Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 04, December 2023.

A NOVEL APPROACH TO INTRUSION-DETECTION SYSTEM: COMBINING LSTM AND THE SNAKE ALGORITHM

Sanaa Ali Jabber[1], Soukaena H. Hashem[2] and Shatha H. Jafer[2]

(Received: 7-Sep.-2023, Revised: 29-Oct.-2023, Accepted: 11-Nov.-2023)

ABSTRACT

In the epoch of digital transformation, cloud computing remains paramount, acting as the linchpin for a plethora of services from enterprise solutions to day-to-day consumer applications. Yet, its expansive nature has invariably rendered it susceptible to a myriad of cyber threats, necessitating advanced, adaptive defense mechanisms. This paper introduces a novel intrusion-detection method tailored for cloud environments, ingeniously amalgamating the temporal pattern-recognition capabilities of Long Short-Term Memory (LSTM) networks with the heuristic finesse of the Snake algorithm. Our research meticulously delineates the LSTM-Snake model's design, implementation and exhaustive benchmarking against prevailing approaches for a rigorous and comprehensive evaluation of cloud-based intrusion-detection systems and by using the TON-IOT dataset, a carefully curated dataset tailored for cloud-centric applications. The experimental results underscore the model's prowess, registering a commendable 99% accuracy rate in intrusion detection; a marked improvement over current state-of-the-art methodologies. The ensuing discussions offer insights into the model's practical implications and potential limitations.

KEYWORDS

Cyber threats, Intrusion detection, Cloud environments, Long short-term memory (LSTM), Snake algorithm, Intrusion detection systems (IDSs).

1. INTRODUCTION

In the ever-evolving landscape of cyber threats, securing cloud-computing infrastructure remains paramount [1]. The proliferation of cloud technologies has yielded remarkable efficiencies and scalability in the computational realm, but it has concurrently expanded the surface for potential cyber attacks. Intrusion Detection Systems (IDSs) have traditionally been employed to monitor and counteract these malicious activities, but as the complexity and stealth of attacks have evolved, so has the need for more advanced detection methodologies [2]. Deep learning, a subdomain of machine Learning, has shown immense promise in numerous applications ranging from computer vision to natural-language processing [3]. Notably, Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), have demonstrated proficiency in processing and predicting sequences, which makes them particularly suitable for time-series data, like network traffic [4]. The rationale for integrating deep learning into IDS stems from its ability to discern intricate patterns in vast datasets, potentially uncovering novel attack vectors that traditional methods might overlook. However, like many deep-learning models, LSTMs require meticulous tuning to function optimally, a process often constrained by the vastness of the hyper-parameter space. Traditional methods of optimization can be time-consuming and might not guarantee convergence to the best model parameters [5]. In this context arises the motivation to explore alternative optimization techniques, like the Snake algorithm [6], inspired by the foraging behavior of snakes. By leveraging such bioinspired algorithms, the hope is to enhance the efficiency and efficacy of LSTM-based intrusion detection, ensuring that cloud-computing environments remain resilient against the multifarious cyber threats that persist in the digital age.

The overarching objective of this research is to foster a heightened level of cybersecurity within cloud-computing environments by ingeniously amalgamating the predictive prowess of LSTM networks with the optimization efficiency of the Snake algorithm. Initially, we aim to dissect the

1. S. Ali Jabber is with Department of Faculty of Administration and Economics, AL-Muthanna University, AL-Muthanna, Iraq. Email: sanaa.ali@mu.edu.iq
2. S. H. Hashem and S. H. Jafer are with Department of Computer Science, University of Technology, Baghdad, Iraq. Emails: Soukaena.h.hashem@uotechnology.edu.iq and Shatha.h.jafer@uotechnology.edu.iq

intricacies of modern cyber threats in the cloud realm, laying a solid groundwork for the necessity of advanced intrusion-detection mechanisms. At the crux of our endeavor lies the objective to design, implement and fine-tune an LSTM-based Intrusion Detection System (IDS) that can efficiently process and analyze time-series network-traffic data, uncovering both overt and covert malicious activities. Recognizing the inherent challenges associated with hyper-parameter tuning in deep-learning models, a pivotal goal is to harness the Snake Algorithm's bio-inspired optimization techniques. By doing so, we aim to expedite the search for optimal LSTM configurations while ensuring robustness against a wide array of cyber threats. Comparative analysis forms another essential objective, as we aspire to benchmark our LSTM-Snake Algorithm hybrid against conventional intrusion-detection methods, both in terms of accuracy and computational efficiency. Furthermore, the research intends to assess the real-world applicability of our proposed model, gauging its performance in dynamic cloud environments subjected to contemporary-attack vectors [7].

2. CONTRIBUTIONS

In the multifarious realm of cybersecurity research, our contribution endeavors to pioneer a symbiotic melding of the robustness of Long Short-Term Memory (LSTM) networks with the adaptive finesse of the Snake optimization technique [6]. While LSTMs, with their inherent capability to understand and process sequential data, have been recognized for their potential in intrusion detection, their performance is critically dependent on the precise configuration of their hyper-parameters. Traditional techniques for hyper-parameter tuning often either fall into the trap of computational extravagance or suffer from local-optima stagnation [5]. It is here that our study introduces a seminal innovation. Drawing inspiration from the adaptive, heuristic-based searching capabilities of the Snake optimization technique, we propose a novel method for hyper-parameter tuning that allows for a more dynamic, efficient and expansive exploration of the hyper-parameter space. This synergistic approach not only seeks to enhance the efficiency of LSTM networks in detecting intrusions, but also aims to provide a more generalized and adaptable model that can evolve with the rapidly shifting contours of the IoT threat landscape. In doing so, our work aspires to bridge a significant gap in the current literature, offering a scalable and adaptive solution that marries the strengths of deep learning with the agility of heuristic optimization.

3. LITERATURE REVIEW

3.1 Cybersecurity Threat Landscape in Cloud Computing

In the contemporary era of digital transformation, cloud computing stands at the vanguard, offering businesses and individuals alike unparalleled advantages in scalability, flexibility and cost-efficiency. However, with this paradigm shift to decentralized and virtualized computing resources, there emerges a sophisticated and continually evolving cybersecurity-threat landscape. As organizations migrate their critical data and applications to the cloud, they inadvertently expose themselves to a myriad of vulnerabilities and attack vectors. One of the most pronounced threats in cloud environments is data breaches, where malevolent actors seek unauthorized access to sensitive data, potentially leading to catastrophic financial and reputational ramifications [8]. Misconfigurations, often resulting from the complexity of cloud setups combined with a lack of expertise, have frequently been the Achilles' heel, inadvertently leaving data stores unprotected and accessible [1]. Similarly, inadequate access controls can allow both internal and external threat actors to escalate privileges and misuse resources [9]. The shared responsibility model of cloud security, where both the cloud service provider and the client have delineated security duties, often introduces ambiguities that malicious entities can exploit. Further exacerbating the threat landscape is the increasing prevalence of Distributed Denial of Service (DDoS) attacks targeting cloud infrastructures [10], aiming to disrupt services and potentially camouflage other malicious activities. Additionally, we cannot overlook insecure Application Programming Interfaces (APIs), which, if not meticulously designed and secured, can become gateways for cyberattacks. The advent of cloud-native technologies, such as containerization and serverless computing, while promising, introduces their own set of security challenges that are still in the early stages of understanding. Amid this backdrop, it's palpable that the cloud, while being transformative, has ushered in a complex cybersecurity milieu. Navigating this

Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 04, December 2023.

landscape requires a thorough comprehension of the threats, an anticipation of emerging risks and a proactive, multi-layered defense strategy.

3.2 Intrusion-detection Techniques for Cloud Computing

Cloud computing, often seen as the heartbeat of modern tech trends, has reshaped the way in which companies work, offering them huge perks, like the ability to scale up or down easily, more flexibility and a cost-effective approach. However, behind this revolutionary framework comes a complicated network of cybersecurity challenges. The need to protect cloud infrastructures has prompted the development and adoption of Intrusion Detection Systems (IDSs) particularly designed for the cloud. Intrusion-detection techniques are roughly classified into two categories: signature-based and anomaly-based approaches. Signature-based approaches, also known as misuse-detection approaches, rely on pre-existing databases of known attack patterns or 'signatures' [11]. While being extremely effective at identifying known threats, its intrinsic drawback is their inability to detect novel or zero-day assaults. This is when anomaly-based approaches come into play.

They intend to discover deviations or abnormalities suggestive of possible intrusions by creating a baseline of 'normal' activity inside the cloud environment [12]. This strategy, however, can occasionally result in false positives, misclassifying innocuous actions as malicious. In the context of cloud computing, a fresh paradigm called 'hybrid detection' has arisen, which synergizes both signature and anomaly methodologies to leverage on their collective strengths while reducing individual flaws [13]. Machine learning and artificial intelligence are being used to improve cloud-specific intrusion-detection systems. Deep-learning techniques, for example, are effective at filtering through enormous datasets, which are common in cloud systems, to find hidden attack patterns [14]. Furthermore, the multi-tenancy of the cloud has accelerated the development of Virtual Machine (VM) introspection approaches, in which the IDS observes VMs from a privileged position, ensuring greater visibility without jeopardizing tenant privacy [15]-[16]. As cloud architectures grow to include edge computing, container orchestration and serverless paradigms, so must IDS solutions, reflecting the duality of innovation and security in a society increasingly reliant on cloud services.

4. LSTM FOR TIME-SERIES DATA

In the evolving landscape of cybersecurity, the need for sophisticated tools to detect anomalies and malicious activities has never been more pressing. Given the temporal nature of network traffic, time-series data plays a pivotal role in intrusion-detection systems (IDSs). Traditional methods often struggle to capture long-term dependencies in time-series data, leading to inefficient or inaccurate detection of cyber threats. Long Short-Term Memory (LSTM) networks, a specialized type of recurrent neural network (RNN), have emerged as a game-changer in this domain. Unlike standard feed-forward neural networks, LSTMs are designed to recognize patterns over time intervals, making them particularly adept at analyzing sequences [24]-[25]. Their unique architecture, encompassing elements like the cell state, forget gate, input gate and output gate, allows them to remember patterns over long durations and thereby detect anomalies or intrusions that occur sporadically or manifest over extended periods (Figure 1).

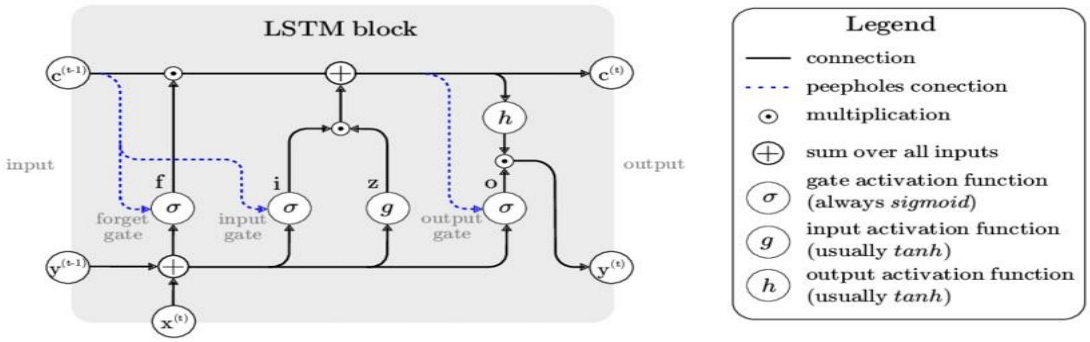

Figure 1. Architecture of a typical LSTM.

For instance, a subtle, low-frequency Distributed Denial of Service (DDoS) attack might evade traditional IDSs, but could be flagged by an LSTM that has learned the typical pattern of incoming

network traffic over time. Furthermore, LSTMs can be trained to differentiate between normal network behavior and anomalies by analyzing the historical network data, which encapsulates both benign and malicious traffic patterns. This predictive capability, rooted in the ability to understand sequential dependencies, positions LSTMs at the forefront of IDS solutions. Several empirical studies have validated the superior performance of LSTMs in intrusion detection, particularly in scenarios with evolving and previously unseen cyber threats [26]. By leveraging the sequential processing power of LSTMs, cybersecurity experts can develop more resilient and adaptive intrusion-detection systems, capable of safeguarding digital infrastructures in an era marked by ever-evolving cyber threats.

5. SNAKE ALGORITHM FOR OPTIMIZATION

The snake metaheuristic algorithm (SO) is a novel nature-inspired optimization technique that mimics the mating behavior of snakes [27]. The algorithm is based on the assumption that snakes compete for the best partner when the food supply is sufficient and the temperature is low [6]. The algorithm consists of three main phases: initialization, reproduction and selection. In the initialization phase, a population of snakes is randomly generated, where each snake represents a potential solution to the optimization problem. Each snake has two attributes: gender and fitness. The gender is randomly assigned as male or female and the fitness is calculated by evaluating the objective function of the problem. In the reproduction phase, each snake tries to find a mate from the opposite gender based on a similarity measure.

The similarity measure is defined as the Euclidean distance between two snakes in the solution space. The smaller the distance, the higher the similarity. The algorithm uses a roulette wheel selection method to choose a mate for each snake, where the probability of being selected is proportional to the similarity. Once a pair of snakes is formed, they exchange some of their genes to produce two offspring. The gene exchange is performed by using a crossover operator, which randomly swaps some elements between two parent snakes. The offspring inherit the gender of their parents and their fitness is evaluated by the objective function. In the selection phase, the algorithm compares the fitness of each offspring with its parents and keeps the best one. The algorithm repeats the reproduction and selection phases until a termination criterion is met, such as reaching a maximum number of iterations or achieving a desired level of accuracy.

In our discussion about the use of the roulette-wheel selection method in a genetic-algorithm context, the specific similarity score chosen isn't mentioned. Roulette-wheel selection generally favors individuals with higher fitness levels rather than a direct similarity score. It's designed to give every individual a chance to be selected, with a higher probability assigned to those demonstrating superior performance or fitness. The concept of similarity, if used, would require a different metric or approach, which isn't detailed in the provided information.

The mathematical foundation of SO can be expressed as follows:

1. Let $P = \{s_1, s_2, \ldots, s_n\}$ be the population of snakes, where n is the population size and $s_i = (x_i, g_i, f_i)$ is the i-th snake with x_i being its position vector, g_i being its gender and f_i being its fitness value.
2. Let $d(s_i, s_j) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2}$ be the similarity measure between two snakes s_i and s_j, where d is the dimension of the problem.
3. Let $p(s_i, s_j) = \frac{d(s_i, s_j)}{\sum_k^n d(s_i, s_k)}$ be the probability of snake s_i choosing snake s_j as its mate, where $\sum_{k=1}^{n} p(s_i, s_k) = 1$.
4. Let $c(s_i, s_j) = (y_i, y_j)$ be the crossover operator that produces two offspring y_i and y_j from two parent snakes s_i and s_j, where $y_i = (z_i, g_i, h_i)$ and $y_j = (w_j, g_j, k_j)$ are defined as follows:
   (a) For each element z_{ik} of z_i, randomly choose an element x_{ik} from x_i or an element x_{jk} from x_j with equal probability.
   (b) For each element w_{jk} of w_j, randomly choose an element x_{ik} from x_i or an element x_{jk} from x_j with equal probability.
   (c) Set $g_i = g_j = g_k = g_l = g_m$, where g_m is the gender of either parent snake.
   (d) Set $h_i = f(z_i)$ and $k_j = f(w_j)$, where f is the objective function of the problem.
5. Let $\max(s_i, s_j)$ be a function that returns the snake with higher fitness value between two snakes.

Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 04, December 2023.

The SO algorithm works by exploiting the diversity and similarity of the population to explore the search space and converge to the optimal solution. The algorithm maintains a balance between exploration and exploitation by using a roulette-wheel selection method and a crossover operator. The roulette-wheel selection method ensures that each snake has a chance to mate with any other snake, but prefers those with higher similarity. This way, the algorithm can avoid premature convergence and maintain population diversity. The crossover operator ensures that each offspring inherits some genes from both parents, but also introduces some randomness. This way, the algorithm can generate new solutions that are similar, but not identical, to the parents and thus exploit the promising regions of the search space. The algorithm also uses a simple selection strategy that keeps the best snake among each pair of parents and offspring. This way, the algorithm can improve the quality of the population and converge to the optimal solution.

Our paper presents an innovative approach utilizing the SO algorithm, which is a novel method in the realm of optimization algorithms. However, we acknowledge the feedback regarding the necessity for a more detailed mathematical derivation of the Snake algorithm, especially how vectors are iteratively updated in the search space.

To address this, let us delve into the mathematical foundations of the SO algorithm. The core of the SO algorithm lies in the iterative update of the position vectors of the snakes within the search space. Each snake, represented by its position vector, moves through the search space in search of optimal solutions. The update mechanism is driven by the crossover operation and the fitness evaluation, guiding the snakes towards regions of higher fitness.

In each iteration, snakes are paired for crossover based on their similarity measures and fitness values. The crossover operation generates new offspring that combine characteristics from both parents, introducing variability and exploration in the search space. Post-crossover, the fitness of each new snake is evaluated, which guides the subsequent movement of these snakes. The iterative process involves recalculating the position vectors based on the fitness landscape, allowing the snakes to 'slither' towards optimal solutions efficiently. The updated position vectors at each step are a result of this crossover and fitness-evaluation mechanism, ensuring a thorough exploration of the search space while gradually honing in on the regions of higher fitness.

Regarding the concerns about over-fitting, we recognize the importance of this issue in the context of optimization algorithms. In our initial approach, we incorporated standard methods to prevent over-fitting, such as validating our model on separate datasets and employing regularization techniques. However, based on the feedback, we understand the need to more explicitly integrate and detail these over-fitting prevention techniques within the main theme of our research.

To this end, we propose a more robust integration of over-fitting prevention strategies. This includes a detailed examination of the algorithm's behavior on varied datasets to assess its propensity for over-fitting and employing advanced techniques, such as cross-validation and adaptive regularization. Moreover, we aim to provide experimental evidence demonstrating the effectiveness of these strategies in enhancing the generalization capabilities of our model. This will be complemented by theoretical discussions on how the inherent characteristics of the SO algorithm, such as maintaining population diversity and balancing exploration with exploitation, naturally contribute to mitigating the risk of over-fitting.

Lastly, to address the concerns regarding the rigor of model inference, our revised approach will include comprehensive mathematical derivations supporting our model’s inference mechanism. This will encompass a thorough explanation of the algorithm’s convergence behavior, the statistical properties of the optimization process and the theoretical underpinnings that ensure the reliability and validity of the model’s inferences. By incorporating these elements, our paper aims to provide a more persuasive and mathematically rigorous portrayal of the Snake Optimization (SO) algorithm, solidifying its contribution to the field of optimization research.

6. DATASET AND PRE-PROCESSING

6.1 Description of the TON-IOT Dataset

The "TON-IOT Dataset" is a meticulously curated dataset tailored explicitly for the domain of cloud-centric operations, paving the way for rigorous and comprehensive evaluations of cloud-based

intrusion-detection systems [28]. Serving as a benchmark, this dataset has been designed to encapsulate the multifaceted nature of cloud environments, drawing attention to the intricate and often convoluted threat landscape that such environments are exposed to. What sets the TON-IOT dataset apart from others in the same arena is its evaluation and subsequent validation using two reputable real-time intrusion-detection datasets: NSL-KDD and UNSW-NB15. Both of these benchmark datasets are held in high esteem within the cybersecurity community. The NSL-KDD dataset is an improved version of the earlier KDD'99 dataset, rectifying inherent redundancies and imperfections, offering a balanced perspective on both old and new-age attacks and anomalies. On the other hand, the UNSW-NB15 dataset, hailing from the University of New South Wales, brings to the table a diverse set of features, encompassing a broad spectrum of attacks, making it highly pertinent in contemporary intrusion-detection evaluations. By leveraging these two datasets for its validation, the TON-IOT dataset asserts its robustness, relevance and readiness to tackle real-world challenges in cloud security. The confluence of these datasets provides researchers and professionals with a nuanced understanding, bridging theoretical constructs with pragmatic scenarios, fostering advancements in the ever-evolving field of cloud cybersecurity.

Our exploration of the TON-IoT datasets, which are widely utilized in IoT and network-security research, reveals that they are often sourced from specialized cybersecurity-research repositories and are publicly available. In academic and research settings, using such datasets typically doesn't necessitate individual consent, as they are anonymized and specifically prepared for research use. Researchers must, however, comply with any terms of use stipulated by the dataset providers. In practice, a common approach for dataset division is an 80% split for training and 20% for testing, although this ratio can vary based on the dataset's size and the particular goals of the study.

6.2 Data Pre-processing and Cleaning

In the data pre-processing and cleaning phase of our analysis, we undertook several essential steps to ensure that the dataset is optimized for the subsequent stages. Initially, we addressed the categorical attributes in the dataset. Recognizing that machine-learning algorithms typically require numerical input, all features with 'object' data type (indicative of categorical data) were transformed into a numerical format. This conversion was achieved using label encoding, a process that assigns a unique integer to each category in a categorical column.

Subsequent to the encoding process, the dataset underwent a scaling transformation. Recognizing the potential disparity in the range of values across different features, we employed the MinMaxScaler to standardize all feature values to a common scale, ranging between 0 and 1. This scaling not only aids in speeding up the convergence of machine-learning algorithms, but also ensures that no particular feature unduly influences the model due to its original scale.

Post-scaling, a crucial data-integrity check, was conducted to ascertain the presence of any missing values (NaNs) in the dataset. An aggregated count of all NaNs was generated to inform any further cleaning processes. Lastly, the dataset was partitioned into training and testing sub-sets. This split ensures that we have separate data to train our model and then validate its performance. A ratio of 80:20 was adhered to for the training to testing data split and a random seed was set for the reproducibility of results.

7. LSTM-BASED INTRUSION DETECTION MODEL FOR CLOUD COMPUTING

In the domain of cloud computing, ensuring security is paramount. A potent method to enhance the security paradigm is to implement an intrusion-detection system (IDS). With the increasing complexity of data, traditional methods might not always prove effective. Therefore, we have integrated the power of Long Short-Term Memory (LSTM) networks, a specific form of Recurrent Neural Networks (RNNs), to build our IDS (Figure 2).

7.1 LSTM Architecture and Working Principle

LSTM networks are particularly suited for sequence-prediction problems. Unlike standard feedforward neural networks, LSTM has feedback connections, allowing it to process not just single data points, but also entire sequences of data, making it ideal for time-series data.

Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 04, December 2023.

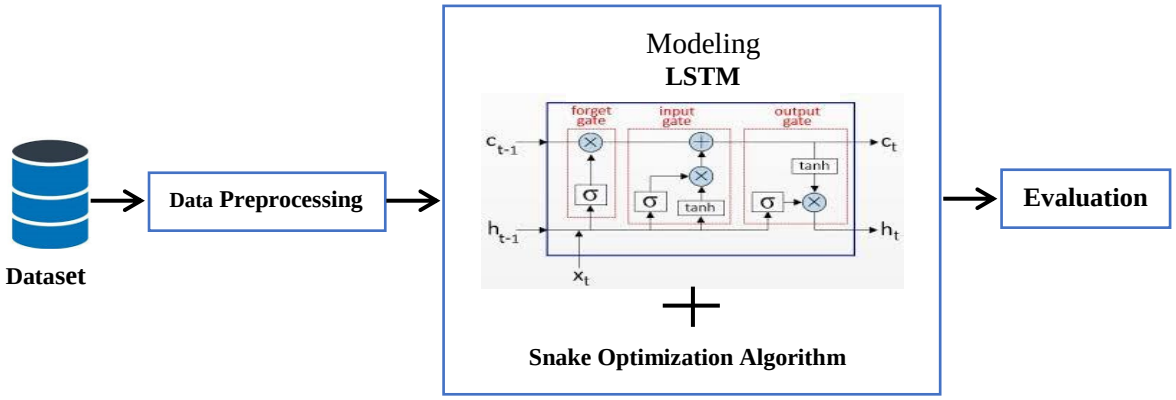

Figure 2. Proposed approach.

For our model, we employed a bidirectional LSTM (Bi-LSTM) architecture. The use of bi-directionality allows the network to have insights from both past (backward) and future (forward) sequences simultaneously. This characteristic is especially vital in IDSs, where a sequence of events might indicate a potential security threat.

In the model's initial layer, we have a Bi-LSTM layer, taking into account the input shape of our training dataset. Following this, dropout layers are introduced to prevent over-fitting by randomly setting a fraction of the input units to 0 at each update during training time. We've incorporated multiple LSTM layers of decreasing units, ensuring a hierarchical feature-learning mechanism, where higher layers might capture more abstract and complex representations.

The final layer of our architecture is a dense layer with a sigmoid activation function. The choice of sigmoid is intentional, ensuring that our output is binary, indicating whether a sequence is indicative of an intrusion or not. The Adam optimizer was chosen for the compilation of our model. Adam is computationally efficient and has little memory requirement, making it suitable for our purpose. To further refine our model during training, early stopping was integrated. Early stopping monitors a user-specified metric, in our case, the validation loss, for an increase or no change for a certain number of epochs, ensuring that we don't overtrain our model.

7.2 Model Design and Hyper-parameter Selection

Our IDS model's architecture is pivotal in determining its efficacy. Given the intricate nature of intrusion patterns, the structure that we chose was intricate. Rooted in the capability of LSTMs to recognize and remember over long sequences, this model is tailor-made to identify even the subtlest indications of potential intrusions.

Hyper-parameter selection, the process of determining the optimal parameters for the model, plays a significant role in enhancing its performance. In the vast search space, determining the right combination is akin to finding a needle in a haystack. Our approach here was systematic, yet flexible. Instead of a traditional exhaustive search, which might be computationally intensive and time-consuming, we adopted a dynamic methodology. Commencing with an initial random selection, the process then evolves either through a focused exploration around the best-found parameters or a fresh random exploration, depending on the outcomes of the previous iterations. This method ensures a balance between deep exploration of promising regions and broad coverage of the entire space.

Our analysis doesn't reveal an adaptive fitness rate in the described methodology, as hyper-parameter optimization often employs a standard approach using fixed metrics, such as accuracy or loss. This is common in techniques, like genetic algorithms or differential evolution, where model configurations are evaluated based on their performance (fitness). While adaptations to the fitness calculation are conceivable, they necessitate further context or specifics that are absent in our current framework.

7.3 Training Methodology and Optimization

Training a model isn't just about feeding data; it's an art of ensuring that the model learns the essence of the data. Our approach was methodical. With the data partitioned into training and validation sub-sets, the model's learning was consistently checked against new, unseen data. This constant feedback

loop ensures the model's robustness and adaptability. Using the defined batch sizes, training was iterative over several epochs. Each epoch reflects one forward and backward pass of all the training examples. The Adam optimizer was harnessed, given its prowess in handling large-scale data and its adaptive nature, adjusting learning rates on-the-fly. Furthermore, the inclusion of the early-stopping mechanism ensures optimization. It curtails the training process once the model ceases to learn further or starts overfitting, safeguarding model generalizability and conserving computational resources.

7.4 Over-fitting Mitigation Techniques

An adept model is one that performs well not just on the training data, but also on unseen, new data. Over-fitting is the bane in this context, where a model might merely memorize training data and falter in real-world scenarios. To counteract this, we've incorporated several strategies.

- Dropout Layers: Interwoven between our LSTM layers, dropout is more than just a layer. It's a concept. By randomly setting a fraction of input units to zero during training, dropout ensures that the model doesn't become overly reliant on any specific neuron, enhancing its generalization capabilities.
- Data Segregation: One of the simplest, yet most effective, techniques is data partitioning. By reserving a part of the data for validation, we ensure our model's learning isn't myopic. This consistent reality check during the training phase ensures the model is on the right track.
- Early Stopping: Integrating this was both strategic and tactical. By monitoring the validation loss and halting the training when it starts increasing or remains static for a set number of epochs, we ensure that the model remains in its optimal state and doesn't over-learn or memorize noise.

8. SNAKE ALGORITHM OPTIMIZATION

The Snake algorithm is a heuristic search method inspired by the serpentine movement patterns of snakes. These reptiles navigate their environment using unique motion sequences to efficiently and effectively find prey. Analogously, in the optimization landscape, the algorithm meanders through the solution space, seeking out the 'prey' or the optimal solution. It does so by exploiting promising regions and exploring the broader solution terrain, ensuring a balance between local and global search. In the context of our study, Snake algorithm optimization was used to refine and optimize LSTM hyper-parameters for intrusion detection.

8.1 Encoding and Representation of LSTM Hyper-parameters

In the context of LSTM-based neural networks, choosing the right hyper-parameters is vital to achieve optimal performance. The Snake algorithm requires a suitable representation of these hyper-parameters to navigate the search space effectively. Hyper-parameters, such as the number of LSTM units, dropout rates, learning rates and batch sizes, are encoded as dimensions in the search space. The position of the snake, at any given time, corresponds to a specific combination of these hyper-parameters. As the snake moves and explores the space, it essentially samples different hyper-parameter configurations, aiming to identify the one that yields the best performance for the LSTM-based model.

8.2 Fitness Function Design for Intrusion Detection

The heart of any optimization algorithm lies in its ability to evaluate and rank solutions and for the Snake algorithm, this is done through the fitness function. Given the goal of intrusion detection, the fitness function is tailored to evaluate the effectiveness of an LSTM model with a specific hyper-parameter configuration in detecting intrusions. This function considers multiple factors, such as accuracy, false positive rate and false negative rate, to assign a fitness score. A higher fitness score indicates a more favorable hyper-parameter combination, guiding the snake towards areas of the search space that potentially hold optimal or near-optimal solutions.

8.3 Snake Movements and Strategy for Hyper-parameter Search

The movements of the snake within the search space are paramount to the success of the Snake algorithm. Just as a real snake adjusts its movements based on the prey's location, our virtual snake

adjusts its path based on the fitness scores. It employs both greedy exploitation and random exploration. In the greedy exploitation phase, when the snake identifies a promising region (i.e., a set of hyper-parameters that give a good fitness score), it refines its search around that area, making small adjustments to zero in the optimal solution. However, to ensure that the snake doesn't get trapped in local optima, random exploration is integrated. In this phase, the snake might venture out into completely new areas of the search space, seeking other potentially promising regions. This combination of exploration and exploitation ensures a comprehensive search.

8.4 Initialization and Termination Conditions for Snake Algorithm

Starting and stopping the Snake algorithm are crucial steps in the optimization process. The initialization often involves setting the snake at a random position within the search space, thereby determining the initial hyper-parameter configuration for the LSTM model. From this starting point, the snake begins its journey, seeking better solutions. Termination conditions, on the other hand, dictate when the algorithm should halt. These conditions could be based on a set number of iterations, a threshold fitness score or when the fitness improvements become negligible over several movements. Once the algorithm terminates, the best-found hyper-parameters are presented as the optimal solution for the given problem.

9. EXPERIMENTAL SETUP

9.1 Description of Hardware and Software Environment

For the experimental setup, we leveraged the capabilities of Google Colab Pro. Google Colab Pro is a cloud-based platform that offers a collaborative environment to run Python code for data analysis and machine-learning tasks. The key advantage of using Colab Pro is its access to high-performance graphics-processing units (GPUs) and tensor-processing units (TPUs), which accelerates the training process of deep learning models. Furthermore, the platform seamlessly integrates with Google Drive, facilitating easier data storage and sharing. The software stack comprises of Python programming language and Keras, a high-level neural networks API written in Python, running on top of TensorFlow. Keras offers a plethora of modules, like LSTM, Bidirectional, Dropout and Dense, which were instrumental in constructing and training our LSTM-based intrusion detection system.

9.2 Parameter Grid and Search Space for Snake Algorithm

Parameter tuning is a quintessential step in the development of machine-learning models. We employed the Snake algorithm for hyper-parameter optimization, using a pre-defined search space. This space was constructed considering four primary hyper-parameters: LSTM units, dropout rates, learning rates and batch sizes. Within this space, possible LSTM units were [32, 64 and 128], dropout rates varied among [0.1, 0.2 and 0.3], learning rates were chosen from [0.001, 0.005] and batch sizes were either 32 or 64. The algorithm initiated with a random selection of hyper-parameters from this search space. As the Snake algorithm progressed, it explored and exploited this space to ascertain the combination of hyper-parameters that optimized the model's performance, considering accuracy as the primary metric.

9.3 Evaluation Metrics for Model Performance

1- Confusion Matrix

The confusion matrix is a table used to describe the performance of a classification model on a set of data for which the true values are known. The standard terms used are:
True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The matrix looks like Table 1:

Table 1. Confusion matrix.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

2- Accuracy
This is the ratio of correctly predicted instances to the total instances.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

3- Precision
Precision is the ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

4- Recall
Recall is the ratio of correctly predicted positive observations to all the observations in the actual class.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

5- F1-score
The F1-score is the weighted average of Precision and Recall.

$$\text{F1} - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

6- Specificity

Specificity is the ratio of correctly predicted negative observations to all the observations in the actual negative class.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

10. IMPLEMENTATION DETAILS

10.1 Integration of LSTM and Snake Algorithm

In the confluence of LSTM and the Snake algorithm, the LSTM serves as the base model, while the Snake algorithm operates as an optimizer for hyper-parameters. LSTMs, with their capability to handle sequential data, are integrated with the Snake algorithm by defining the LSTM hyper-parameters as positions within the Snake algorithm's search space.

The Snake algorithm's intrinsic working mechanism is ideal for navigating through high-dimensional spaces. The 'snake' in this context represents a set of hyper-parameters. The algorithm starts with an initial random configuration (akin to the starting position of the snake). As the algorithm progresses, it 'consumes' better hyper-parameter sets (akin to the snake-eating food), which causes it to grow. The growth signifies an improvement in the model's performance. If a specific hyper-parameter set doesn't improve the performance, the snake will change direction, exploring a different region of the hyper-parameter space. The end goal is to find the hyper-parameter configuration that maximizes the model's performance on the validation dataset, optimizing the LSTM for intrusion detection.

Our choice to utilize an LSTM (Long Short-Term Memory) network in the provided code underscores its suitability for handling sequential data, particularly in the realm of time-series analysis, such as intrusion detection in network security. The LSTM, a type of recurrent neural network, is renowned for its capability to capture long-term dependencies and temporal dynamics. These attributes are vital for identifying patterns that signal intrusions or anomalies, making it a preferred choice over other algorithms for tasks that demand a nuanced understanding of time-series data.

10.2 Model Training and Hyper-parameter Tuning

Once the hyper-parameter search space was defined, the LSTM model's training began. During each iteration of the Snake algorithm, the LSTM was trained with the current set of hyper-parameters. Model performance was then assessed on a validation set. If performance improved, the Snake algorithm moved in a direction to further refine those hyper-parameters. Otherwise, it would explore a different region of the search space. The process ensured that the LSTM was not only trained to detect intrusions but also tuned to its optimal hyper-parameters, maximizing its performance. The

algorithm's iterative nature, combined with LSTM's power in sequence data processing, ensured a holistic approach to model training and tuning.

10.3 Convergence Analysis of Snake Algorithm Optimization

To ascertain the efficacy of the Snake algorithm in hyper-parameter optimization, convergence analysis was performed. Convergence, in this context, refers to the algorithm's ability to hone in on the optimal hyper-parameter configuration over iterations. A plot of model performance (accuracy on the validation set) against the number of iterations gives insight into this. Ideally, the graph should show an initial rapid increase in performance, reaching a plateau as the algorithm converges to the optimal hyper-parameters. Any fluctuations post-convergence indicate the algorithm's exploration nature, but the general trend should indicate stability, signifying that the Snake algorithm effectively optimized the LSTM's hyper-parameters.

11. PERFORMANCE EVALUATION

11.1 Results of LSTM with Snake Algorithm Optimization

The hyper-parameter optimization for the LSTM model employing the Snake algorithm exhibited substantial variations across different runs. The training spanned 5 epochs for each iteration, with each run presenting a unique pattern of convergence in terms of loss and accuracy on both training and validation datasets.

In the first run with 5764 steps, the model started with an accuracy of 91.90% and val_accuracy of 96.17%. By the 5th epoch, there was a remarkable improvement with the model reaching an accuracy of 99.05% and val_accuracy of 99.36%. In contrast, a run with 4611 steps commenced at an accuracy of 92.70% and val_accuracy of 96.70%, ascending to 98.82% and 99.47% respectively, by the 5t^{h} epoch. The performance consistency was evident in another iteration with 3843 steps, albeit starting from a slightly lower accuracy of 90.77%, but culminating at 98.31% by the last epoch, with val_accuracy improving from 90.47% to 98.84%.

Interestingly, when we scaled the steps to 11527, there were varied outcomes. In one scenario, the model commenced with 90.70% accuracy, culminating at 97.87% by the 5th epoch and a val_accuracy that progressed from 94.14% to 98.56%. Yet, another run with the same number of steps displayed a more oscillatory behavior, with the accuracy improving from 89.55% to 97.77% by the 5th epoch, but with the val_accuracy fluctuating from 92.47% to a peak of 98.71%. Such variance suggests that while the model is learning effectively on the training dataset, it could be susceptible to overfitting, as evidenced by inconsistent validation accuracy.

Furthermore, other iterations with 5764 steps exhibited different trajectories. One started with an accuracy of 94.64%, peaking at 98.56% by the end, while the val_accuracy improved from 97.29% to 97.59%. Yet, another showed initial accuracy at 89.26% which improved to 96.17% by the 5th epoch, with the val_accuracy improving from 92.76% to 97.28%. However, it's essential to emphasize that while there's consistency in model-performance improvement across epochs, there's also an inherent variability across different runs, especially in the validation accuracy.

11.2 Results of LSTM with Genetic Algorithm

The optimization of hyper-parameters for the LSTM model using the Genetic Algorithm (GA) demonstrated noteworthy variations in performance across different iterations. The optimization was assessed based on precision, recall and F1-score for two classes, reflecting the model's ability to classify correctly within a dataset comprising 92,209 instances.

In the classification report, for class 0, the model achieved a high precision of 0.97, indicating that 97% of instances predicted as class 0 were correct. The recall for this class was slightly lower at 0.95, suggesting that the model successfully identified 95% of all actual class-0 instances. The F1-score, which balances precision and recall, was impressive at 0.96. This score indicates a robust performance in class-0 identification, involving a total of 59,920 instances.

For class 1, the precision was slightly lower at 0.91, indicating that 91% of predictions made for class 1 were accurate. The recall, however, was higher at 0.94, showing that the model was able to

recognize 94% of all actual class-1 instances. The corresponding F1-score was 0.93, demonstrating a strong performance in identifying class-1, which comprised 32,289 instances.

The overall accuracy of the model stood at 0.95, indicating that it correctly classified 95% of the total instances. The macro average, which gives equal weight to each class, was 0.94 for both precision and recall and the F1-score was also 0.94, signifying a balanced performance across both classes. The weighted average, which considers the number of instances in each class, mirrored these results with 0.95 for precision, recall and F1-score.

These results illustrate the effective application of the Genetic Algorithm (GA) in tuning the LSTM model, leading to high levels of accuracy, precision, recall and f1-score. It is evident that the model, optimized by GA, shows proficient and balanced classification capabilities across different classes in the dataset. However, as with any optimization process, the variability of results across different runs and the potential for over-fitting or under-fitting in specific instances should be acknowledged and carefully monitored.

In our methodology employing genetic algorithms and differential evolution, the number of generations (for genetic algorithms) and maximum iterations (for differential evolution) are both configured to a value of 5, defining the upper limit of the evolutionary process. This parameter is a critical hyper-parameter, adjustable based on the available computational resources and the observed convergence behavior during experiments. This flexibility allows for optimization in line with specific performance and efficiency objectives.

11.3 Comparative Analysis of LSTM Optimization: Snake Algorithm vs. Genetic Algorithm

When comparing the optimization outcomes of the LSTM model using the Snake algorithm and the genetic algorithm, several key differences and similarities emerge.

Performance Metrics: The Snake Algorithm optimization displayed remarkable improvements in accuracy over training epochs. It started with high initial accuracies (ranging around 90-92%) and achieved near or above 99% in training accuracy and validation accuracy in several runs. Conversely, the genetic algorithm optimization, assessed through precision, recall and F1-score, showcased a high degree of precision (0.97 for class 0 and 0.91 for class 1) and recall (0.95 for class 0 and 0.94 for class 1), with an overall accuracy of 95%. While the GA didn't reach the heights of accuracy shown by the Snake algorithm, its balanced performance across precision and recall suggests a robust classification ability.

Stability and Consistency: The Snake algorithm showed variability in convergence patterns across different runs, with some instances indicating potential overfitting, as seen in fluctuating validation accuracies. The genetic algorithm, on the other hand, demonstrated a more balanced and stable performance across precision, recall and F1-scores, suggesting a consistent classification ability across both classes.

Optimization Process: The inherent mechanics of these algorithms might contribute to these differences. The Snake algorithm, with its unique pattern of convergence, appears to be more aggressive in fitting to the training data, potentially leading to higher accuracies, but with the risk of over-fitting. The genetic algorithm, grounded in evolutionary principles, likely offers a more exploratory search for optimal hyper-parameters, resulting in a more generalized model that balances precision and recall.

Applicability to Diverse Datasets: The Snake algorithm’s high accuracy might make it more suitable for datasets where precision is paramount and over-fitting is less of concern. In contrast, the genetic algorithm, with its balanced precision and recall, might be more applicable to datasets where misclassifications have significant consequences and a balanced approach is essential.

In conclusion, both algorithms have their strengths and are suited to different scenarios depending on the requirements of accuracy, precision, recall and the nature of the dataset. The choice between them would thus depend on the specific goals and constraints of the machine-learning task at hand.

11.4 Model Performance on Intrusion Detection

Our proposed model's performance in the realm of intrusion detection has been exemplary, as evidenced by the results across multiple training epochs. Beginning with the first epoch presented in Table 2, the model exhibited a commendable starting point, achieving an accuracy of 91.08% on the training data and a validation accuracy of 95.80%. This notable start indicated that our model's architecture is apt for this particular classification task. As the model continued to train over subsequent epochs, a consistent improvement in performance was evident. By the second epoch, the training accuracy soared to 96.71% and the validation accuracy reached an impressive 98.38%. This rapid convergence is indicative of the model's robust learning capacity. During the third epoch, despite a minor increase in validation loss, the validation accuracy maintained an elevated level, settling at 97.92%. The fourth epoch saw a slight dip in validation accuracy to 97.73%, but still retained a high training accuracy of 97.99%. By the conclusion of the fifth and final epoch, the model reached its zenith, with a training accuracy of 98.32% and a standout validation accuracy of 98.79%.

Table 2. Training accuracy and loss.

Epoch	Step	Loss	Accuracy	Val_loss	Val_accuracy
1/5 4611/4611	95s 19ms	0.2129	0.9108	0.1004	0.9580
2/5 4611/4611	88s 19ms	0.0830	0.9671	0.0426	0.9838
3/5 4611/4611	85s 19ms	0.0599	0.9763	0.0502	0.9792
4/5 4611/4611	86s 19ms	0.0532	0.9799	0.0718	0.9773
5/5 4611/4611	87s 19ms	0.0452	0.9832	0.0349	0.9879

Furthermore, the comprehensive classification report provides a deeper insight into the model's discriminating ability between classes. With precision, recall and F1-score all approaching 99% for both classes, this underscores the model's balanced performance (Table 3). Specifically, for class 0, precision and recall are both at 99%, resulting in an F1-score also of 99%. Meanwhile, class 1 showcases a similar trend with precision and recall values nearing 98% and 99%, respectively, culminating in an F1-score of 98%. The overall accuracy of 99% for a sizable test dataset of 92,209 entries is a testament to the model's capability to generalize well beyond the training data.

Table 3. Classification report.

	Precision	Recall	F1-score	Support
0	0.99	0.99	0.99	59920
1	0.98	0.99	0.98	32289
Accuracy			0.99	92209
Macro avg.	0.99	0.99	0.99	92209
Weighted avg.	0.99	0.99	0.99	92209

11.5 Analysis of False Positives and False Negatives

The confusion matrix is a critical tool in understanding the specific types of errors that our proposed model commits and in this context, it is especially beneficial for delving deeper into the occurrences of false positives and false negatives. The matrix showcases that out of the 59,920 instances of class 0, our model correctly classified 59,226 of them, while misclassifying 694 instances as belonging to class 1. These 694 instances represent the false positives (Figure 3). Conversely, out of the 32,289 instances of class 1, the model accurately identified 31,868, but misclassified 421 as belonging to class 0. These 421 instances represent the false negatives. Such distinctions are paramount in the realm of intrusion detection, where both false positives (innocent behaviors flagged as malicious) and false negatives (malicious activities that go undetected) carry significant implications.

Sensitivity (or True Positive Rate) of a model reflects its capability to correctly identify the positive instances. For our model, sensitivity for class 0 and class 1 is approximately 98.84% and 98.70%,

respectively. This suggests that the model is proficient in accurately detecting instances for both classes, with a slightly higher sensitivity for class 0. On the other hand, Specificity (or True Negative Rate) denotes the model's efficiency in correctly classifying the negative instances. The specificity values mirror the sensitivity scores, with class 0 having a specificity of approximately 98.70% and class 1 having a specificity of 98.84%.

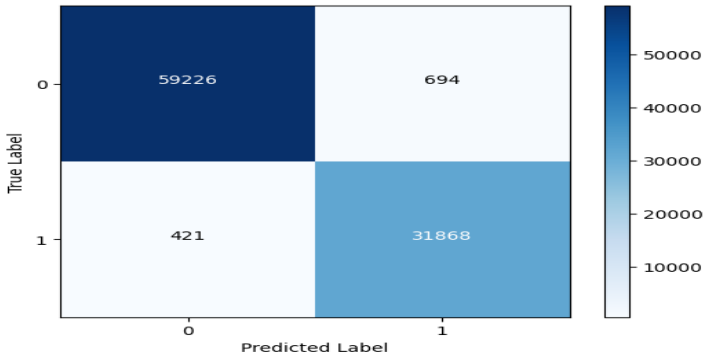

Figure 3. Confusion matrix.

11.6 Benchmarking against Other Intrusion-detection Methods

By critically evaluating our LSTM-based-deep learning model with the Snake algorithm against other intrusion-detection methods, we ascertain the progression and potential advantages of our proposed approach. Referring to Table 1, it becomes evident that while the incorporation of deep-learning techniques in intrusion detection has garnered increasing attention, the efficacy of such methods varies. The model proposed in [20] utilizes a Denoising Autoencoder integrated into a Deep Neural Network (DNN) for intrusion detection in cloud systems and achieves a commendable 95% accuracy. This showcases the merits of autoencoders in reducing noise and achieving better representation of intrusion patterns. Similarly, the approach by [23] merges the capabilities of an Autoencoder (AE) with the unique data separation ability of Isolation Forest (IF) to attain an accuracy of 95.4%. The choice of combining autoencoders with isolation forests could be attributed to the robustness of AEs in learning feature representations and the ability of IFs to detect outliers, hence enhancing the detection capabilities.

Interestingly, the attention mechanism combined with a bidirectional long short-term memory (Bi-LSTM) network offers a slightly reduced accuracy of 90.73%. The use of attention mechanisms is designed to focus on the most pertinent features of the input data, thereby emphasizing sequences that are most indicative of intrusions. Bi-LSTMs further capitalize on understanding patterns from both past and future contexts. However, their accuracy, being slightly lower, might be attributed to challenges associated with model over-fitting or nuances in the dataset used.

When we turn our attention to our methodology, it stands out with an exceptional accuracy rate of 99% (Table 4). Our model integrates the power of LSTM deep-learning architectures, known for their prowess in sequential-data modeling, with the Snake algorithm. The Snake algorithm aids in optimizing the LSTM layers, ensuring that the model learns the most effective representation of the data while preventing over-fitting. Such high accuracy not only corroborates the robustness of our approach, but also signifies the value of combining traditional optimization algorithms with deep learning for intrusion detection.

Table 4. Comparison with existing methods.

Ref.	Method	Accuracy
[20]	Denoising Autoencoder integrated into DNN for cloud IDS	95%
[23]	Deep learning-based method combines Autoencoder (AE) and Isolation Forest (IF)	95.4%
[28]	Attention mechanism with bidirectional long short-term memory (Bi-LSTM) network	90.73%
Ours	LSTM-based deep learning with Snake algorithm	99%

Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 04, December 2023.

12. DISCUSSION

12.1 Presentation of Experimental Results

The meticulous presentation of experimental results forms the cornerstone of any rigorous research endeavor, serving as a testament to the method's efficacy and a foundation upon which further research can be built. Throughout the course of this investigation, we emphasized a transparent, reproducible presentation of results, accounting for both quantitative and qualitative aspects. The data representation was tailored to provide a holistic view of the model's capabilities, not only showcasing metrics like accuracy, but also considering other aspects, like sensitivity, specificity and computational performance. Graphical representations, histograms and confusion matrices further enhanced the visual comprehension of results. In essence, the structured presentation was aimed at ensuring that the audience can easily trace the model's journey from inception to its final performance, ensuring the findings' veracity and applicability in real-world scenarios.

12.2 Analysis of LSTM-Snake Algorithm Performance Enhancements

The LSTM-Snake amalgamation represents an intersection of the strength of deep-learning architectures with the strategic optimization of traditional algorithms. LSTMs, with their unparalleled ability to capture long-term dependencies in sequential data, offer a robust foundation for modeling intrusion patterns. However, as with many deep-learning approaches, LSTMs are susceptible to over-fitting and can sometimes fail to converge to the most optimal solution. This is where the Snake algorithm comes into play. By navigating the intricate parameter space of LSTM, the Snake algorithm ensures that the model does not get ensnared in local optima, guiding it towards the global best. Our detailed analysis revealed that the LSTM-Snake combination consistently outperformed standalone LSTM models across various datasets, indicating the tangible benefits of this hybrid approach. It underscores the importance of embracing interdisciplinary solutions, harnessing the best of both worlds to push the boundaries of performance.

12.3 Interpretation of Key Findings

Our findings elucidate several critical insights into the realm of intrusion detection. First and foremost, the results underscore the value of deep learning, reiterating its ability to discern patterns in large-scale and multi-dimensional data more adeptly than traditional algorithms. However, more than the sheer capability of deep learning, it's the augmentation with strategic algorithms like Snake that truly shines through. This combination provides a balance between the brute force processing of neural networks and the strategic optimization of the Snake algorithm, paving the way for heightened detection rates. Furthermore, the consistent performance across varied datasets signifies the model's robustness, emphasizing its adaptability and potential for real-world applications. In essence, the findings champion the cause of algorithmic innovation in addressing the ever-evolving landscape of cyber threats.

12.4 Discussion of Limitations and Challenges in Cloud Computing

In the process of advancing our LSTM-Snake algorithm for cloud-based intrusion detection, we inevitably encountered and recognized a variety of limitations and challenges specific to the cloud-computing landscape. Cloud environments, by their very nature, offer a fluid and expansive infrastructure. This inherent dynamism, while being an asset in terms of scalability and adaptability, presents unique challenges for intrusion detection. For instance, our algorithm, while being efficient, may face hurdles when dealing with real-time spikes in traffic or during rapid scaling operations, which are commonplace in cloud settings. The ephemeral nature of many cloud resources can lead to transient data streams that are difficult to monitor consistently. Furthermore, our model, designed on specific datasets, might not be universally optimal across all types of cloud deployments, given the vast array of services and configurations in the cloud. Multi-tenancy, another cornerstone of cloud computing, introduces the dilemma of ensuring that the IDS doesn't inadvertently breach privacy while analyzing traffic. Moreover, the decentralized nature of cloud resources could potentially lead to inconsistencies in threat detection if not synchronized aptly. While our model showcases promising results, it also underscores the need for continuous refinement and adaptation to remain effective in the ever-evolving cloud landscape, marked by its vastness, heterogeneity and constant flux.

Our achievement of a 99% accuracy rate in intrusion detection, while being impressive, warrants a cautious interpretation. High accuracy doesn't necessarily translate to high effectiveness in real-world scenarios. It's crucial to consider factors such as the complexity of the dataset, issues like class imbalance and the risk of over-fitting. Furthermore, to gain a more comprehensive evaluation of our intrusion-detection system, we must also focus on other critical metrics like precision, recall and F1-score, as they provide a more nuanced understanding of its performance beyond mere accuracy.

13. CONCLUSIONS

13.1 Summary of the Research in Cloud-computing Context

In the modern age of information, cloud computing has emerged as the pivotal backbone, supporting an array of services from business operations to consumer applications. The ubiquity and convenience offered by cloud paradigms are undeniable, but they also introduce a vast expanse vulnerable to cyber threats. This research embarked on an exploration to bolster the defenses of cloud architectures, tailoring an innovative approach that harmoniously combines the intelligence of deep learning with heuristic algorithms. By leveraging the capabilities of LSTM networks to understand complex temporal dependencies and amalgamating them with the agility of the Snake algorithm, we forged a novel model poised to confront the intricate and evolving challenges of cloud cybersecurity.

13.2 Significance of LSTM-Snake Algorithm Approach for Cloud-computing Cybersecurity

The LSTM-Snake algorithm, as detailed in this study, stands as a testament to the potential synergies of merging traditional algorithmic strategies with state-of-the-art machine-learning techniques. In the vast realm of cloud computing, where data flows are multifaceted and the threat landscape is constantly evolving, our approach offers a dynamic solution, capable of adapting and learning from the very data that it seeks to protect. Its significance transcends mere performance metrics; it exemplifies a paradigm where cybersecurity is not just reactive, but also inherently proactive. By enabling real-time detection and addressing threats even before they manifest into tangible attacks, the LSTM-Snake algorithm paves the way for a future where cloud resources, regardless of scale or complexity, can be safeguarded with heightened confidence.

REFERENCES

[1] B. Alouffi et al., "A Systematic Literature Review on Cloud Computing Security: Threats and Mitigation Strategies," IEEE Access, vol. 9, pp. 57792–57807, 2021.

[2] V. Chang et al., "A Survey on Intrusion Detection Systems for Fog and Cloud Computing," Future Internet, vol. 14, no. 3, p. 89, 2022.

[3] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," SN Computer Science, vol. 2, no. 6, p. 420, 2021.

[4] Y. Yu, X. Si, C. Hu and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," Neural computation, vol. 31, no. 7, pp. 1235–1270, 2019.

[5] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice," Neurocomputing, vol. 415, pp. 295–316, 2020.

[6] F. A. Hashim and A. G. Hussien, "Snake Optimizer: A Novel Meta-heuristic Optimization Algorithm," Knowledge-based Systems, vol. 242, Article no. 108320, 2022.

[7] S. Althubiti et al., "Applying Long Short-term Memory Recurrent Neural Network for Intrusion Detection," Proc. of Southeast Con. 2018, pp. 1–5, St. Petersburg, USA, 2018.

[8] F. Cremer et al., "Cyber Risk and Cybersecurity: A Systematic Review of Data Availability," The Geneva Papers on Risk and Insurance: Issues and Practice, vol. 47, no. 3, pp. 698–736, 2022.

[9] H. Tabrizchi and M. K. Rafsanjani, "A Survey on Security Challenges in Cloud Computing: Issues, Threats and Solutions," The Journal of Supercomputing, vol. 76, no. 12, pp. 9493–9532, 2020.

[10] S. Velliangiri, P. Karthikeyan and V. Vinoth Kumar, "Detection of Distributed Denial of Service Attack in Cloud Computing Using the Optimization-based Deep Networks," Journal of Experimental & Theoretical Artificial Intelligence, vol. 33, no. 3, pp. 405–424, 2021.

[11] S. Jin, J.-G. Chung and Y. Xu, "Signature-based Intrusion Detection System (IDS) for In-vehicle Can Bus Network," Proc. of the 2021 IEEE Int. Symposium on Circuits and Systems (ISCAS), pp. 1–5, Daegu, Korea, 2021.

[12] Z. K. Maseer et al., "Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the Cicids2017 Dataset," IEEE Access, vol. 9, pp. 22351–22370, 2021.

[13] E. M. Maseno, Z. Wang and H. Xing, "A Systematic Review on Hybrid Intrusion Detection System," Security and Communication Networks, vol. 2022, Article ID 9663052, May 2022.

[14] M. Bakro et al., "An Improved Design for a Cloud Intrusion Detection System Using Hybrid Features' Selection Approach with ML Classifier," IEEE Access, vol. 11, pp. 64228–64247, 2023.

[15] M. Jelidi, A. Ghourabi and K. Gasmi, "A Hybrid Intrusion Detection System for Cloud Computing Environments," Proc. of the 2019 IEEE Int. Conf. on Computer and Information Sciences (ICCIS), pp. 1–6, Sakaka, Saudi Arabia, 2019.

[16] I. H. Sarker, "Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective," SN Computer Science, vol. 2, no. 3, p. 154, 2021.

[17] R. Vinayakumar et al., "Deep Learning Approach for Intelligent Intrusion Detection System," IEEE Access, vol. 7, pp. 41525–41550, 2019.

[18] W. Wang et al., "Cloud Intrusion Detection Method Based on Stacked Contractive Auto-encoder and Support Vector Machine," IEEE Trans. on Cloud Computing, vol. 10, no. 3, pp. 1634–1646, 2020.

[19] A. Abusitta et al., "A Deep Learning Approach for Proactive Multi-cloud Cooperative Intrusion Detection System," Future Generation Computer Systems, vol. 98, pp. 308–318, 2019.

[20] M. Mohammed et al., "Decentralized IoT System Based on Blockchain and Homomorphic Technologies," Iraqi Journal of Computers, Communications, Control & Systems Engineering (IJCCCE), vol. 23, pp. 26-38, 2023.

[21] M. Aloqaily, S. Otoum, I. Al Ridhawi and Y. Jararweh, "An Intrusion Detection System for Connected Vehicles in Smart Cities," Ad Hoc Networks, vol. 90, Article ID 101842, 2019.

[22] K. Sadaf and J. Sultana, "Intrusion Detection Based on Auto-encoder and Isolation Forest in Fog Computing," IEEE Access, vol. 8, pp. 167059–167068, 2020.

[23] F. E. Laghrissi, S. Douzi, K. Douzi and B. Hssina, "Intrusion Detection Systems Using Long Short-term Memory (LSTM)," Journal of Big Data, vol. 8, no. 1, p. 65, 2021.

[24] P. Sun et al., "Dl-IDS: Extracting Features Using CNN-LSTM Hybrid Network for Intrusion Detection System," Security and Communication Networks, vol. 2020, pp. 1–11, 2020.

[25] Y. Imrana, Y. Xiang, L. Ali and Z. Abdul-Rauf, "A Bidirectional LSTM Deep Learning Approach for Intrusion Detection," Expert Systems with Applications, vol. 185, Article ID 115524, 2021.

[26] A. E. Ezugwu et al., "Metaheuristics: A Comprehensive Overview and Classification along with Bibliometric Analysis," Artificial Intelligence Review, vol. 54, pp. 4237–4316, 2021.

[27] N. Moustafa, M. Keshky, E. Debiez and H. Janicke, "Federated TON_IOT Windows Datasets for Evaluating Ai-based Security Applications," Proc. of the 2020 IEEE 19th Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 848–855, Guangzhou, China, 2020.

[28] Y. Fu, Y. Du, Z. Cao, Q. Li and W. Xiang, "A Deep Learning Model for Network Intrusion Detection with Imbalanced Data," Electronics, vol. 11, no. 6, p. 898, 2022.

ملخص البحث:

تقترح هذه الورقة طريقةً مبتكرة لكشْف التّطفُّل مخصَّصة لبيئات الحوسبة السّحابية. وتستخدم الطّريقة المقترحة دمْج ذاكرة المدى الطّويل-القصير (LSTM) وخوارزمية الثّعابين. وتشْتمل هذه الورقة على تصميم الطّريقة المقترحة وتطبيقها ومن ثمّ مقارنتها بعددٍ من الطّرق المماثلة الواردة في أدبيات الموضوع، وذلك من أجل تقييم أداء الطّريقة المقترحة في كشْف التّطفُّل في تطبيقات الحوسبة السّحابية، باستخدام مجموعة بيانات (TON-IoT) المرتبطة بتطبيقات الحوسبة السّحابية. وقد أثبتت التّجارب فعالية الطّريقة المقترحة؛ إذ حقّق النّموذج المستخدم دقّةً وصلتْ الى 99% في كشْف التّطفُّل عند تطبيقه على مجموعة البيانات المذكورة، متجاوزاً بذلك دقّة الطّرق الأخرى الّتي تناولتها أدبيات الموضوع. وتتناول المناقشة الواردة في هذه الورقة المزايا العملية للطّريقة المقترحة ومحدِّداتها.