M. Menon and N. Radhika. (2016). *Article a Secure Deep Belief Network Architecture for Intrusion Detection in Smart Grid Home Area Network*. [Online]. Available: www.iioab.org
[26] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *Proc. 2nd Int. Conf. Adv. Cloud Big Data*, 2014, pp. 247–252.
[27] W. Elmasry, A. Akbulut, and A. H. Zaim, "Empirical study on multiclass classification-based network intrusion detection," *Comput. Intell.*, vol. 35, no. 4, pp. 919–954, Nov. 2019, doi: 10.1111/coin.12220.
[28] C. Ieracitano, A. Adeel, F. C. Morabito, and A. Hussain, "A novel statistical analysis and autoencoder driven intelligent intrusion detection approach," *Neurocomputing*, vol. 387, pp. 51–62, Apr. 2020, doi: 10.1016/j.neucom.2019.11.016.
[29] M.-T. Kao, D.-Y. Sung, S.-J. Kao, and F.-M. Chang, "A novel two-stage deep learning structure for network flow anomaly detection," *Electronics*, vol. 11, no. 10, p. 1531, May 2022, doi: 10.3390/electronics11101531.
[30] Z. Yang, X. Liu, T. Li, D. Wu, J. Wang, Y. Zhao, and H. Han, "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," *Comput. Secur.*, vol. 116, May 2022, Art. no. 102675, doi: 10.1016/j.cose.2022.102675.
[31] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, Jan. 2021, doi: 10.1002/ett.4150.
[32] M. Aljabri, S. S. Aljameel, R. M. A. Mohammad, S. H. Almotiri, S. Mirza, F. M. Anis, M. Aboulnour, D. M. Alomari, D. H. Alhamed, and H. S. Altamimi, "Intelligent techniques for detecting network attacks: Review and research directions," *Sensors*, vol. 21, no. 21, p. 7070, Oct. 2021, doi: 10.3390/s21217070.
[33] R.-H. Hwang, M.-C. Peng, C.-W. Huang, P.-C. Lin, and V.-L. Nguyen, "An unsupervised deep learning model for early network traffic anomaly detection," *IEEE Access*, vol. 8, pp. 30387–30399, 2020, doi: 10.1109/ACCESS.2020.2973023.
[34] M. K. Hooshmand and D. Hosahalli, "Network anomaly detection using deep learning techniques," *CAAI Trans. Intell. Technol.*, vol. 7, no. 2, pp. 228–243, Jun. 2022, doi: 10.1049/cit2.12078.
[35] N. Yoshimura, H. Kuzuno, Y. Shiraishi, and M. Morii, "DOC-IDS: A deep learning-based method for feature extraction and anomaly detection in network traffic," *Sensors*, vol. 22, no. 12, p. 4405, Jun. 2022, doi: 10.3390/s22124405.
[36] P. Lin, K. Ye, and C. Z. Xu, "Dynamic network anomaly detection system by using deep learning techniques," in *Cloud Computing—CLOUD* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2019, pp. 161–176, doi: 10.1007/978-3-030-23502-4_12.
[37] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, "Improving performance of autoencoder-based network anomaly detection on NSL-KDD dataset," *IEEE Access*, vol. 9, pp. 140136–140146, 2021, doi: 10.1109/ACCESS.2021.3116612.
[38] B. Andreas, J. Dilruksha, and E. McCandless, "Flow-based and packet-based intrusion detection using BLSTM," *SMU Data Sci. Rev.*, vol. 3, no. 3, p. 8, 2020. [Online]. Available: https://scholar.smu.edu/datasciencereviewAvailableat and https://scholar.smu.edu/datasciencereview/vol3/iss3/8http://digitalrepository.smu.edu
[39] J. Suganthi, B. Nagarajan, and S. Muhtumari, "Network anomaly detection using hybrid deep learning technique," in *Advances in Parallel Computing Algorithms, Tools and Paradigms*, D. J. Hemanth et al., Eds. IOS Press, 2022, pp. 103–109, doi: 10.3233/APC220014.
[40] Y. Zeng, Z. Qi, W. Chen, Y. Huang, X. Zheng, and H. Qiu, "TEST: An end-to-end network traffic examination and identification framework based on spatio-temporal features extraction," 2019, *arXiv:1908.10271*.
[41] B. Cao, C. Li, Y. Song, Y. Qin, and C. Chen, "Network intrusion detection model based on CNN and GRU," *Appl. Sci.*, vol. 12, no. 9, p. 4184, Apr. 2022, doi: 10.3390/app12094184.
[42] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 1–6, doi: 10.1109/CISDA.2009.5356528.
[43] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000, doi: 10.1145/382912.382923.
[44] T. Ahmad, J. Wu, H. S. Alwageed, F. Khan, J. Khan, and Y. Lee, "Human activity recognition based on deep-temporal learning using convolution neural networks features and bidirectional gated recurrent unit with features selection," *IEEE Access*, vol. 11, pp. 33148–33159, 2023, doi: 10.1109/ACCESS.2023.3263155.
[45] G. Olaoye. (2024). *Deep Learning Approaches for Natural Language Processing: Advancements and Challenges*. [Online]. Available: https://www.researchgate.net/publication/378399190
[46] T. O'Malley. (2019). *KerasTuner*. [Online]. Available: https://keras.io/keras_tuner/
[47] K. Team. (2022). *Simple. Flexible. Powerful*. Keras. [Online]. Available: https://keras.io/
[48] Y.-C. Wang, Y.-C. Houng, H.-X. Chen, and S.-M. Tseng, "Network anomaly intrusion detection based on deep learning approach," *Sensors*, vol. 23, no. 4, p. 2171, Feb. 2023, doi: 10.3390/s23042171.
[49] M. R. Hadi and A. S. Mohammed, "A novel approach to network intrusion detection system using deep learning for SDN: Futuristic approach," in *Proc. AIRCC*, Jun. 2022, pp. 69–82, doi: 10.5121/csit.2022.121106.
[50] Z. Long, H. Yan, G. Shen, X. Zhang, H. He, and L. Cheng, "A transformer-based network intrusion detection approach for cloud security," *J. Cloud Comput.*, vol. 13, no. 1, p. 5, Jan. 2024, doi: 10.1186/s13677-023-00574-9.

SALWA ELSAYED received the B.Sc. degree (Hons.) in systems from Systems and Computers Engineering Department, Faculty of Engineering, Al-Azhar University, Egypt, in 2019.

She is currently an AI Researcher with over five years of experience in network engineering with Huawei Technologies. She has actively participated in competitions, such as the Global final Huawei ICT Competition, China, where she received the Gold Medal, and the Capture the Flag (CTF) competition, showcasing expertise in networking and malware analysis.

KHALIL MOHAMED received the Ph.D. degree in robotics and control engineering from Al-Azhar University, Egypt, in 2019.

He is currently an Assistant Professor with the Systems and Computers Engineering Department, Al-Azhar University. His research interests include AI, machine learning, deep learning, reinforcement learning, robotics, control theory, intelligent control systems, automotive control systems, robust control, stochastic control, motion and navigation control, traffic and transport control, predictive control, optimal control, mathematics, optimization, task assignment in multi-robot systems, and task decomposition.

MOHAMED ASHRAF MADKOUR received the B.Sc. and M.Sc. degrees in electrical engineering, in 1968 and 1974, respectively, and the Ph.D. degree in computer networking from the Electrical Engineering Department, Ain Shams University, in 1981. He is currently a Professor Emeritus with the Systems and Computers Department, Faculty of Engineering, Al-Azhar University, where he teaches courses and does research on computer networking, internetworking, and systems and data security. He has published more than 40 research papers in international and local journals and conferences. His research interests include internetworking and wireless networks, cyber security, artificial intelligence, and data science.