National University of Sciences & Technology (NUST)

School of Electrical Engineering and Computer Science

(SEECS)

IS-823: Computer Forensics (3+0) – (MSIS)

Project	
M. Faaz Qadeer	**495705**
Zahid Ullah Khan	**495181**
Shah Fahad	**493130**

Paper Title:

"A new network forensic framework based on deep learning for Internet of Things networks: A particle deep framework"

A new network forensic framework based on deep learning for Internet of Things networks: A particle deep framework

1. Introduction:

This report outlines the workflow for optimizing the hyperparameters of Multi-Layer Perceptron (MLP) models using Particle Swarm Optimization (PSO) applied to IoT cybersecurity datasets. The optimization process aims to enhance the performance of MLP classifiers, which are widely used for detecting various cybersecurity threats, by fine-tuning key hyperparameters such as hidden layer size, learning rate, and regularization strength. The workflow includes steps for data preprocessing, feature extraction, handling class imbalance, and evaluating the model's performance using various metrics.

2. Abstract:

This report presents a comprehensive approach to optimizing MLP hyperparameters using Particle Swarm Optimization (PSO) on IoT cybersecurity datasets. Initially, data preprocessing steps were implemented to clean and balance the dataset, including feature extraction and conversion. PSO was then applied to optimize the hyperparameters of the MLP model, including hidden layer size, learning rate, and regularization strength. The optimization process was evaluated across multiple sampling strategies (original, oversampled, and undersampled data), and the results demonstrated the robustness of PSO in fine-tuning the model's parameters. The final model achieved high performance with the best results obtained using the original dataset (AUC: 0.9938). The study highlights the potential of PSO for improving the classification accuracy and generalization ability of MLP models in cybersecurity applications.

3. **Used Libraries**:

The workflow begins by importing essential Python libraries required for data processing, modeling, and optimization. These include
- numpy for numerical operations and array handling,
- pandas for data manipulation and analysis,
- matplotlib.pyplot for data visualization,
- scikit-learn modules such as MLPClassifier for neural network modeling,
  train_test_split for data splitting,
  roc_auc_score for evaluation,
  classification_report and confusion_matrix for metrics,
  StandardScaler for normalization, and
  SimpleImputer for handling missing values.

4. Data Preprocessing and Feature Engineering:

We started by inspecting the dataset categories to identify all unique categories in the category column across all CSV files in the IOT Dataset. We iterated through each file, counted occurrences of each category, and observed the results. Next, we segregated the data by category, extracting and saving all rows with a specific category (e.g., 'DoS') into new separate CSV files. Following that, we segregated the data by subcategory, extracting and saving rows with a specific subcategory (e.g., 'UDP') into new separate CSV files. We then performed feature extraction, where we extracted specific columns from each file for further analysis. We selected a predefined list of columns. Finally, we identified unique values in each selected column across all files. We iterated through the columns and files, counting the unique values and printing the results.

i. Raw Dataset:

Category	Subcategory	Count
1. DDoS	Total	38,532,480
	HTTP	19,771
	TCP	19,547,603
	UDP	18,965,106
2. DoS	Total	33,005,194
	HTTP	29,706
	TCP	12,315,997
	UDP	20,659,491
3. Normal	Total	9,543
4. Reconnaissance	Total	1,821,639
	OS Fingerprint	358,275
	Service Scan	1,463,364
5. Theft	Total	1,587
	Data Exfiltration	118
	Keylogging	1,469

Table 1. Raw Dataset

ii. Data Cleaning:

Dropped rows with missing values, especially in critical columns. Dropped duplicate rows from each file. Removed rows with unwanted or very few instances protocol or state values (e.g., 'rarp', 'igmp', 'CLO', etc.).

iii. Data Conversion

- String to Numeric: Converted protocol and state columns from strings to integers using mapping dictionaries.
- IP Address Conversion: Converted IP addresses to 32-bit integers.
- Hexadecimal to Decimal: Converted hexadecimal port numbers to decimal.
- Labeling: Assigned numeric labels to each attack.

proto	Int value	state	Int values
tcp	1	ACC	1
udp	2	CON	2
icmp	3	ECO	3
arp	4	FIN	4
ipv6-icmp	5	INT	5
		MAS	6
		NRS	7
		REQ	8
		RST	9
		TST	10
		URP	11

Table 2. Conversion Mapping

Attack	Label
DoS via HTTP	0
DDoS via TCP	1
DDoS via HTTP	2
DoS via TCP	3
DoS via UDP	4
OS Fingerprint	5
Service Scan	6
Data Exfiltration	7
Keylogging	8
Normal	9
DDoS UDP	10

Table 3. Labelling

iv. Feature Selection

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. As a classifier, it is highly effective for handling large datasets with many features and can also be used for feature selection. In feature selection, Random Forest works by evaluating the importance of each feature based on how well it improves the accuracy of the model. Features that contribute the most to reducing the impurity (or error) across the trees are considered more important. By ranking the features according to their importance, Random Forest helps in selecting the most relevant features, which can improve model performance, reduce overfitting, and lower computation time.

Rank	Feature	Importance
1	dur	27.08%
2	proto	14.95%
3	state	12.55%
4	sbytes	7.81%
5	bytes	7.37%
6	spkts	7.35%
7	mean	6.64%
8	sport	5.95%
9	pkts	5.55%
10	stddev	2.31%
11	rate	1.33%
12	dport	1.07%
13	saddr	0.01%
14	daddr	0.01%

Table 4. Features' importance

5. Balancing the Dataset

To ensures each class has a similar number of samples. We used approach so that we have samples up to a maximum 50k number of rows per class and shuffles the data.

Label	Total	Train	Validation
4	50,000	30,000	10,000
5	50,000	30,000	10,000
6	50,000	30,000	10,000
10	50,000	30,000	10,000
1	50,000	30,000	10,000
3	50,000	30,000	10,000
0	29,705	17,823	5,941

Label	Total	Train	Validation
9	9,251	5,551	1,850
2	3,414	2,048	683
8	1,469	881	294
7	118	70	24

Table 5. Initial Balanced dataset

i. Advanced Balancing Techniques

SMOTE (Synthetic Minority Over-sampling Technique) is a technique used to address class imbalance by generating synthetic samples for the minority class, rather than simply duplicating existing ones. This helps to balance the class distribution and improve model performance, especially for imbalanced datasets. On the other hand, **Tomek Links** is a technique used for under-sampling, specifically designed to remove ambiguous or borderline examples by identifying pairs of instances from different classes that are closest to each other. While SMOTE increases the number of minority class samples, Tomek Links reduces the number of majority class samples by eliminating potentially noisy examples. Both methods aim to enhance model training, but SMOTE focuses on balancing through over-sampling, while Tomek Links focuses on cleaning the dataset via under-sampling.

Label	Total	Train	Val
6	47,979	28,787	9,596
5	47,963	28,777	9,593
1	45,397	27,238	9,080
3	45,397	27,239	9,079
10	42,280	25,368	8,456
4	41,932	25,160	8,386
0	29,290	17,574	5,858
9	9,221	5,533	1,844
2	3,024	1,814	605
8	1,454	872	291
7	118	71	23
Total	314,055	188,443	62,816

Table 6. Balanced Dataset after SMOTE (Over-Sampling)

Label	Total	train	val
0	10,000	6,000	2,000
1	10,000	6,000	2,000
2	10,000	6,000	2,000
3	10,000	6,000	2,000
4	10,000	6,000	2,000
5	10,000	6,000	2,000
6	10,000	6,000	2,000
7	10,000	6,000	2,000
8	10,000	6,000	2,000
9	10,000	6,000	2,000
10	10,000	6,000	2,000

Table 7. Balanced Dataset after Tomek Links (Under-Sampling)

6. Data Loading and Preprocessing

Several functions were defined to load and preprocess IoT cybersecurity datasets from CSV files. We began by loading the data using pandas, either from a single file or multiple files in a directory. Next, we performed feature and target separation, extracting features (X) from the target variable (y). These preprocessing steps ensured that the dataset was clean, consistent, and ready for machine learning. In the dataset exploration and splitting phase, we printed the shape and column names of the loaded data, displayed the distribution of the target variable to understand class balance, and normalized the features using StandardScaler to ensure all features contributed equally to the model. Finally, we split the data into training, validation, and test sets, typically using stratified sampling to preserve the class distribution.

7. Particle Swarm Optimization (PSO) Implementation

Particle Swarm Optimization (PSO) is a nature-inspired optimization algorithm introduced in 1995 by Kennedy and Eberhart. It mimics the social behaviour of bird flocking or fish schooling, where individual particles (candidate solutions) move through a search space by updating their positions based on their own best experience and that of the swarm. This movement is governed by inertia, a cognitive component (self-learning), and a social component (learning from others), allowing the swarm to balance exploration and exploitation efficiently.

PSO is widely applied in various domains requiring optimization. In machine learning and AI, it is used for hyperparameter tuning and feature selection. Engineering fields use it for structural design and control system optimization, while operations research applies it in scheduling and resource allocation. It’s also useful in image processing tasks like segmentation and edge detection, and in cybersecurity for optimizing intrusion detection systems and classification rule tuning.

The performance and outcome of Particle Swarm Optimization (PSO) are influenced by several key parameters. These include swarm size and number of iterations, which affect exploration depth and computational cost. In this PSO implementation, optimization is performed in a 3-dimensional space representing the hidden layer size, learning rate, and regularization parameter of an MLP model. The search space is bounded for each parameter: hidden layer size ranges from 10 to 200, and both learning rate and regularization parameter range from 0.0001 to 0.1. The swarm consists of 20 particles exploring the space over 12 iterations. These settings aim to balance exploration and convergence efficiency, enabling the algorithm to fine-tune hyperparameters within a defined, meaningful range while keeping computational cost manageable.

i. PSO Fitness Evaluation

In the PSO-based fitness evaluation, each particle represents a unique set of MLP hyperparameters—hidden layer size, learning rate, and regularization strength. For every iteration, these parameters are used to train an MLP on the training set and evaluate its performance on a validation set using Area Under the ROC Curve (AUC) as the fitness metric. This ensures effective optimization for both binary and multi-class classification problems. Any configuration that fails during training is assigned a low fitness score to prevent its influence on swarm behaviour, effectively guiding the optimization toward high-performing hyperparameter sets.

The PSO optimization process works by continuously updating each particle's position and velocity based on both its personal best and the swarm's global best solutions. At each iteration, particles are evaluated using a fitness function (in this case, AUC), and updates are made to track improvements. The best fitness value is recorded at every step to monitor convergence, and over several iterations, the swarm gradually moves toward an optimal set of hyperparameters.

The hyperparameter optimization execution is managed through a dedicated function that defines the search space (bounds) and sets the PSO configuration, including particle count and number of iterations. It runs the entire optimization loop, displays progress, and ultimately returns the best-performing hyperparameter set. This modular function makes the optimization process efficient, reusable, and easy to integrate into various model training workflows.

PSO was applied to three different data sampling strategies: original imbalanced data, oversampled data (equal class distribution), and under-sampled data (reduced class imbalance). The best AUCs achieved were 0.9938 (imbalanced/original), 0.9918 (oversampled), and 0.9909 (under-sampled), showing consistently high performance. The original imbalanced dataset yielded the best overall performance (weighted F1-score = 0.89), while oversampling helped boost performance on rare classes like labels 7 and 8 (F1 = 0.98). The under-sampled setup achieved balanced but slightly lower metrics overall. These results demonstrate PSO's ability to robustly adapt hyperparameters under various class distributions, with the model performing best when trained on the full, imbalanced dataset while oversampling proved most beneficial for rare-class detection.

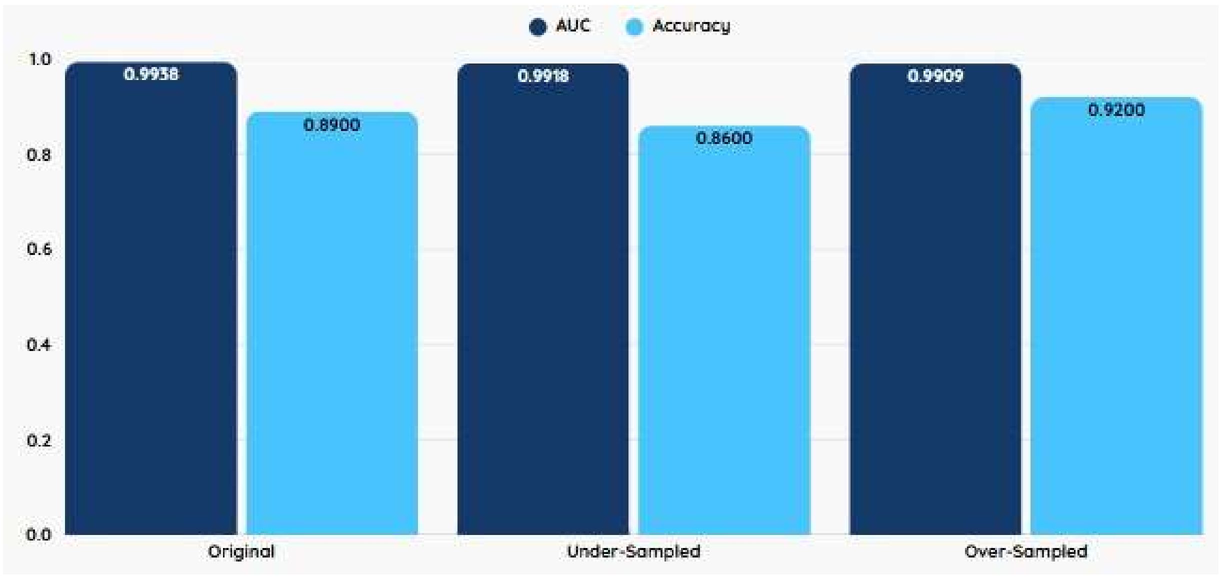

Figure 1. AUC, Accuracy and F1 score for different strategies

Comparing these metrics across different sampling strategies using visual graphs helps highlight the trade-offs between overall performance and fairness across classes. The AUC and accuracy chart illustrate how well the model ranks predictions and its general correctness, while the F1 score chart shows how balanced the model's performance is across all classes—especially important in imbalanced datasets. Evaluating both allows us to assess whether the model is biased toward frequent classes (high accuracy, low macro F1) or more balanced (higher macro F1, possibly lower accuracy). This comparison is crucial in domains like cybersecurity or medical diagnostics, where minority classes can represent critical anomalies, and balanced detection is often more valuable than just overall correctness.

8. Final Model Training and Evaluation

A Multi-Layer Perceptron (MLP) is a type of feedforward artificial neural network that consists of at least three layers: an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to every neuron in the next layer through weighted connections. The MLP uses activation functions (like ReLU or sigmoid) to introduce non-linearity, enabling it to learn complex patterns. During training, it adjusts its weights through a process called backpropagation using optimization algorithms like stochastic gradient descent.

MLPs are particularly well-suited for supervised learning tasks where the goal is to map input data to a known output. They are widely used for classification (e.g., image or text classification), regression, and function approximation. Their ability to capture non-linear relationships in data makes them effective in scenarios where simpler models fail to capture the underlying structure.

One of the key advantages of MLPs is their versatility; they can model both linear and non-linear relationships. They are also relatively easy to implement and can scale to large datasets. With

enough hidden units and layers, MLPs can approximate any continuous function, making them highly flexible. Additionally, they can learn from noisy and complex data without requiring manual feature engineering
In cybersecurity, MLPs are valuable tools for detecting and classifying threats such as malware, phishing, and network intrusions. They can be trained on labelled datasets to distinguish between normal and suspicious activity, often outperforming traditional rule-based systems. Their ability to adapt to evolving patterns makes them useful in dynamic environments where new attack vectors frequently emerge, thereby enhancing the robustness of intrusion detection systems (IDS) and other security mechanisms.

With the optimized hyperparameters obtained from PSO, a final MLP Classifier is trained using the entire training dataset to leverage all available data for learning. This trained model is then evaluated on a separate test set to assess its generalization performance. Key evaluation metrics include the final AUC score, which measures the model's ability to distinguish between classes, a detailed classification report showing precision, recall, and F1-score for each class, and a confusion matrix that provides insight into the distribution of correct and incorrect predictions across all classes.

9. Our findings

i. Original data

Metric by PSO	Value
Best AUC	0.9938
Hidden Layer Neurons	109
Learning Rate	0.004385
Regularization Alpha	0.000100

Table 8. PSO outputs (Original Data)

Label	Precision	Recall	F1-Score	Support
0	0.98	0.98	0.98	5,941
1	0.94	0.87	0.90	10,000
3	0.87	0.94	0.91	10,000
4	0.86	0.84	0.85	10,000
5	0.83	0.97	0.89	10,000
6	0.97	0.79	0.87	10,000
7	0.60	0.12	0.21	24
8	0.89	0.90	0.90	294
9	0.90	0.99	0.94	1,850
10	0.85	0.85	0.85	10,000
Overall	0.89	0.89	0.89	68,109
Macro Avg	0.87	0.83	0.83	—
Weighted Avg	0.89	0.89	0.89	—

Table 9. MLP Classification Report (Original Data)

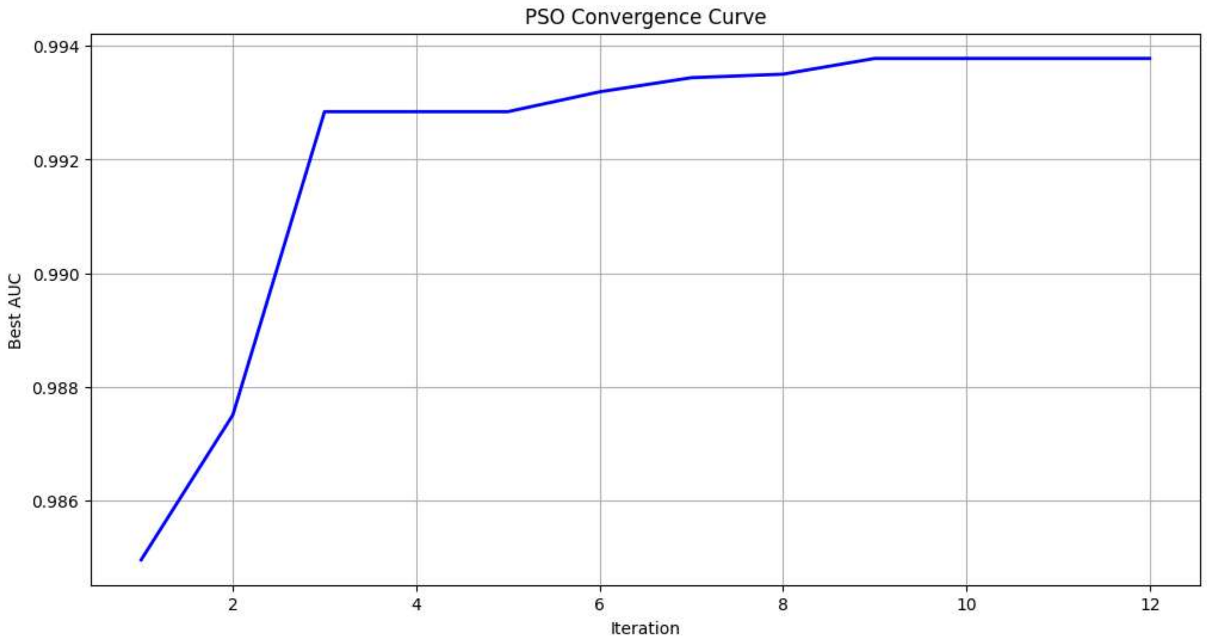

Figure 2. PSO Convergence Curve (Original Data)

True \ Pred	0	1	2	3	4	5	6	7	8	9	10
0	5845	48	41	0	2	0	0	1	4	0	0
1	106	8684	1205	0	1	1	0	0	2	1	0
2	26	530	9442	1	0	0	0	0	1	0	0
3	0	0	0	8424	0	0	0	0	95	1481	0
4	6	3	43	0	9709	228	0	9	1	1	0
5	8	3	61	8	2022	7872	1	7	18	0	0
6	0	1	0	0	3	2	3	14	1	0	0
7	1	0	0	0	19	4	1	266	3	0	0
8	0	0	0	1	7	8	0	1	1827	6	0
9	0	0	0	1402	0	0	0	0	84	8514	0

Table 10. MLP Confusion Matrix (Original Data)

ii. Under Sampled Data

Metric by PSO	Value
Best AUC	0.9918
Hidden Layer Neurons	200
Learning Rate	0.006454
Regularization Alpha	0.000100

Table 11. PSO outputs (Under-Sampling)

Label	Precision	Recall	F1-Score	Support
0	0.90	0.78	0.83	2000
1	0.83	0.81	0.82	2000
2	0.88	0.95	0.92	2000
3	0.73	0.79	0.76	2000
4	0.74	0.78	0.76	2000
5	0.81	0.92	0.86	2000
6	0.93	0.80	0.86	2000
7	0.98	0.98	0.98	2000
8	0.98	0.96	0.97	2000
9	0.95	0.99	0.97	2000
10	0.77	0.72	0.74	2000
Accuracy			0.86	22000
Macro avg	0.86	0.86	0.86	22000
Weighted avg	0.86	0.86	0.86	22000

Table 12. MLP Classification Report (Under-Sampling)

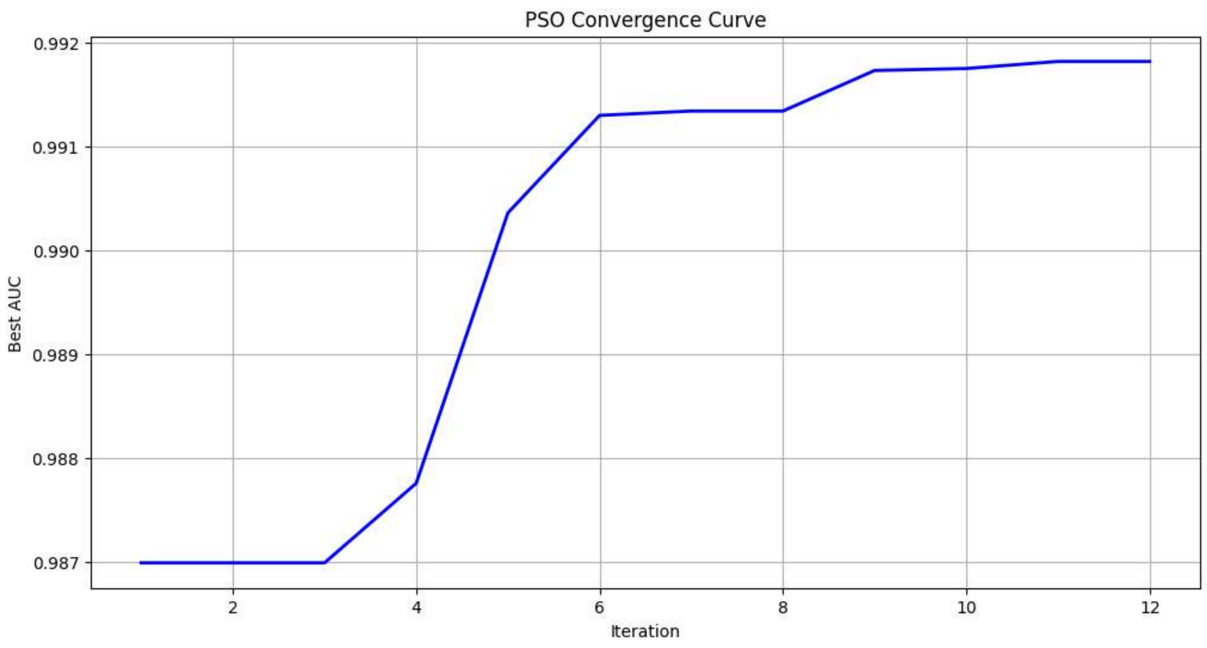

Figure 3. PSO Convergence Curve (Under-Sampling)

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	1550	21	192	233	0	2	0	0	1	1	0
1	70	1626	11	290	0	0	1	2	0	0	0
2	15	1	1906	57	0	0	0	1	0	20	0
3	75	296	44	1582	0	0	0	3	0	0	0
4	0	0	0	0	1555	0	0	0	0	18	427
5	2	1	1	8	0	1832	119	9	27	1	0
6	2	8	3	7	2	357	1607	6	7	1	0
7	0	2	0	0	0	0	0	1969	2	27	0
8	1	2	0	0	0	59	0	9	1925	4	0
9	0	0	0	0	3	0	2	3	4	1977	11
10	0	0	0	0	542	0	0	0	0	23	1435

Table 13. MLP Confusion Matrix (Under-Sampling)

iii. Over sampled data

Metric by PSO	Value
Best AUC	0.9909
Hidden Neurons	200
Learning Rate	0.004174
Reg. Alpha	0.000100

Table 14. PSO outputs Over-Sampling)

Label	Precision	Recall	F1-Score	Support
0	0.94	0.97	0.95	5,858
1	0.94	0.92	0.93	9,079
2	0.75	0.67	0.71	605
3	0.93	0.93	0.93	9,079
4	0.87	0.90	0.88	8,386
5	0.92	0.95	0.93	9,593
6	0.95	0.91	0.93	9,596
7	0.60	0.38	0.46	24
8	0.87	0.94	0.91	291
9	0.94	0.98	0.96	1,844
10	0.90	0.86	0.88	8,456
Overall	0.92	0.92	0.92	62,811
Macro Avg	0.87	0.86	0.86	—
Weighted Avg	0.92	0.92	0.92	—

Table 12. MLP Classification Report (Over-Sampling)

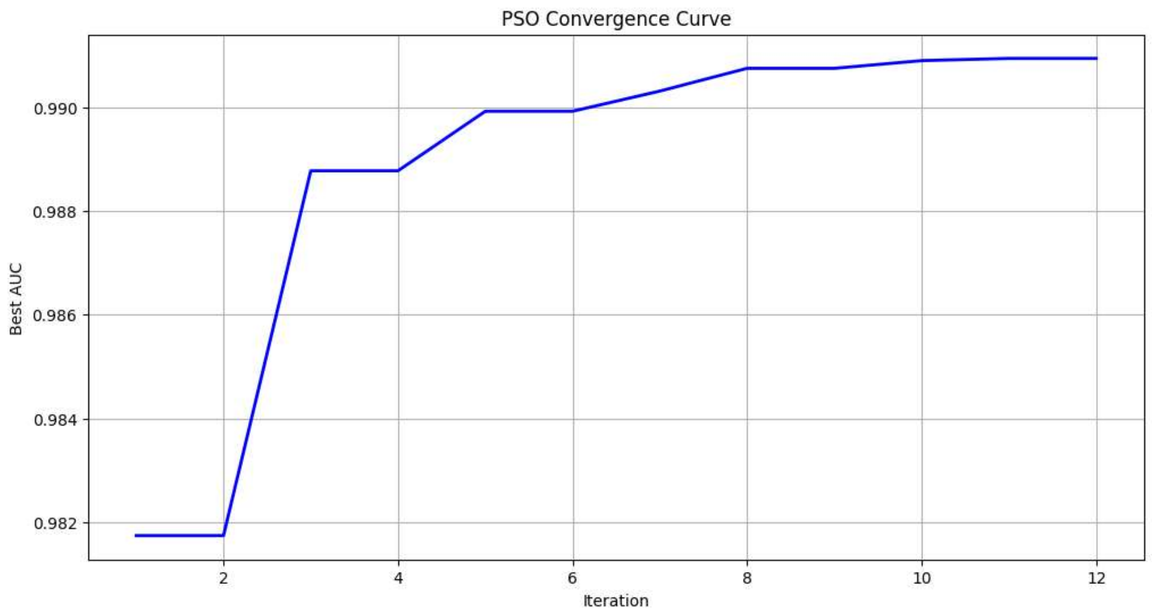

Figure 4. PSO Convergence Curve Over-Sampling)

Confusion Matrix:

True \ Pred	0	1	2	3	4	5	6	7	8	9	10
0	5669	47	123	8	0	8	1	0	0	2	0
1	88	8395	6	578	0	11	1	0	0	0	0
2	195	0	407	0	0	0	0	0	0	3	0
3	70	511	3	8470	0	0	24	0	0	0	1
4	0	0	0	0	7508	0	0	0	0	49	829
5	8	12	1	9	0	9076	462	1	20	4	0
6	10	0	0	36	10	805	8721	3	10	0	1
7	0	0	0	0	0	0	1	9	10	4	0
8	1	0	0	0	0	9	5	2	274	0	0
9	2	0	0	0	20	0	1	0	0	1815	6
10	0	0	0	0	1102	0	0	0	0	58	7296

Table 16. MLP Confusion Matrix Over-Sampling)

10. Final Analysis:

The results from applying Particle Swarm Optimization (PSO) to MLP hyperparameter tuning on the IoT cybersecurity dataset indicate that PSO is an effective tool for enhancing model performance. The comparison of different data sampling strategies revealed that the original, imbalanced dataset yielded the best AUC (0.9938) and the highest overall model performance. However, the oversampled dataset demonstrated better detection of rare classes, which is crucial in cybersecurity tasks where certain types of attacks are less frequent but highly impactful. PSO's ability to balance exploration and exploitation of the search space allowed for optimal hyperparameter settings, resulting in a more accurate and efficient MLP model. These findings emphasize the importance of data handling and optimization techniques in improving the detection capabilities of machine learning models in the context of cybersecurity.

11. Future work

Due to limited computational resources, the extended PSO implementation—which allows optimization of additional hyperparameters such as the sizes of two hidden layers and the training batch size—was not executed in this study. This more advanced variant, implemented as a subclass with a modified fitness function, offers the potential for deeper network architectures and finer control over training dynamics. Although it was not utilized, it represents a valuable direction for future work, enabling more comprehensive and powerful hyperparameter tuning for complex or high-capacity models.