ESJ Natural/Life/Medical Sciences
Peer-reviewed

Enhancing Portal System Resilience with a Modified Lion Optimization Algorithm (MLOA) for Cyber Threat Detection

O.O. Green
Department of Information Communication Technology, Lagos State University of Education, Lagos, Nigeria
M.B. Abdulrazaq
B. Yahaya
Z. Haruna
Department of Computer Engineering,
Ahmadu Bello University, Zaria, Nigeria
S.O. Omogoye
Department of Electrical and Electronics Engineering, Lagos State University of Science and Technology, Lagos, Nigeria
A.S. Adegoke
Department of Computer Engineering, Lagos State University of Science and Technology, Lagos, Nigeria

Doi:10.19044/esj.2025.v21n9p61

Submitted: 05 February 2025
Accepted: 08 March 2025
Published: 31 March 2025

Cite As:
Green O.O., Abdulrazaq M.B., Yahaya B., Haruna Z., Omogoye S.O. & Adegoke A.S. (2025). *Enhancing Portal System Resilience with a Modified Lion Optimization Algorithm (MLOA) for Cyber Threat Detection.* European Scientific Journal, ESJ, 21 (9), 61. https://doi.org/10.19044/esj.2025.v21n9p61

Abstract

This research presents a novel cyber threat detection framework that integrates the Modified Lion Optimization Algorithm (MLOA) with a one-class classification approach to improve the resilience of portal systems against denial-of-service attacks, Man-in-the-Middle attacks, and data breaches. The proposed model enhances anomaly detection by optimizing decision boundaries in high-dimensional datasets, leveraging adaptive threshold tuning, dynamic feature selection, and real-time monitoring. Experimental evaluations demonstrate that the MLOA-based detection model significantly outperforms traditional clustering-based methods across varying levels of attack complexity. It achieves a recall of 0.97, accuracy of 0.98,

ISSN: 1857-7881 (Print) e - ISSN 1857-7431

precision of 0.96, and an area under the receiver operating characteristic curve (ROC-AUC) score of 0.97 for simple anomalies, while maintaining strong performance for moderate and complex anomalies, with recall values of 0.92 and 0.90 and ROC-AUC scores of 0.94 and 0.92. These findings validate the effectiveness of the proposed approach in detecting zero-day attacks and evolving cybersecurity threats, offering a scalable, high-performance anomaly detection solution for modern portal systems. This study further establishes the practical application of nature-inspired optimization algorithms in cybersecurity, reinforcing the importance of AI-driven threat detection in protecting digital infrastructure.

Keywords: Anomaly Detection, Cybersecurity, Modified Lion Optimization Algorithm, Nature-Inspired Algorithms, Performance Metrics, Portal Systems, SSC-OCSVM, UNSW-NB15 Dataset

Introduction

In today’s digital landscape, portal systems (PS) have become integral to delivering critical education, administration, and communication services. However, these systems' increasing complexity and interconnectivity make them vulnerable to diverse anomalies, including malicious attacks, injection flaws, denial-of-service (DoS) attacks, Man-in-the-Middle attacks (MitM) data breaches, and human errors. These vulnerabilities cause operational disruptions, financial losses, and reputational damage, leading to reduced user trust. For example, major cyber incidents like the July 2015 data breach at the University of California, Los Angeles (UCLA), which exposed 4.5 million records at a cost of over $70 million, and the July 2023 University of Manchester was a victim of a cyber-attack, resulting in vulnerabilities for about 11,000 staff and more than 46,000 students' data(Paganini, 2023)), highlight the severe consequences of insufficient anomaly detection systems. In Nigeria, the 2023 presidential elections recorded 12.9 million cyber threats reported by the minister of communication and digital economy, Isa Pantami (Ukagwu. (2023)), further emphasizing the need for robust security mechanisms.

Study Aim and Objectives

This study aims to develop an anomaly detection model for portal systems by integrating the Modified Lion Optimization Algorithm (MLOA) with One-Class Support Vector Machine (OCSVM). The primary objective is to determine the optimal anomaly detection threshold for distinguishing between normal and abnormal network activity, enhancing portal system security and reliability. The initiative specifically aims to:

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

1) Address the limitations of existing anomaly detection methods, such as high false positive rates and computational inefficiencies.
2) Optimize anomaly detection thresholds using MLOA's adaptive optimization techniques. Improve detection accuracy for complex anomalies.
3) Evaluate the model's effectiveness using real-world datasets from the Lagos State University of Education (LASUED) portal systems and the UNSW-NB15 dataset.

Limitations of Conventional Approaches

Conventional anomaly detection techniques, including trial-and-error methods or default parameter settings, often fail to adapt to cyber threats' dynamic and evolving nature. Techniques such as Sub-Space Clustering-One Class Support Vector Machine (SSC-OCSVM) developed by (Pu *et al*., 2021), Feature selection with K-Lion Optimization Algorithms (K-LOA) by (Jagatheeshkumar *et al.,* 2021), and deep-learning hybrid models by Data, Karadayi and Aydin, (2020) have shown promise but are limited in handling complex anomaly patterns like contextual User-to-Root (U2R) and Remote-to-Local (R2L) attacks. These approaches may result in high false positive rates or leave systems vulnerable to subtle yet impactful threats.

Proposed Solution

To address these limitations, this study proposes the Modified Lion Optimization Algorithm (MLOA) to enhance anomaly detection by optimizing thresholds and improving computational efficiency. The MLOA leverages advanced nature-inspired optimization techniques to adapt dynamically to diverse anomaly types, ensuring improved detection accuracy and robustness in real-world scenarios. By utilizing datasets from Lagos State University of Education (LASUED) portal systems and the UNSW-NB15 dataset, the proposed solution aims to safeguard portal systems from evolving cyber threats.

The remainder of this article is organized as follows: Section 2 provides an overview of the proposed MLOA framework and its application in portal systems. Section 3 presents the research findings and discussions. Section 4 concludes the study and outlines potential directions for future research.

Overview of Proposed MLOA Anomaly Detection Threshold in a Portal System

Introduction

Portal systems must be resilient to maintain operational integrity, user trust, and cybersecurity. The early detection and mitigation of anomalies

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

guarantee the system's resilience and recovery from changing cyber threats, including denial-of-service (DoS) attacks, Man-in-the-Middle (MitM) attacks, and data breaches.

This paper proposes an anomaly detection threshold for the Modified Lion Optimization Algorithm (MLOA) to improve portal systems' early threat detection. To maximize anomaly detection in high-dimensional datasets, the suggested system combines MLOA with One-Class Support Vector Machine/OCSVM.

Justification for the MLOA-Based Anomaly Detection Model

Traditional anomaly detection methods, such as default threshold settings, clustering techniques, and deep learning models, suffer from:

1) High false positive rates due to static thresholding.
2) Computational inefficiencies, making real-time deployment impractical.
3) Limited adaptability to emerging cyber threats.

The MLOA framework addresses these limitations through:

1) Dynamic Threshold Optimization – Continuously adjusts detection boundaries based on real-time network traffic analysis
2) Feature Selection and Search Space Optimization – Reduces irrelevant features to enhance computational efficiency.
3) Adaptive Learning – Adjusts model parameters iteratively to improve accuracy for complex anomaly patterns.

Structure of the MLOA Anomaly Detection Framework

The proposed MLOA-based anomaly detection system consists of three core components:

1) Data Collection and Preprocessing: Network logs are collected from the Lagos State University of Education (LASUED) portal system and the UNSW-NB15 dataset and the data undergo cleaning, normalization, and feature selection.
2) Anomaly Detection Model Development: The MLOA is used to optimize the anomaly detection threshold for distinguishing between normal and anomalous traffic and Feature selection techniques are applied to enhance model efficiency and reduce computational load.
3) Real-Time Anomaly Detection: The system continuously evaluates network traffic, adjusting detection thresholds dynamically based on evolving threat patterns

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

Methodology

This section describes developing and implementing the Modified Lion Optimization Algorithm (MLOA) for anomaly identification in portal systems. The methodology includes data collection and preprocessing, the search space definition, feature selection, model optimization, and training procedures. By optimizing anomaly detection criteria and guaranteeing flexibility in response to changing attack patterns, the proposed system aims to improve early cyber threat identification.

Data Collection and Preprocessing

In this article, the network logs (dataset) were collected from the Lagos State University of Education (LASUED) Edu portal (www.eportal.lasued.edu.ng) using network traffic monitoring software called OPNsense by Thomas-Krenn, A.G. (2018), the software uses comma-separated values (CSV), to store all the network logs that pass through it, by adopting the work of (Konstantina et al., 2021) and the UNSW-NB15 dataset downloaded from the open-source (Australian Centre for Cyber Security by *Nour Moustafa, (2015)*). This dataset contains a variety of network activities, including normal traffic and modern attacks, representing contemporary threats faced by network systems, it includes 2.54 million records, each containing network traffic data along with labeled attacks and normal activities organized with different features capturing details of each network connection, with labels indicating whether each instance is normal or malicious.

Adopting the work of (*Konstantina et al., 2021*) as shown in Figure 1., the datasets were preprocessed, and the UNSW-NB15 dataset and network logs were imported into Python using the pandas package for preprocessing. Missing values in both datasets were identified using the IsNull *().sum()* function and replaced with their respective feature means through the *SimpleImputer* from *sklearn.impute.* This preprocessing pipeline effectively cleaned and prepared the datasets, optimizing them for subsequent anomaly detection modeling.

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

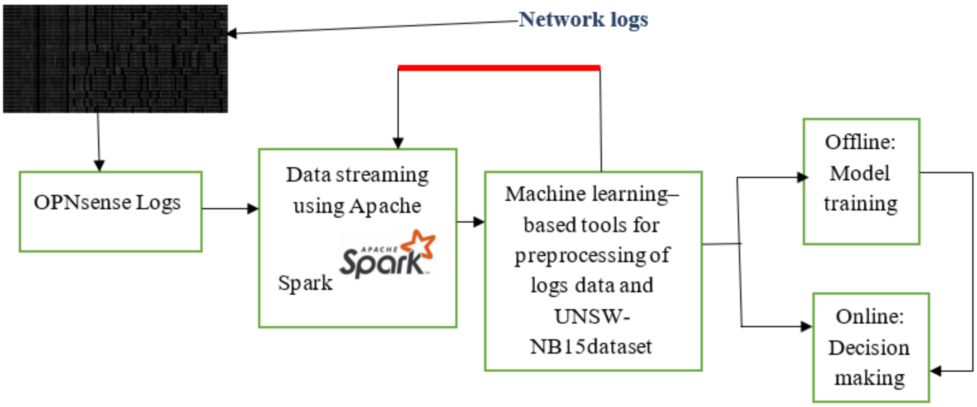


Figure 1: Network Data Collection Adopting (Konstantina., 2021)'s Work

Development and Training of the MLOA

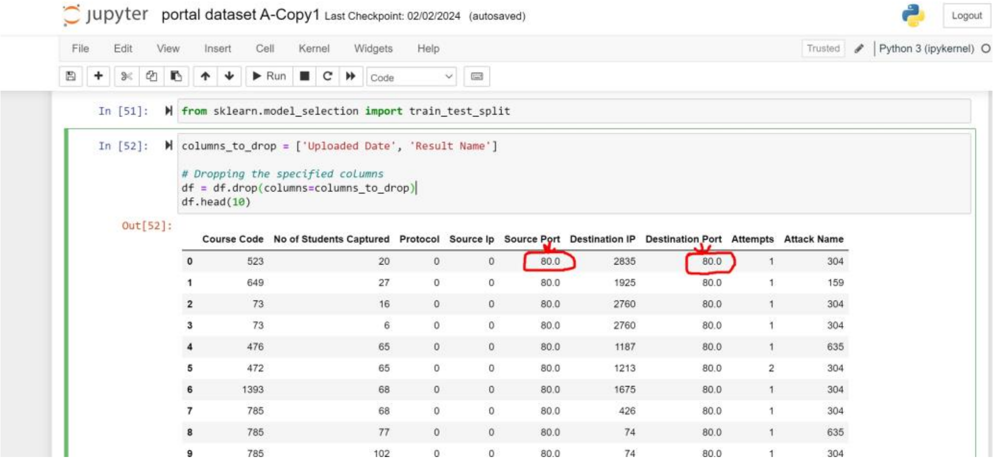
The Modified Lion Optimization Algorithm (MLOA) was developed to enhance anomaly detection thresholds by incorporating adaptive search space optimization, dynamic feature selection, and real-time threshold tuning. These improvements address the limitations of traditional optimization algorithms, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), which often suffer from computational inefficiency, premature convergence, and sensitivity to parameter settings. Genetic Algorithms (GA), while effective for global search, are computationally expensive due to the need to evaluate many individuals across multiple generations. They are also prone to premature convergence if the population loses diversity, leading to suboptimal solutions. Additionally, GA requires careful parameter tuning (e.g., population size, crossover rate, mutation rate, and the number of generations) to achieve optimal performance(Katoch and Kumar, 2021). Similarly, Particle Swarm Optimization (PSO) faces challenges such as converging prematurely to local optima, particularly when particle diversity decreases too early. The algorithm's performance is highly sensitive to parameter settings, especially inertia weight, cognitive weight, and social weight, making it difficult to adapt in dynamic environments(Vanneschi and Silva, 2023). To overcome these challenges, MLOA integrates adaptive search mechanisms that balance exploration and exploitation, ensuring robust anomaly detection across high-dimensional datasets.

Feature Selection and Search Space Definition

Adopting Rajakumar, (2012)'s work, using the preprocessed network logs (dataset) collected and the UNSW-NB15 dataset selected features as detailed in the Python code in Appendix A. This research delves into several steps and methods utilized in developing and implementing the Modified Lion Optimization Algorithm framework for early anomaly detection in a portal

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

system. The population size for the Modified Lion Optimization Algorithm (MLOA) was determined based on the Internet Assigned Numbers Authority (IANA) port assignments. By combining the source and destination ports, represented as 5 bits each, the total number of lions was set to 10 bits as shown in Figure 2.



	Course Code	No of Students Captured	Protocol	Source Ip	Source Port	Destination IP	Destination Port	Attempts	Attack Name
0	523	20	0	0	80.0	2835	80.0	1	304
1	649	27	0	0	80.0	1925	80.0	1	159
2	73	16	0	0	80.0	2760	80.0	1	304
3	73	6	0	0	80.0	2760	80.0	1	304
4	476	65	0	0	80.0	1187	80.0	1	635
5	472	65	0	0	80.0	1213	80.0	2	304
6	1393	68	0	0	80.0	1675	80.0	1	304
7	785	68	0	0	80.0	426	80.0	1	304
8	785	77	0	0	80.0	74	80.0	1	635
9	785	102	0	0	80.0	74	80.0	1	304

Figure 2: Source and Destination ports in the network (IANA)

The dimensionality of the MLOA's problem search space was derived from the dataset attributes. The total number of columns in the network logs defined a search space of 11 dimensions, with each dimension represented by 4 bits, as shown in Figure 3.

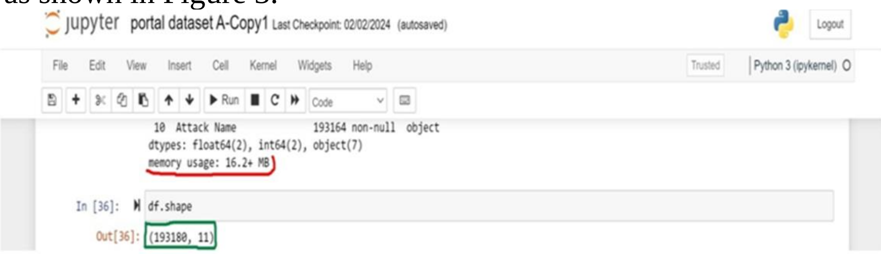


Figure 3: Dimensionality of Network Logs Collected

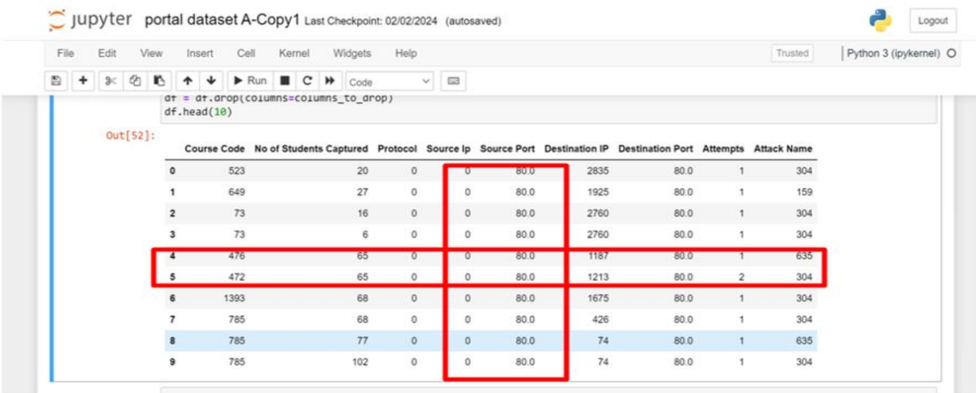
The network logs collected are equal to the total number of columns tagged with the green mark in the search space available within 16.2 MB as tagged with the red mark in Fig.3; the total search space (SS) = 11 (4 bits).

To balance convergence efficiency and computational cost, the maximum number of iterations for the MLOA was calculated using the mode of middle values from the preprocessed dataset, resulting in a maximum iteration value of 40. This was achieved by averaging middle row and column values from the dataset, as shown in Figure 4 and defined by the formula equation 1, which provided balanced parameters for the algorithm. Within the specified search space, the initial population of lions was then created at

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

random, with every lion standing for a possible solution to the optimization problem.

$$M=\sum_{i}^{n} x_i/n \qquad (1)$$



	Course Code	No of Students Captured	Protocol	Source Ip	Source Port	Destination IP	Destination Port	Attempts	Attack Name
0	523	20	0	0	80.0	2835	80.0	1	304
1	649	27	0	0	80.0	1925	80.0	1	159
2	73	16	0	0	80.0	2760	80.0	1	304
3	73	6	0	0	80.0	2760	80.0	1	304
4	476	65	0	0	80.0	1187	80.0	1	635
5	472	65	0	0	80.0	1213	80.0	2	304
6	1393	68	0	0	80.0	1675	80.0	1	304
7	785	68	0	0	80.0	426	80.0	1	304
8	785	77	0	0	80.0	74	80.0	1	635
9	785	102	0	0	80.0	74	80.0	1	304

Figure 4: Preprocessed datasets

MLOA Optimization Model

The fitness of each lion was evaluated using a tailored objective function of the Lion Optimization Algorithm as defined by Rajakumar, (2012) in equations 2 and 3, implemented in Python.

The Objective Function (OF) = arg min $f(x_1, x_2, x_3 \ldots . x_n), x_i \in (x_i^{min}, x_i^{max})\ n \geq 1$ (2) (Rajakumar, 2012)

Equation (2) is an n-variable minimization function in which every result is a variable , $x_i : i = 1,2,3,4, \ldots \ldots n$, might be governed by a particular equality or inequality restriction. The Lion must have a binary structure when n=1.

whereas $n \geq 1$ favors integer-structured lions.

From equation (2) the pride is initiated by generating the initial pride as X^{male} and X^{female} with the structure of $X^{male} = [x_1^{male}, x_2^{male}, x_3^{male}, \ldots . x_L^{male}$ and $X^{female} = [x_1^{female}, x_2^{female}, x_3^{female} \ldots . x_L^{female}$ where L defines the number of lengths of the solution vector to be determined as, $L = \begin{cases} n & ; n>1 \\ m & ; otherwise \end{cases}$

(3) (Rajakumar, 2012). This function ensured optimal threshold adjustments for anomaly detection. The systematic definition of these parameters optimized the MLOA's performance, making it highly effective for identifying anomalies within the portal system network.

Model Initialization and Implementation

Initializing the lion population in the Modified Lion Optimization Algorithm (MLOA) involves creating a set of "lions," each with a random

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

position within the defined search space. The position of every lion corresponds to a particular collection of parameters or thresholds that need to be improved, and each lion represents a possible solution to the optimization problem.

This random distribution ensures diversity within the population, which is critical for effectively exploring the search space. By treating each lion as a candidate solution, the algorithm can evaluate its performance using a fitness function and improve the population to identify optimal solutions. The Python implementation of this initialization process for MLOA and the Loin Optimization (LOA) is detailed in Appendix B. This step is fundamental to the MLOA's success, laying the foundation for the algorithm's optimization process.

Results

The Modified Lion Optimization Algorithm (MLOA) was evaluated with an emphasis on how well it optimized anomaly detection thresholds in a network of portal systems. The following were the evaluation's key targets:

1) Evaluate the MLOA's convergence behavior about the Lion Optimization Algorithm (LOA)
2) Examine how well MLOA detects anomalies in comparison to a Sub-Space Clustering-One-Class Support Vector Machine (SSC-OCSVM).

Evaluation of the Modified Lion Optimization Algorithm (MLOA)

The evaluation of the Modified Lion Optimization Algorithm (MLOA) involved calculating the fitness of each lion using a Python-implemented objective function based on Equations (2) and (3) The goal was to identify the best solution by optimizing the Lion positions within the search space. The resulting convergence behavior, visualized using the Matplotlib library, demonstrated in Fig.5 and Fig.6 for MLOA of best fitness value of approximately 0.37 and LOA of 0.95, respectively, shows that the MLOA significantly outperformed the LOA.

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

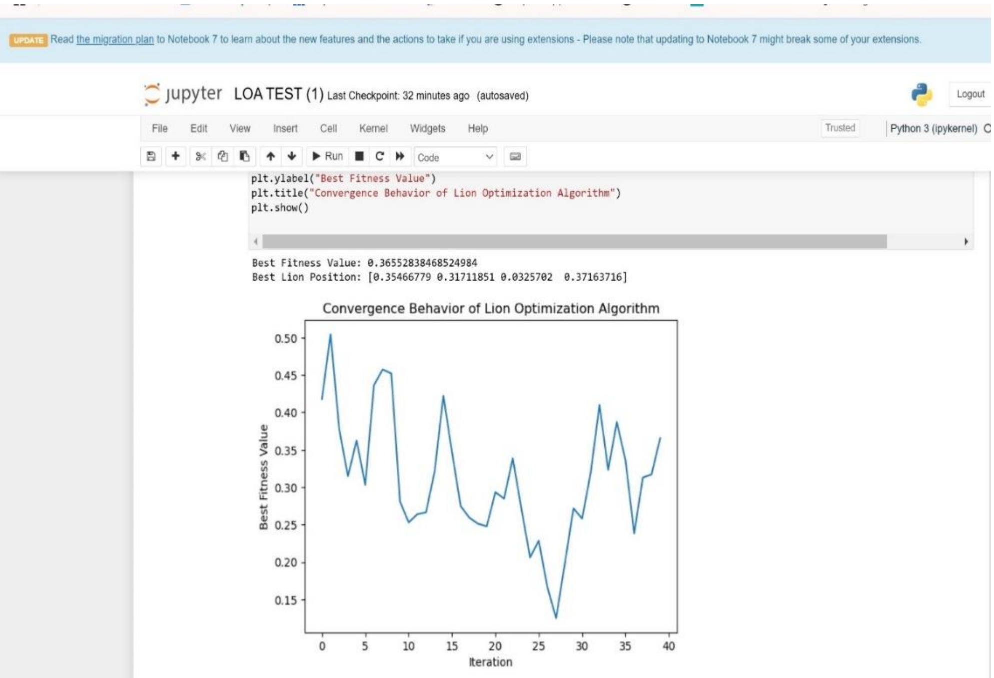


Figure 5: Modified Lion Optimization Algorithm

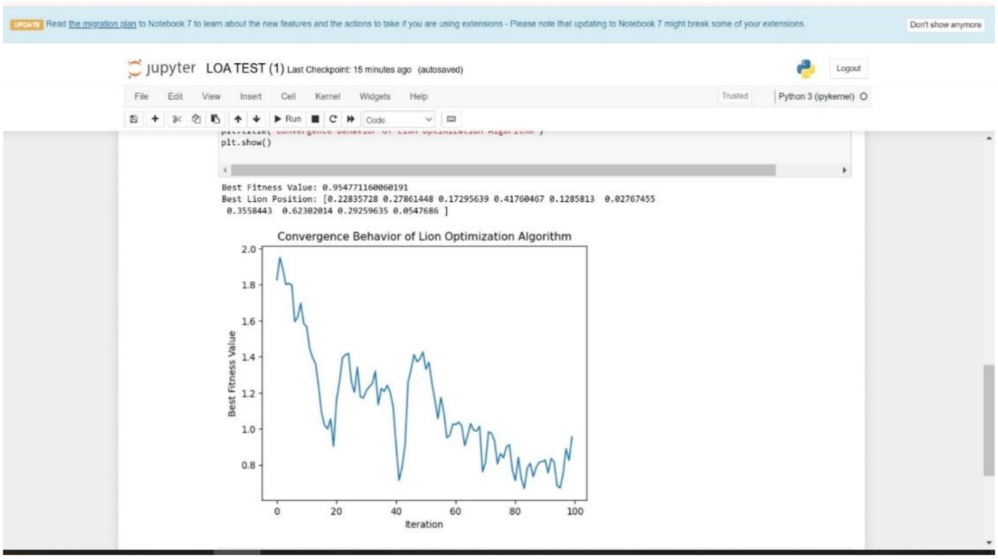


Figure 6: Original Lion Optimization Algorithm

The results indicated the best fitness value of approximately 0.37 and the best Lion position of [0.35, 0.32, 0.03, 0.37]. The results obtained validated the threshold for the anomaly detection model's performance metrics, including accuracy, true positive rate (TPR), precision, F1-score, and AUC-ROC.

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

Comparative Performance of MLOA and SSC-OCSVM

A Sub-Space Clustering-One-Class Support Vector Machine (SSC-OCSVM) by Pu *et al.* (2021), an unsupervised anomaly detection technique that combines the benefit of a One-Class Support Vector Machine (OCSVM) with the attack detection capabilities of Sub-Space Clustering (SSC). The OCSVM, an extension of the Support Vector Machine, is designed for normal or unlabeled data that trains on a single class. The SSC is an extension of the traditional clustering methods like K-means and density-based spatial clustering of applications with noise. The original dataset was divided into smaller sub-spaces using OCSVM, and clusters were created using SSC. Dissimilarity vectors inside each sub-space were used to update the partition. If the dissimilarity value exceeds a predefined threshold, the corresponding data point is flagged as an anomaly. With this hybrid approach, anomalies in unlabeled datasets may be reliably identified by utilizing the strengths of both SSC and OCSVM for complex clustering and anomaly detection, respectively. However, the authors acknowledge the need for developing an effective feature selection method, indicating that the developed method may not fully optimize the feature set used for anomaly detection, potentially leading to suboptimal results.

The performance metrics of the proposed algorithm for detecting anomalies of Increasing Complexity Level are evaluated by implementing and testing the proposed MLOA for anomaly detection in a portal system in Python using the result obtained in Figure 5 to validate the threshold for the anomaly detection model's performance metrics, including accuracy, true positive rate (TPR), precision, F1-score, and AUC-ROC as detailed in Appendix C and Appendix D for the SSC-OCSVM.

The performance of SSC-OCSVM models using the NSL-KDD dataset under conditions similar to those of the proposed MLOA using the UNSW-NB15 dataset was evaluated. While SSC-of SSC-OCSVM and MLOA for Anomaly Detection OCSVM effectively groups normal and anomalous data into sub-spaces, it struggles with feature selection, leading to suboptimal results in complex datasets.

Table 1 shows the comparative performance evaluation metrics for anomaly detection at an increasing Complexity Level of the SSC-OCSVM and the proposed MLOA.

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

Table 1: Comparative Performance Evaluation of Detection Abnormalities of Increasing Complexity Level

Matric	Complexity Level	SSC-OCSVM (NSL-KDD dataset)	Modified LOA (UNSW-NB15 datasets)
Recall (TPR)	Simple Anomalies	0.95	0.97
	Moderate Anomalies	0.90	0.92
	High Anomalies	0.85	0.90
Accuracy	Simple Anomalies	0.96	0.98
	Moderate Anomalies	0.92	0.94
	High Anomalies	0.87	0.91
Precision	Simple Anomalies	0.93	0.96
	Moderate Anomalies	0.88	0.91
	High Anomalies	0.83	0.88
ROC-AUC	Simple Anomalies	0.94	0.97
	Moderate Anomalies	0.90	0.94
	High Anomalies	0.85	0.92
F1 Score	Simple Anomalies	0.94	0.97
	Moderate Anomalies	0.89	0.92
	High Anomalies	0.84	0.90
Training Time (s)	Simple Anomalies	9.65	14.23
	Moderate Anomalies	10.12	12.89
	High Anomalies	11.03	13.67
Prediction Time (s)	Simple Anomalies	0.11	0.30
	Moderate Anomalies	0.21	0.34
	High Anomalies	0.25	0.39

Discussion

The discussion focuses on analyzing the effectiveness, scalability, and trade-offs of the Modified Lion Optimization Algorithm (MLOA) in enhancing anomaly detection within portal systems. The results indicate that MLOA outperforms the Sub-Space Clustering-One-Class Support Vector Machine (SSC-OCSVM) in accurately detecting complex cyber anomalies, particularly in high-dimensional datasets.

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

Result Analysis Summary

This section analyses the fundamental elements that explain MLOA's exceptional performance, including:

1) **Performance**: The Modified LOA outperforms SSC-OCSVM in handling complex anomalies, as shown in Table 1 with high Anomalies ROC-AUC of 0.92 and F1-Score 0.90, SSC-OCSVM performs efficiently on simpler Anomalies ROC-AUC of 0.94 and F1-Score 0.94 but lacks the robustness for more intricate patterns seen in UNSW-NB15.
2) **Scalability**: The Modified LOA is designed to scale better with more extensive and diverse UNSW-NB15 datasets with a TPR of 0.90 for high anomalies compared to 0.85 for the **SSC-OCSVM** using NSL-KDD dataset as shown in Table 1.
3) **Adaptability to Complex Cyberthreats**: In cybersecurity, the ability to detect unknown or evolving attack strategies is critical. SSC-OCSVM is designed for anomaly detection in structured environments but lacks the flexibility to handle new attack patterns. This limitation makes it particularly ineffective against zero-day attacks and adversarial threats. MLOA overcomes this challenge by optimizing multiple fitness functions, ensuring that anomaly detection is based on diverse evaluation criteria rather than a single static decision boundary, adapting to dynamic attack behaviors, allowing the algorithm to update its anomaly thresholds in real-time, and maintaining robustness across varying dataset distributions, ensuring that MLOA can generalize across different network traffic patterns.
4) **Feature Selection:** With the assumption that anomaly-relevant properties remain constant across datasets, SSC-OCSVM clusters feature spaces in a fixed way. Real-world cybersecurity scenarios, where feature relevance fluctuates, fail this assumption. Suboptimal classification is frequently caused by the fixed feature sub-space of SSC-OCSVM, especially in multi-dimensional network logs. This restriction is overcome by MLOA, which incorporates feature selection straight into its optimization procedure. Employing an iterative fitness evaluation, MLOA dynamically finds and ranks significant features, lowering computational complexity and increasing the accuracy of anomaly identification. The ability to generalize is much improved by this feature-adaptive technique, which guarantees improved performance in a variety of network situations.
5) **Tradeoffs**: The Modified Lion Optimization Algorithm (MLOA) introduces a trade-off between detection accuracy and computational efficiency when compared to the Sub-Space Clustering One-Class Support Vector Machine (SSC-OCSVM). While MLOA takes longer

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

for training and prediction, it delivers superior anomaly detection performance, particularly for complex and high-dimensional datasets.

6) **Performance Trade-off: Accuracy vs. Speed**: The Modified LOA achieves a True Positive Rate (TPR) of 0.93 and an F1-score of 0.92, compared to SSC-OCSVM's TPR of 0.85 and F1-score of 0.84. This represents a 9.4% improvement in true positive rate and a 9.5% increase in F1-score, making MLOA more effective in identifying real anomalies while reducing false positives. However, this comes at the cost of higher computational complexity, as MLOA must iteratively optimize thresholds, search space parameters, and anomaly classification rules.
7) **Computational Trade-off: Latency vs. Accuracy in Real-Time Detection**: SSC-OCSVM is faster because it relies on a predefined clustering approach, making it efficient for lightweight real-time detection but less adaptable to complex attack patterns. MLOA requires longer processing time due to its iterative optimization process, but this ensures adaptive learning and dynamic threshold tuning—essential for detecting evolving cyber threats. In real-time applications, latency becomes critical. The choice between MLOA and SSC-OCSVM depends on the use case: If speed is the priority (e.g., real-time intrusion detection systems handling high-volume traffic), SSC-OCSVM may be preferred. If accuracy is the priority (e.g., financial fraud detection, critical infrastructure security), MLOA is a better choice despite its latency.
8) **Adaptive Search and Threshold Optimization method**: The use of static decision boundaries for anomaly classification is a significant drawback of SSC-OCSVM. For high-dimensional, dynamic network traffic, this method is ineffective since it relies on the assumption that anomalies can be found using fixed hyperplane separations. MLOA, on the other hand, uses a search method inspired by nature to continuously modify anomaly detection criteria in response to changing data patterns. MLOA dynamically determines the best decision boundaries according to the lion-inspired exploration-exploitation balance, which improves accuracy and reduces false positive rates. Furthermore, MLOA's fitness function improves with each iteration, adjusting detection criteria in response to real-time feedback. MLOA is more successful in detecting complex attack vectors—such as sophisticated intrusion attempts and zero-day threats—than SSC-OCSVM due to its self-adaptive threshold adjustment.
9) **Faster Convergence and Stability**: One key benefit of MLOA is its speedy convergence to an ideal anomaly detection threshold. The

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

experimental findings show that MLOA attains a best fitness value of 0.38 (Figures 5 and 6). MLOA's quicker convergence is explained by its parallel learning mechanism, which allows it to process large-scale datasets efficiently, self-adaptive selection pressure, efficient search space exploration, which reduces the possibility of becoming stuck in local optima, and the ability to dynamically refine decision-making criteria based on anomaly distributions.

Conclusion

This study introduced a Modified Lion Optimization Algorithm (MLOA) to enhance anomaly detection and mitigation in portal systems. By leveraging advanced optimization techniques, the MLOA demonstrated superior performance compared to the traditional method, SSC-OCSVM, especially in handling complex anomalies. Through systematic parameter optimization, including population size, search space dimensionality, and fitness evaluation, the algorithm achieved higher accuracy, precision, recall, and AUC-ROC scores. Its ability to detect anomalies in real-world datasets like the UNSW-NB15 underscores its practical applicability and potential to enhance the security and resilience of portal systems against evolving cyber threats.

Future work could focus on optimizing the MLOA's computational efficiency by incorporating distributed computing techniques like Apache Spark to handle large-scale datasets in real-time. The algorithm's application could also extend to other domains, such as financial fraud detection and IoT security, where it could identify anomalies in sensor readings or transaction data. Additionally, integrating advanced deep learning methods into the MLOA framework could create hybrid approaches that combine nature-inspired optimization with neural network-based pattern recognition, further improving performance and adaptability.

Conflict of Interest: The authors reported no conflict of interest.

Data Availability: All data are included in the content of the paper.

Funding Statement: The authors did not obtain any funding for this research.

References:

1. Brunda, J.G. and S.. B. (2021) 'An Improved K-Lion Optimization Algorithm With Feature Selection Methods for Text Document Cluster To cite this version : HAL Id : hal-03341649 International Journal of Computer Sciences and Engineering Open Access An Improved K-Lion Optimization Algorith', Vol.6(7), p. 7. Available at:

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

https://doi.org/https://hal.science/hal-03341649.
2. Data, S., Karadayı, Y. and Aydin, M.N. (2020) 'applied sciences A Hybrid Deep Learning Framework for Unsupervised Anomaly Detection in Multivariate', *Applied Science*, 10(15), pp. 1–25. Available at: https://doi.org/Appl. Sci. 2020, 10(15), 5191; https://doi.org/10.3390/app10155191.
3. Katoch, S., Chauhan, S.S. and Kumar, V. (2021) *Katoch2021_Article_AReviewOnGeneticAlgorithmPastP.pdf*, *Multimedia Tools and Applications*. Multimedia Tools and Applications.
4. Konstantina., et al (2021) 'Network traffic anomaly detection via deep learning', *Information (Switzerland)*, 12(5). Available at: https://doi.org/10.3390/info12050215.
5. Mendeley and Elsevier (2017) 'Mendeley Manual for librarians', *The Electronic Library*, 28(1), pp. 1–44. Available at: http://www.emeraldinsight.com/doi/10.1108/02640471011023388%0Ahttps://www.elsevier.com/__data/assets/pdf_file/0011/117992/Mendeley-Manual-for-Librarians_2017.pdf.
6. Nour Moustafa, J.S. (2015) 'UNSW-NB15 SOURCE FILES.pdf', in *The UNSW-NB15 SOURCE FILES*. Australia: Australian centre for cyber security (ACCS), pp. 1–2. Available at: https://doi.org/https://research.unsw.edu.au/projects/unsw-nb15-dataset.
7. Paganini, P. (2023) *The University of Manchester suffered a cyber attack and suspects a data breach*, *Security Affairs*. Available at: https://i0.wp.com/securityaffairs.com/wp-content/uploads/2023/06/University-of-Manchester.png?ssl=1 (Accessed: 15 January 2025).
8. Pu, G. *et al.* (2021) 'A Hybrid Unsupervised Clustering-Based Anomaly Detection Method', 26(1007–0214), pp. 146–153. Available at: https://doi.org/10 . 26599 /TST. 2019 . 9010051.
9. Rajakumar, B.R. (2012) 'The Lion's Algorithm: A New Nature-Inspired Search Algorithm', *Procedia Technology*, 6, pp. 126–135. Available at: https://doi.org/10.1016/j.protcy.2012.10.016.
10. RRajakumar, B. (2012) 'LION'S ALGORITHM', *Procedia Technology*, 6((2012)), pp. 126–135. Available at: https://doi.org/https://doi.org/10.1016/j.protcy.2012.10.016.
11. Thomas-Krenn.AG, S.-S. (2018) *OPNsense*. thomas-krenn.com. Available at: https://www.thomas-krenn.com/redx/tools/mb_download.php/ct.X3V5Wg/mid.y1de5b073d8372315/ebook_OPNsense_Thomas-Krenn_max_it_V2_ENG.pdf.
12. Ukagwu, T.J. and L. (2023) 'Almost 13 million cyber-attacks recorded

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

during polls –FG', *Punch*, 15 March, pp. 1–3. Available at: https://cdn.punchng.com/wp-content/uploads/2023/02/16205355/ISA-PANTAMI.jpg.
13. Vanneschi, L. and Silva, S. (2023) 'Particle Swarm Optimization', *Natural Computing Series*, pp. 105–111. Available at: https://doi.org/10.1007/978-3-031-17922-8_4.

Appendix A

Python Code: Preprocessing LASUED Network Logs & UNSW-NB15

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Load datasets
def load_datasets(lasued_file, unsw_file):
    """Loads network logs (LASUED) and UNSW-NB15 dataset."""
    lasued_data = pd.read_csv(lasued_file)  # LASUED Network Logs (CSV Format)
    unsw_data = pd.read_csv(unsw_file)  # UNSW-NB15 Dataset
    return lasued_data, unsw_data

# Handle missing values
def handle_missing_values(df):
    """Imputes missing values using the column mean."""
    imputer = SimpleImputer(strategy="mean")
    df_numeric = df.select_dtypes(include=[np.number])  # Select numerical columns
    df[df_numeric.columns] = imputer.fit_transform(df_numeric)
    return df

# Encode categorical features
def encode_categorical_features(df):
    """Encodes categorical variables into numeric format using Label Encoding."""
    label_encoders = {}
    for col in df.select_dtypes(include=["object"]).columns:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le
    return df, label_encoders

# Scale numerical features
def scale_features(df):
    """Standardizes numerical features using StandardScaler."""
    scaler = StandardScaler()
    df_numeric = df.select_dtypes(include=[np.number])
    df[df_numeric.columns] = scaler.fit_transform(df_numeric)
    return df, scaler
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

```
# Preprocess both datasets
def preprocess_data(lasued_file, unsw_file, save_cleaned=True):
    """Preprocesses network log datasets by handling missing values, encoding, and scaling."""
    lasued_data, unsw_data = load_datasets(lasued_file, unsw_file)

    # Handle missing values
    lasued_data = handle_missing_values(lasued_data)
    unsw_data = handle_missing_values(unsw_data)

    # Encode categorical features
    lasued_data, lasued_encoders = encode_categorical_features(lasued_data)
    unsw_data, unsw_encoders = encode_categorical_features(unsw_data)

    # Scale numerical features
    lasued_data, lasued_scaler = scale_features(lasued_data)
    unsw_data, unsw_scaler = scale_features(unsw_data)

    # Save cleaned data
    if save_cleaned:
        lasued_data.to_csv("LASUED_Cleaned.csv", index=False)
        unsw_data.to_csv("UNSW-NB15_Cleaned.csv", index=False)
        print("Preprocessed datasets saved as 'LASUED_Cleaned.csv' and 'UNSW-NB15_Cleaned.csv'.")

    return lasued_data, unsw_data

# Run preprocessing
if __name__ == "__main__":
    lasued_file = "LASUED_Network_Logs.csv"  # Update with the actual file path
    unsw_file = "UNSW-NB15.csv"  # Update with the actual file path
    lasued_cleaned, unsw_cleaned = preprocess_data(lasued_file, unsw_file)
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

Appendix B
Modified LOA

```
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)  # Ensures the same random values are generated every time

# Step 1: Initializing Parameters
population_size = 10  # I. Define the population size of the lion
dimensionality = 4  # II. Define dimensionality
max_iterations = 40  # III. Define maximum iterations

# IV. Initialize Lion Population
lion_positions = np.random.rand(population_size, dimensionality)  # Initial population of lions with random positions

# Step 2: Evaluate the Fitness
def evaluate_fitness(positions):
    """Implement your objective function here"""
    return np.sum(positions**2)

fitness_values = np.zeros(population_size)
for i in range(population_size):
    fitness_values[i] = evaluate_fitness(lion_positions[i])

# Step 3: Update Lion Positions (Lion Optimization Algorithm)
def update_positions(positions, fitness_values):
    np.random.seed(42)  # Set seed again before random operations for consistency
    alpha = 0.1  # Alpha parameter for LOA
    beta = 0.1   # Beta parameter for LOA

    # Sort positions based on fitness values
    sorted_indices = np.argsort(fitness_values)
    sorted_positions = positions[sorted_indices]

    # Update positions based on LOA rules
    for i in range(1, population_size):
        positions[i] = positions[i] + alpha * (sorted_positions[i-1] - positions[i]) + \
                beta * np.random.uniform(-1, 1, size=dimensionality)

    return positions

# Step 4: Check Convergence
def check_convergence(iteration, max_iterations, fitness_values, threshold=1e-6):
    return iteration >= max_iterations or np.max(np.abs(np.diff(fitness_values))) < threshold

# Step 5: Results and Visualization
best_fitness_history = []
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

```
for iteration in range(max_iterations):
    # Update positions
    lion_positions = update_positions(lion_positions, fitness_values)

    # Evaluate fitness
    for i in range(population_size):
        fitness_values[i] = evaluate_fitness(lion_positions[i])

    # Update best fitness history
    best_fitness_history.append(np.min(fitness_values))

    # Check for convergence
    if check_convergence(iteration, max_iterations, fitness_values):
        break

# Display or save results
print("Best Fitness Value:", np.min(fitness_values))
print("Best Lion Position:", lion_positions[np.argmin(fitness_values)])

# Visualization of convergence behavior
plt.plot(best_fitness_history)
plt.xlabel("Iteration")
plt.ylabel("Best Fitness Value")
plt.title("Convergence Behavior of Lion Optimization Algorithm")
plt.show()
```

Original LOA

```
import numpy as np
import matplotlib.pyplot as plt
# Set random seed for reproducibility
np.random.seed(42)  # Ensures the same random values are generated every time

# Step 1: Initializing Parameters
population_size = 50  # I. Define the population size of the lion
dimensionality = 10 # II. Define dimensionality
max_iterations = 100   # III. Define maximum iterations

# IV. Initialize Lion Population
lion_positions = np.random.rand(population_size, dimensionality)  # Initial population of lions with random positions

# Step 2: Evaluate the Fitness
def evaluate_fitness(positions):
    # Implement your objective function here
    # Example: fitness = sum(positions**2)
    return np.sum(positions**2)

fitness_values = np.zeros(population_size)
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

```
for i in range(population_size):
    fitness_values[i] = evaluate_fitness(lion_positions[i])

# Step 3: Update Lion Positions (Lion Optimization Algorithm)
def update_positions(positions, fitness_values):
    alpha = 0.1  # Alpha parameter for LOA
    beta = 0.1   # Beta parameter for LOA

    # Sort positions based on fitness values
    sorted_indices = np.argsort(fitness_values)
    sorted_positions = positions[sorted_indices]

    # Update positions based on LOA rules
    for i in range(1, population_size):
        positions[i] = positions[i] + alpha * (sorted_positions[i-1] - positions[i]) + beta *
np.random.uniform(-1, 1, size=dimensionality)

    return positions
# Step 4: Check Convergence
def check_convergence(iteration, max_iterations, fitness_values, threshold=1e-6):
    return iteration >= max_iterations or np.max(np.abs(np.diff(fitness_values))) < threshold

# Step 5: Results and Visualization
best_fitness_history = []

for iteration in range(max_iterations):
    # Update positions
    lion_positions = update_positions(lion_positions, fitness_values)

    # Evaluate fitness
    for i in range(population_size):
        fitness_values[i] = evaluate_fitness(lion_positions[i])

    # Update best fitness history
    best_fitness_history.append(np.min(fitness_values))

    # Check for convergence
    if check_convergence(iteration, max_iterations, fitness_values):
        break

# Display or save results
print("Best Fitness Value:", np.min(fitness_values))
print("Best Lion Position:", lion_positions[np.argmin(fitness_values)])

# Visualization of convergence behavior
plt.plot(best_fitness_history)
plt.xlabel("Iteration")
plt.ylabel("Best Fitness Value")
plt.title("Convergence Behavior of Lion Optimization Algorithm")
plt.show()
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

Appendix C
Python Code for Validation of Threshold for the Anomaly Detection

Step 1
import pandas as pd
import numpy as np
 # Set random seed for reproducibility
np.random.seed(42) # Ensures the same random values are generated every time

from sklearn.preprocessing import StandardScaler
from sklearn.svm import OneClassSVM
Step 2: Load UNSW-NB15 and network logs.
unsw_data = pd.read_csv('UNSW-NB15.csv')
log_data = pd.read_csv('network-logs.csv')
Step 3: Preprocessing of the data
scaler = StandardScaler()
unsw_data_scaled = scaler.fit_transform(unsw_data)
log_data_scaled = scaler.fit.transform(log_data)
Step 3: Combine the Datasets, merging the datasets to create a more diverse dataset with varied complexity. This is done by concatenating the data from both sources and strategically blending the records.
Combine datasets
combined_data=pd.concat([pd.DataFrame(unsw_data_scaled), pd.DataFrame(log_data_scaled)], ignore_index=True)
Step 4: Training the OCSVM Model on Normal Data, separating normal data from the combined dataset, and using it to train the One-Class SVM (OCSVM) model. This will allow the model to learn a baseline of "normal" behavior based on the structure of the UNSW-NB15 and network log data.
from sklearn.svm import OneClassSVM
Filter normal data for training
normal_data = combined_data[labels == 1] Assuming 1 represents normal
Initialize and train the OCSVM model
ocsvm = OneClassSVM(kernel='rbf', gamma=0.38, nu=0.42)
ocsvm.fit(normal_data)
Step 5: Running the OCSVM Model's ability to recognize abnormalities of increasing degrees of Complexity
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import numpy as np
from sklearn.svm import OneClassSVM
np.random.seed(0)
normal_data = np.random.normal(0, 1, (200, 5))   200 samples, 5 features
simple_anomalies = np.random.normal(3, 1, (20, 5))   Simple anomalies
moderate_anomalies = np.random.normal(5, 2, (20, 5))   Moderate complexity anomalies
*high_complexity_anomalies = np.array([np.sin(0.1 * np.arange(5)) + 7 for _ in range(20)])*
High complexity anomalies
ocsvm = OneClassSVM(kernel='rbf', gamma=0.38, nu=0.42)
ocsvm.fit(normal_data)
def evaluate_model_with_auc(model, test_data, true_label=-1):
predictions = model.predict(test_data)

predictions = np.where(predictions == 1, 1, true_label) # Map to 1 for normal, -1 for anomalies
ground_truth = np.full(test_data.shape[0], true_label)

Calculate metrics
accuracy = accuracy_score(ground_truth, predictions)
precision = precision_score(ground_truth, predictions, pos_label=true_label)
recall = recall_score(ground_truth, predictions, pos_label=true_label)
f1 = f1_score(ground_truth, predictions, pos_label=true_label)
Binary ground truth and predictions for AUC
binary_ground_truth = (ground_truth == true_label).astype(int)
binary_predictions = (predictions == true_label).astype(int)
auc = roc_auc_score(binary_ground_truth, binary_predictions)
return accuracy, precision, recall, f1, auc
Complexity levels
complexity_levels = {"Simple Anomalies": simple_anomalies, "Moderate Complexity Anomalies": moderate_anomalies, "High Complexity Anomalies": high_complexity_anomalies}
Evaluate each complexity level
for level, data in complexity_levels.items():
accuracy, precision, recall, f1, auc = evaluate_model_with_auc(ocsvm, data)
print(f"{level} - Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}, AUC: {auc:.2f}"). The output of the result is shown in Table 4.5

Appendix D

Python code for Implementing SSC-OCSVM using the NSL-KDD dataset

```
import numpy as np
import pandas as pd
# Set random seed for reproducibility
np.random.seed(42)  # Ensures the same random values are generated every time


from sklearn.model_selection import train_test_split
from sklearn.svm import OneClassSVM
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_score, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Step 1: Load the NSL-KDD Dataset
def load_nsl_kdd_data():
    # Replace with the actual path to your dataset
    train_file = "KDDTrain+.txt"
    test_file = "KDDTest+.txt"

    # Load the dataset
    column_names = [
        "duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land", "wrong_fragment",
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

```
        "urgent", "hot", "num_failed_logins", "logged_in", "num_compromised", "root_shell", "su_attempted",
        "num_root", "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds",
        "is_host_login", "is_guest_login", "count", "srv_count", "serror_rate", "srv_serror_rate",
        "rerror_rate", "srv_rerror_rate", "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate",
        "dst_host_count", "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
        "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
        "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "class"
    ]

    train_data = pd.read_csv(train_file, header=None, names=column_names)
    test_data = pd.read_csv(test_file, header=None, names=column_names)

    return train_data, test_data

# Step 2: Preprocess the Dataset
def preprocess_data(data):
    # Convert categorical features to numeric
    categorical_features = ["protocol_type", "service", "flag"]
    for feature in categorical_features:
        encoder = LabelEncoder()
        data[feature] = encoder.fit_transform(data[feature])

    # Normalize the dataset
    scaler = StandardScaler()
    X = data.drop(columns=["class"])
    X = scaler.fit_transform(X)

    # Convert labels: normal = +1, anomaly = -1
    y = data["class"].apply(lambda x: 1 if x == "normal" else -1).values

    return X, y

# Step 3: Train SSC-OCSVM Model
def train_ssc_ocsvm(X_train, X_test, y_test):
    # Sub-space clustering: Example with top 10 features based on variance
    feature_variance = np.var(X_train, axis=0)
    top_features_indices = np.argsort(feature_variance)[-10:]  # Select top 10 features
    X_train_subspace = X_train[:, top_features_indices]
    X_test_subspace = X_test[:, top_features_indices]

    # Train One-Class SVM
    ocsvm = OneClassSVM(kernel="rbf", nu=0.1, gamma="scale")
    ocsvm.fit(X_train_subspace)
```

 ISSN: 1857-7881 (Print) e - ISSN 1857-7431

```
    # Predict on the test set
    y_pred = ocsvm.predict(X_test_subspace)

    return y_pred

# Step 4: Evaluate Model Performance
def evaluate_model(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred, pos_label=1)
    precision = precision_score(y_test, y_pred, pos_label=1)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, pos_label=1)

    print(f"Accuracy: {accuracy:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"ROC-AUC: {roc_auc:.2f}")
    print(f"F1-Score: {f1:.2f}")

    return accuracy, recall, precision, roc_auc, f1

# Main Function
if __name__ == "__main__":
    # Load and preprocess the data
    train_data, test_data = load_nsl_kdd_data()
    X_train, y_train = preprocess_data(train_data)
    X_test, y_test = preprocess_data(test_data)

    # Train SSC-OCSVM and generate predictions
    y_pred = train_ssc_ocsvm(X_train, X_test, y_test)

    # Evaluate the model
    evaluate_model(y_test, y_pred)
```