Journal Européen des Systèmes Automatisés
Vol. 59, No. 1, January, 2026, pp. 9-20
Journal homepage: http://iieta.org/journals/jesa

Fast Learning Hyper-Heuristic Framework for Intrusion Detection System

Ahmed Adnan[1,2*], Abdullah Muhammed[1], Abdul Azim Abd Ghani[3], Azizol Abdullah[1], Fahrul Hakim[1]

[1] Department of Communication Technology and Networks, Faculty of Computer Science, and Information Technology, University Putra Malaysia, Serdang 43300, Malaysia
[2] Department of Environmental Science, College of Energy and Environmental Science, Al-Karkh University of Science, Baghdad 10081, Iraq
[3] Department of Software Engineering and Information System, Faculty of Computer Science, and Information Technology, University Putra Malaysia, Serdang 43300, Malaysia

Corresponding Author Email: a.algbory@kus.edu.iq

https://doi.org/10.18280/jesa.590102

Received: 13 April 2025
Revised: 16 June 2025
Accepted: 3 July 2025
Available online: 31 January 2026

Keywords:
intrusion detection system, classification, parallel weights, single weights, hyper-heuristic

ABSTRACT

Detecting cybersecurity attacks remains a challenging problem. This challenge arises from the evolving nature of attacks, a phenomenon commonly referred to as concept drift in machine learning. To address this issue, hyper-heuristic models have been identified as an effective approach. However, the various components embedded in the hyper-heuristic models have created concern about the efficiency of the model as well as its over-fitting or under-fitting of free performance. In this study, the core classifier in the hyper-heuristic model of Intrusion Detection System (IDS) is developed as parallel structure neural network (NN), which enables more controllability of reaching an optimal learning without falling into sub-optimality because of over- and under-fitting. In addition, it enables more efficiency because of reaching higher accuracy with a lower number of neurons. An evaluation of various hyper-heuristics frameworks, some of which are based on single connection NN and others based on the developed parallel connections NN provides the superiority of the latter over the former in terms of all classification metrics when lower number of neurons is used. The evaluation has been conducted on three datasets: KDD 99, NSL-KDD, and LandSat. For KDD, the reached accuracy was 97%-99%. On the other side, we observe that the single connection has generated only an accuracy of 79% with the same number of neurons. From the computation perspective, all hyper-heuristic models have outperformed the benchmark.

1. INTRODUCTION

In the era of Internet of Things (IoT) and cloud-based systems, the security of networks becomes crucial for assuring the operations of the systems [1-3]. The economic and security costs caused by successful network attacks can reach billions of dollars. For example, the impact of false data injection attacks on power systems can adversely affect economic dispatch by increasing the operational cost of the power system or causing sequential overloads and even outages [4].

Hence, industrial firms have dedicated significant budgets to developing immune security for their systems. However, the world is still suffering from non-detected attacks and their impacts on the privacy of users or the normal operation of the systems. Furthermore, according to (https://www.coxblue.com/12-ddos-statistics-that-should-concern-business-leaders/), the rate of attacks has increased more than 2.5 times in the last three years, with an average size of Gbps, which is adequate to take most companies and organizations offline. The concern increases when we consider that 86% of the distributed denial of service (DDoS) attacks belong to multiple types instead of one type. Another concerning factor is the low cost of hiring a cyber-criminal attacker. We add to that the negative impact on the business reputation when the confidence of the consumer decreases.

Security systems aim at updating their immunity against network attacks by using various types of approaches and models [5-7]. Recently, the interest in the artificial intelligence-based security algorithm has increased due to its high potential in protecting systems from non-traditional attacks. Artificial intelligence-based approaches can be built by using neural networks [8], semi-supervised and unsupervised clustering [9], meta-heuristics [10], or even a combination of them in a hybrid approach [8]. The combination of the various components of artificial intelligence has more power in the detection of attacks. Hence, researchers are interested in hybrid-based models for eliminating the risk of failure and increasing the overall performance of the security of the network system.

Regardless, one of the emerging problems in AI-based security systems is the phenomenon of concept drift, which refers to the change over time in the statistical properties of the input data. This occurs with most constructs of the type. Its dynamic environments, such as evolving cyber-attack patterns, where models based on historical data become outdated. Consequently, concept drift affects the marginal distribution

P(X) as well as the conditional P(Y|X) and thus both portions require constant adaptation on model efforts. The traditional models, such as single-connection neural networks (SCNNs), face the same limitations due to their static structure and constant configuration. Without retraining, where real-time computation becomes the goal, SCNNs have no means to effectively update.

The framework of hyper-heuristic that has been developed in the study [11] was found to perform well in terms of the accuracy of attack identification. However, it suffers from the issue of a large number of needed neurons to accomplish the aimed accuracy. Consequently, an increase in the dimension of Moore-Penrose results leads to a higher computational cost when updating the knowledge of the neural network. Another limitation is that the granularity of the neurons' effects is coarse, which might lead to oscillation between under-fitting, which causes biased behaviour or over-fitting, which causes variance behaviour. This falls under the issue of bias-variance trade-off [12]. To resolve this, a newer variant of the hyper-heuristic framework is proposed where the same or even higher accuracy can be achieved with a lower number of neurons and consequently lower computation. This is based on the concept of parallel connectivity inside the structure of a neural network. In other words, there will be no need to increase the number of neurons when the neural network is required to be trained on a higher volume of knowledge; rather, we increase the internal connections or weights inside the neural network. Hence, lower computation is accomplished with satisfactory performance to meet the online nature of the problem. In addition, we modify the classical learning algorithm from being one shot learning to incremental learning to enable knowledge update in the model.

In order to address these challenges, we suggest a new addition to the hyper-heuristic framework that adds parallel interconnectivity within the neural network. Unlike increasing the number of neurons simply to fit additional data, we enhance the internal connections (weights) of the model, which allows it to scale knowledge without increasing complexity. In addition, we shift from a static and “one-shot” learning paradigm to an incremental learning framework that enables real-time changes and keeps the model current amid concept drift.

The remainder of the article is organized as follows. In Section 2, we present the literature survey. Next, the contributions are provided in Section 3. Afterwards, we present the methodology in Section 4. The experimental evaluation and results are given in Section 5. Lastly, the conclusion and future works are provided in Section 6.

2. LITERATURE SURVEY

The literature on intrusion detection systems has emerged in recent years. This problem has not been tackled from various perspectives. Some researchers have focused on stream data clustering [13-15]. while others were more interested in the classification models in general, adopting ensemble-based approaches [13, 16, 17]. The first part has been focusing on the evolving aspect of network traffic when it includes attacks or any type of anomaly, which makes it suitable for handling the concept drift. The second part has been focusing on the issues of imbalanced aspects of training data in intrusion detection systems. Other researchers have focused on the high dimensionality issue of Intrusion Detection System (IDS). In the work [18], the dimensionality has been reduced based on feature ranking and correlation in order to identify useful features that are fed into A feed-forward neural network trained utilizing the Levenberg-Marquardt algorithm. This work has the issue of slow iterative training comparing of its neural network comparing with the one-shot training of the extreme learning machine. In the work [19], an intrusion detection system based on integrated backpropagation and extreme learning machine has been proposed in order to avoid the random behaviour of weights initialization in the input and hidden layer, and replace it with gradient-corrected weights provided from back-propagation, while keeping the online nature of the method. The usage of extreme learning machine variants in intrusion detection systems has been found in other approaches. In the work [20], an extreme learning machine with a hybrid kernel function (HKELM). The algorithm has used for optimizing the hyper-kernel parameter using two algorithms, namely, the gravitational search algorithm (GSA) and differential evolution (DE).

A minority of the approaches have proposed a joint handling of the two issues in one framework. In a recent framework developed by us were proposed combining the power of stream clustering with the power of multi-classification for intrusion detection. The framework has included an evolutionary algorithm for communicating the changes desired to the clustering by the classifier to maintain the learned behaviour of the past attacks. The framework has identified us as the hyper-heuristic framework (HH-F). It includes an online extreme learning machine as a core classifier and online-offline core clustering, it has provided superior performance in terms of predicting attacks. However, the synchronization between the classifier and the clustering blocks requires a fast-learning classifier. The usage of an extreme learning machine, which is a single hidden-layer neural network for this purpose is partially adequate. However, its performance requires adjusting the number of hidden neurons to make it suitable for learning, while avoiding under-fitting on one side because of a lower of number of neurons and over-fitting on the other side because of a higher number of neurons. Furthermore, increasing the number of neurons might cause slow learning. The goal of this article is to upgrade HH.F framework from its current single hidden neural network with only single connections between the input-hidden and hidden output layers to parallel connections from both the input-hidden, hidden output, and input-output layer. In addition, we aim to enable an online learning algorithm to allow knowledge update for every new chunk of data. With such a change, the core classification can reach knowledge fitting with a lower number of neurons, which accomplishes better efficiency.

3. CONTRIBUTIONS

The ultimate goal of the article is to build an intrusion detection system that handles concept drift while achieving efficiency in the learning. The specific contributions are counted as follows.

(1) Modifying the framework presented in the study [11], in the classification block from being single connections of SLFN to parallel connections.

(2) Deriving an incremental learning algorithm for the parallel connections SLFN.

(3) Proving the computational efficiency of the novel

framework.

(4) Evaluating both the core parallel connections SLFN and the whole hyper-heuristic framework based on the core on both synthetic and actual IDS datasets.

4. METHODOLOGY

This section outlines the established methodology for achieving the study's objectives. It starts with an overview of the hyper-heuristic framework and pseudocodes in sub-section A. Next, we provide the parallel structure of the single hidden-layer neural network in sub-section B. Next, the prediction and training model is provided in section C. Afterwards, the online learning model is given in sub-section D. Afterwards, the computational complexity of the algorithm is given in sub-section E. Next, we provide the dataset description in sub-section F. Finally, we have provided the evaluation measurement in sub-section G.

4.1 Overview of the hyper-heuristic framework and pseudocodes

The framework of hyper-heuristic is presented in Figure 1. The framework is composed of three distinct blocks, namely, neural network (NN), core-clustering with online and offline phase and genetic algorithm (GA). We add to them a configuration responsible for feeding the GA with an objective value for optimization based on a calculated error and a class prediction responsible for performing the predictions of the classes (normal vs. attack and the type of attacks).

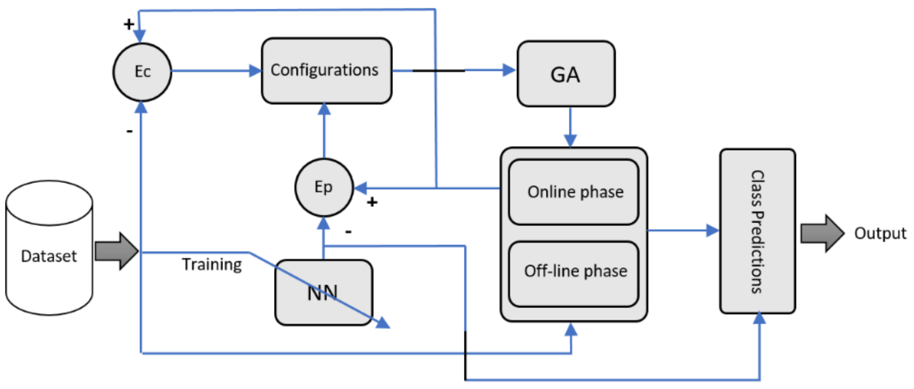


Figure 1. Framework of the hyper-heuristic framework (HH-F) for stream data classification

This framework can operate under one of two variants, namely, hyper 1 and hyper 2, as it is presented in Algorithms 1 and 2, respectively.

Algorithm 1. Pseudocode of hyper-heuristic 1

```
   Data={xi,yi}, i=1,2 ...N          // Streaming data in real-time
   SS Boundary                       // Constraints for the random parameters
   Iterations Numbers
   Rate of Mutations
   Rate of Crossover
   Objective Function='The Core'
   NN0                               // The Output of the initial NN
   Yp={ypi}, i=1,2...N
   Start
   //Phase of boosting
       1.  P0=Random    // the random parameters of the Core.
       2.  NN0=OSELM(x0,y0)
       3.  GA.Options=(SSBoundary,IterationsNumbers,
           RateofMutations,
           RateofCrossover,ObjectiveFunction,
           classPrediction(core(P0,X0))-y0)
       4.  P0=GA.run;
   // Phase of iterative
       5.  For i=2 until N
       6.  //Prediction
       7.  GA.Options=(SSBoundary,IterationsNumbers,
           RateofMutations,
           RateofCrossover,ObjectiveFunction,ClassPrediction(core(P(i-1),xi))-NN(i-1,xi))
       8.  Pi=GA.run;
       9.  ypi= ClassPrediction (core(P(i),xi))
       10. //Correction
       11. NN(i)=OSELM(xi,yi)
   End
```

Algorithm 2. Pseudocode of hyper-heuristic 2

```
   Data={xi,yi}, i=1,2 ...N          // Streaming data in real-time
   SS Boundary                       // Constraints for the random parameters
   Iterations Numbers
   Rate of Mutations
   Rate of Crossover
   Objective Function='The Core'
   NN0                               // The Output of the initial NN
   Yp={ypi}, i=1,2...N
   Start
   //Phase of boosting
       1.  P0=Random    // the random parameters of the Core.
       2.  NN0=OSELM(x0,y0)
```

```
    3.  GA.Options=(SSBoundary,IterationsNumbers,
        RateofMutations,
        RateofCrossover,ObjectiveFunction,
        classPrediction(core(P0,X0))-y0)
    4.  P0=GA.run;
// Phase of iterative
    5.  For i=2 until N
    6.  //Prediction
    7.  GA.Options=(SSBoundary,IterationsNumbers,
        RateofMutations,
        RateofCrossover,ObjectiveFunction,ClassPredi
        ction(core(P(i-1),xi))-NN(i-1,xi))
    8.  Pi=GA.run;
    9.  ypi= ClassPrediction (core(P(i),xi))
    10. //Correction
    11. NN(i)=OSELM(xi,yi)
    12. GA.Options=(SSBoundary,IterationsNumbers,
        RateofMutations,
        RateofCrossover,ObjectiveFunction,ClassPredi
        ction (core(Pi,Xi))-yi)
    13. Pi=GA.run;
End
```

In the first variant, the pseudocode is composed of two phases, namely, the boosting phase and the iterative phase. In the boosting phase, the algorithm receives a labeled boosting chunk for the purpose of initial training to produce the initial neural network NN0. Next, the genetic optimization runs to tune or optimize the parameters of the core clustering, where it uses the difference between the predicted classes by the classifier and the correct ones. Afterwards, the algorithm runs for every newly arrived chunk and performs two sub-phases, namely, prediction based on the last learned classifier and training based on the last labelled chunk. The difference between hyper 1 and hyper 2 is in the correction phase, where the former does not perform an additional optimization step based on the correct values of the classes, like the latter.

4.2 Parallel structure of a single hidden layer neural network

In the original single hidden layer neural network used in hyper-heuristic, there were only connections between the input and hidden layer and connections between the hidden and output layer [21]. The topology of this NN is shown in Figure 2.

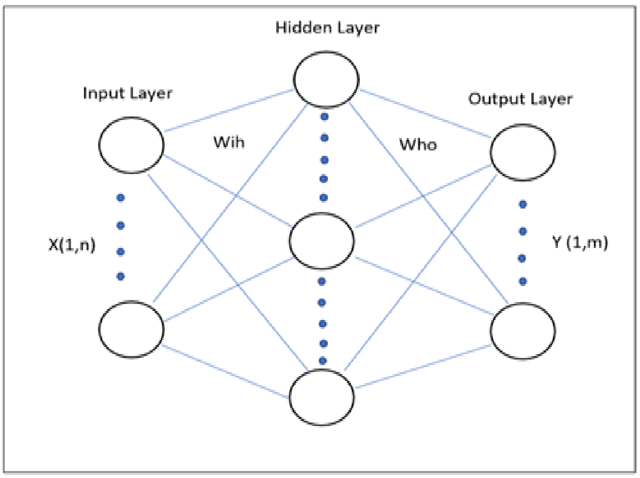


Figure 2. Neural network for a single hidden layer

In the new topology, we are inspired by the study [22] where direct connections between the input and output layer are added, which enables more capability of embedding knowledge with a lower number of neurons in the hidden layer. The topology of the new structure is given in Figure 3.

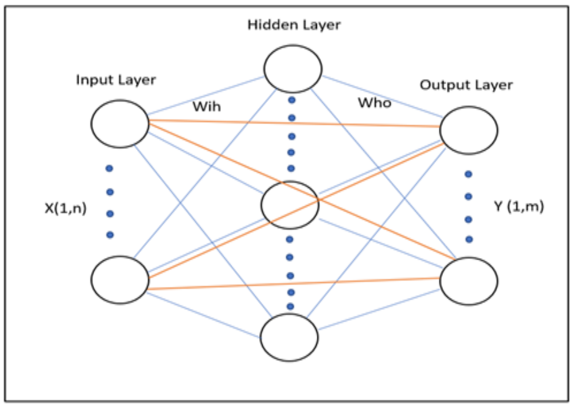


Figure 3. Parallel structure of a single hidden layer neural network (add the red connections between the input and output to reduce the number of neurons)

As shown in Figure 3, the red connections is the newly added connections as parallel links incorporated into the neural network structure, which are designed to increase internal connectivity without increasing the number of neurons. This connection provides more flexibility in adapting to changes in the data and improves the model's efficiency in performing incremental learning.

The weights between the input and hidden layer are denoted by W_{ih} ,which is represented in a matrix with size of $n \times \tilde{N}$ and it is given as it is shown in Eq. (1).

$$W_{ih} = \begin{bmatrix} w_{11} & w_{12} & . & w_{1\tilde{N}} \\ w_{21} & w_{22} & . & w_{2\tilde{N}} \\ . & . & . & . \\ . & . & . & . \\ w_{n1} & w_{n2} & . & w_{n\tilde{N}} \end{bmatrix} \quad (1)$$

where,

n denotes the number of inputs or features.

$\tilde{N}$ denotes the number of hidden neurons.

The weights between the hidden layer and the output are denoted by W_{ho}, which is represented in the matrix $\tilde{N} \times m$ and it is given as it is shown in Eq. (2).

$$W_{ho} = \begin{bmatrix} w_{11} & w_{12} & . & w_{1m} \\ w_{21} & w_{22} & . & w_{2m} \\ . & . & . & . \\ . & . & . & . \\ w_{\tilde{N}1} & w_{n2} & . & w_{\tilde{N}m} \end{bmatrix} \quad (2)$$

The weights between the input and output (which represents the parallel weights) are given as W_{io} with the size $n \times m$ and it is given as it is shown in Eq. (3).

$$W_{io} = \begin{bmatrix} w_{11} & w_{12} & . & w_{1m} \\ w_{21} & w_{22} & . & w_{2m} \\ . & . & . & . \\ . & . & . & . \\ w_{n1} & w_{n2} & . & w_{nm} \end{bmatrix} \quad (3)$$

In addition to the weights, we have the biases that are introduced as additive terms to the neurons in the hidden layer, and they are given as it is shown in Eq. (4).

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ b_{\tilde{N}} \end{bmatrix} \tag{4}$$

4.3 Prediction and training model

Assuming that we know the weights of the neural network W^{oi}, W_k^{oh}, W_k^{in} where, W^{oi} represents the weights of the input output layer, W_k^{oh} represents the weights of the hidden layer with the output for neuron k, W_k^{in} is the weights of input layer with the hidden neuron k and assuming that we have samples x_j where $j = 1, \dots N$. The predicted output of x_j is given in y_j. It is given as it is presented in Eq. (5).

$$y_j = f(W^{oi}x_j + c + \sum_{k=1}^{m} W_k^{oh} g(W_k^{in}x_j + b_k)), \quad j = 1,2, \dots, N \tag{5}$$

The equation can be represented in compact form as it is shown in Eqs. (6) and (7).

$$Y = f(W^{oi}X + W^{oh}G + c) = f([W^{oi}W^{oh}c]\begin{bmatrix} X \\ G \\ I \end{bmatrix}) = f(W\begin{bmatrix} \text{X} \\ \text{G} \\ \text{I} \end{bmatrix}) \tag{6}$$

$$G = g(W^{in}X + b) \tag{7}$$

where,

W^{oi} denotes the input-output weights.

W^{oh} denotes the hidden output weights.

b denotes the biases of the hidden layer.

I denotes vector of ones and length of N.

c denotes the bias of the output layer.

g denotes the first activation function.

f denotes the second activation function.

G denotes the output of the input layer.

The training model is responsible of calculating the weight matrices W^{oi}, W^{oh}, b. It assumes that we have X. Next, W^{in} is calculated randomly and we use it to calculate G while W^{oi} and W^{oh} are calculated based on the Eqs. (8) and (9).

$$\widehat{w} = TH^{\dagger} \tag{8}$$

$$H = \begin{bmatrix} X \\ G \end{bmatrix} \tag{9}$$

In the remainder of the article, we denote the single weights model as (HS), and it has two variants, H1S and H2S for hyper 1 and hyper 2, respectively. Similarly, we denote the model of parallel weights as (HP), and it has two variants, H1P and H2P for hyper 1 and hyper 2, respectively.

4.4 Online learning model

The issue with the parallel classifier that is adopted is that it is not suitable to be incorporated directly into the hyper-heuristic framework. The reason is that it does not support online learning according to the previous training equation. Hence, we further develop it to support online learning as follows.

We assume that the data is given as $\text{X} = \{(X_i, t_i)|X_i \in R^{n_i \times n}, t_i \in R^{n_i \times m}, i = 1, \dots, \tilde{N}\}$.
where, n_i denotes the number of records in the chunk i and n denotes the number of columns. X_i denotes a chunk of data and m denotes the number of classes. t_i denotes the target. The sequential training of the parallel structure is given in two phases: the first one is the boosting phase, and the second one is the sequential learning phase.

(1) Phase 1: The boosting phase

1)-Generate randomly the input weights ${w_0}^{in}$.

2)-Calculate the hidden layer output matrix G_0.

3)-Calculate the output weight matrix using Moore-Penrose.

4)-Extract the two weights matrices ${W_0}^{oi}$ and ${W_0}^{oh}$.

(2) Phase 2: Sequential learning phase

For each further coming chunk (X_i, t_i), where $X_i \in R^{n_i \times n}$ and $t_i \in R^{n_i \times m}$.

$i = 1,2, \dots NC$ where NC denotes the number of chunks:

1-Compute the output matrix of the hidden layer G_k.

2-Determine the most recent output matrix $\widehat{w}^{(k+1)}$ using Eqs. (10) and (11).

$$M_{k+1} = M_k - \frac{M_k\begin{bmatrix} X_k \\ G_k \end{bmatrix} + \begin{bmatrix} X_k \\ G_k \end{bmatrix}_{k+1}^T M_k}{1 + \begin{bmatrix} X_k \\ G_k \end{bmatrix}_{k+1}^T M_k \begin{bmatrix} X_k \\ G_k \end{bmatrix}_{k+1}} \tag{10}$$

$$\widehat{w}^{(k+1)} = \widehat{w}^{(k)} + M_{k+1}H_{k+1}(t_i^T - H_{k+1}^T \widehat{w}^{(k)}) \tag{11}$$

Extract the two weights matrices ${W_{k+1}}^{oi}$ and ${W_{k+1}}^{oh}$ from $\widehat{w}$

$$k = k + 1$$

The boosting phase of algorithm training embodies the following steps: the generation of input weights ${w_0}^{in}$ randomly, the calculation of the hidden layer output matrix G_0, the computation of the output weight matrix with Moore-Penrose where two weights matrices input-output ${W_0}^{oi}$ and hidden-output ${W_0}^{oi}$ are extracted. The steady sub-phase (the sequential learning phase) includes processing each for chunks (X_i, t_i), so in the earlier iterations we compute the hidden layer output matrix G_k and after the latest output matrix at this iterative step, extract the two weights matrices ${W_{k+1}}^{oi}$ and ${W_{k+1}}^{oh}$, which is how we modified the offline version of the parallel scheme, which exists in the literature.

The recursive learning is supported because the Eqs. (10) and (11) enable updating the weights of the neural network $\widehat{w}^{(k+1)}$ based on the previous weights $\widehat{w}^{(k)}$.

In this work, the problem of class imbalance in the data streams was not resolved with intent of retaining the realism of intrusion detection systems in which attack instances are considerably infrequent compared to normal traffic. Rebalancing the data artificially by oversampling the minority classes or undersampling the majority class creates a distribution that would not exist in reality and would put the model's generalizability at risk.

The model evaluation, instead, is done with the inherent imbalance of KDD99 or NSL-KDD datasets which provides a more realistic appraisal of model evaluation in real-world scenarios. The goal of this decision is to emulate an operational IDS for which unknown attacks would be seen as large and benign traffic would be seen as surreptitious and unpredictable.

We do understand that class imbalance is an important problem in the design of IDS. In future work, we intend to investigate adaptive approaches that are imbalanced-aware by default in unstable environments and respond to the degree of imbalance without destroying the flow of data.

4.5 Computational complexity

The difference between single hidden layer training with and without parallel weights is calculated by using Moore-Penrose computational complexity which is given as $O(n\tilde{N}^2) + O(\tilde{N}^3)$ for a matrix H of size $n \times \tilde{N}$ or it is given as $O(n\tilde{N}) + O(\tilde{N}^2)$ by using the approach of the study [23].

In our case, the number of rows in the matrix represents the number of samples n and the number of columns in the matrix represents the number of neurons in the hidden layer $\tilde{N}$. Assuming that we reduced the number of neurons for the parallel model with percentage of $1/R$, the calculated complexity of the parallel model is $O\left(n\frac{1}{R^2}\tilde{N}^2\right) + O\left(\frac{1}{R^3}\tilde{N}^3\right)$ or $O\left(n\frac{1}{R}\tilde{N}\right) + O\left(\frac{1}{R^2}\tilde{N}^2\right)$.

The theoretical complexity of the proposed model was already analyzed as $O(n/N^2)$ but, as far as the actual runtime results are concerned, it is important to confirm empirically such claim. To this end, we measured the CPU time of the model with different input sizes and settings.

Experimental Setup:

• Hardware: Intel Core i7-12700K 32 GB RAM
• Software: MATLAB 2022a
• Dataset: NSL-KDD and KDD99
• Chunk sizes tested: 1000, 2000, 5000, 10,000 samples

Results from the runtime analysis indicate that the system keeps a sublinear increase in execution time with increase in input size, particularly when the number of neurons N is scaled down proportionately, aligning with the theoretical assumption of $O(n/N^2)$. This verification shows that the expected application of parallel connectivity, along with incremental updates does effectively reduces computational costs, thus adapting the model to be more efficient for online learning in real-world settings.

4.6 Dataset description

This part explains the evaluation datasets to be utilized. Three real datasets are chosen for evaluating the work: KDD'99, NSL-KDD, and Landsat.

(1) KDD'99

KDD'99 [24] may be the famous complete dataset for testing and analysis anomaly detection techniques. This dataset derives from the DARPA'98 IDS data structure which encompasses many numerous gigabytes of compressed raw (binary) TCP dump data on networks over the period of seven weeks, in addition to 5 million connection of records reached to be around 100 bytes each. Furthermore, the dataset contains around 2 million connection of records. Each record has 41 fields which are marked as normal, DoS, U2R, R2L, or probing along with some graded level attacks.

(2) NSL-KDD

Multiple studies have come across challenges regarding workplace productivity, which are linked to the KDD'99 dataset [24]. NSL-KDD was proposed, in which they eliminated duplicated entries from the training and testing data; furthermore, they also set the level of certain groups to be negatively correlated to the percentage of records contained within the original KDD'99 dataset. There were 21 unique attacks that were included in the training dataset out of the 37 available in the test dataset. Some of the known attacks are incorporated into the training data, while some unknown attacks are also some of the attacks not included in the training data.

(3) Landsat satellite

This dataset contains a total of 4,435 objects that come from remote-sensing satellite images. Each object pertains to a region and is subdivided into sub-regions based on four intensity captures at different wavelengths. The dataset has a total of 36 attributes. This dataset is used by the study [25].

4.7 Evaluation measurement

This part describes the measures that used to evaluate the effectiveness of HH-F and how it stacks up against the best-known approaches in the domain. TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

(1) Accuracy

Accuracy is defined as the ratio of correct predictions to the total number of predictions made [26]. This calculation is expressed in Eq. (12).

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

(2) Precision (PPV)

PVV According to PVV refers to the number of positive identifications that were actually correct. This reflects the number of true positive predictions made by the classifier in the relation of a total count of records predicted as positive. In this case, records containing confirmed positive examples included in the data set [27]. This calculation is expressed in Eq. (13).

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{13}$$

(3) Recall (TPR)

TPR signifies the ratio of true positives predicted by the classifier to the actual positive records on the dataset [28]. The calculation is expressed in Eq. (14).

$$TPR = \frac{TP}{TP + FN} \tag{14}$$

(4) Specificity (TNR)

TNR indicates the proportion of true negatives predicted by the classifiers relative to all negative records that were evaluated (tested) [29]. This calculation is presented in Eq. (15).

$$TNR = \frac{TN}{N} \tag{15}$$

(5) NPV

This metric refers to negative predicted value which is calculated from the ratio of predictive true negatives to the total value of negative cases as explained in the study [30]. The calculation is expressed in Eq. (16).

$$NPV = \frac{TN}{TN + FN} \quad (16)$$

(6) G_mean

This metric is determined by forming the combination of precision and recall [27]. This calculation is presented in Eq. (17).

$$G\_mean = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}} \quad (17)$$

(7) F_Measure

The F_Measure denotes the harmonic mean of both precision and recall. This calculation is presented in Eq. (18).

$$F\_measure = \frac{2 \times precision \times recall}{precision + recall} \quad (18)$$

5. EXPERIMENTAL WORKS AND RESULTS

The evaluation has been conducted by comparing both H1P and H2P with H1S and H2S. In addition, we compared our models with PSO-FLN [22]. For generating the results, we used the parameters provided in Table 1. The results generation was based on two stages: the first one is characterization based on the number of neurons ranging from 25 to 100 to observe the dependency of the models on that and it is given in sub-section A. the next one is Comparison with benchmarks, and it is given in sub-section B, the computation evaluation is presented in sub-section C.

Table 1. Configuration used for generating the results

Parameter Name	Value
Number of generations	10
Population	35
Mutation rate	0.05
Crossover rate	0.6
Stream speed	100
Activation function	sig
Chunk size	100
Number of neurons	25,50,100

To guarantee the robustness and reproducibility of the proposed model, we carried out hyperparameter tuning for the GA parameters such as population size, mutation rate, and crossover probability. We used a grid search method and the following ranges:

• Size of the population: {35,55,75,100}

• Rate of mutation: {0.01, 0.05, 0.1, 0.2}

• Crossover rate: {0.6, 0.7, 0.8, 0.9}

The model was evaluated on a validation subset, and the average classification correctness and stability over multiple runs were used for selection.

5.1 Performance characterization

To optimize the trade-off between the number of the neurons in each of the two models and the accomplished performance, we have run each HS and HP models on number of neurons equal to 25, 50 and 100. We repeated this for the three types of datasets, namely, KDD99, NSL and LandSat.

We present in Figure 4 a radar graph for all performance metrics of both HS and HP of KDD 99. The evaluation shows the superiority of both H1P and H2P over H1S and H2S in terms of all prediction's metrics. This indicates to the useful ness of the parallel structure in enabling more optimal learning with avoidance of under or over fitting comparing with the single structure.

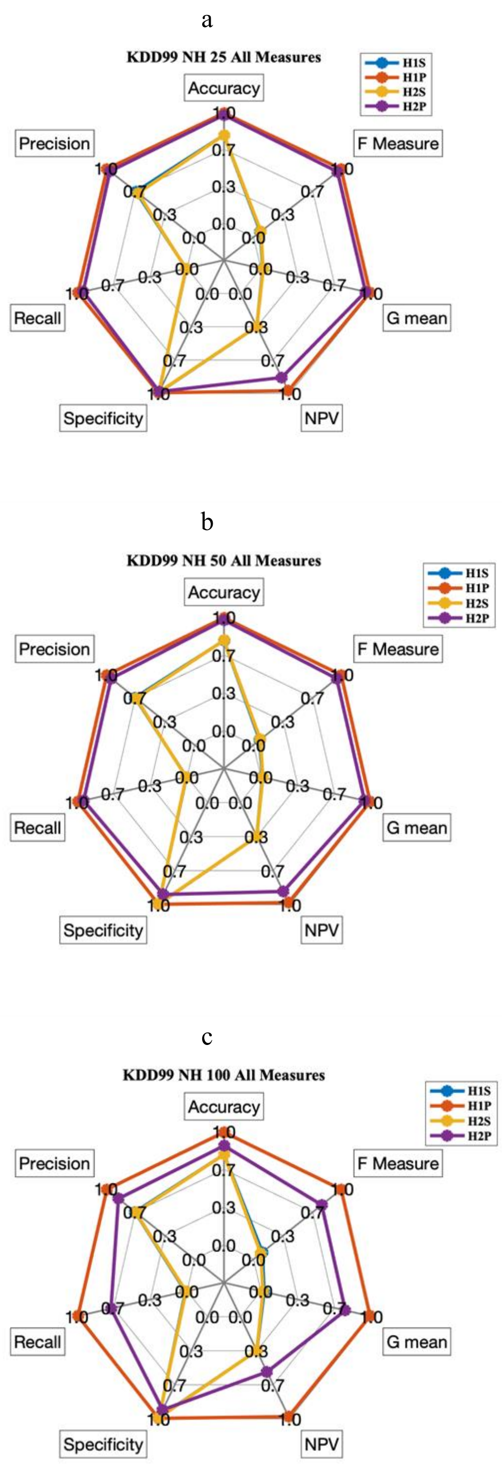
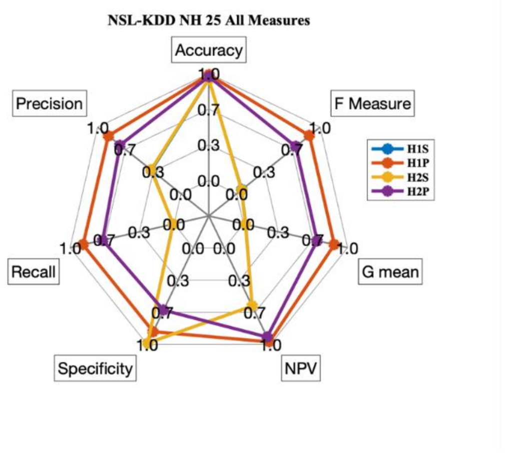
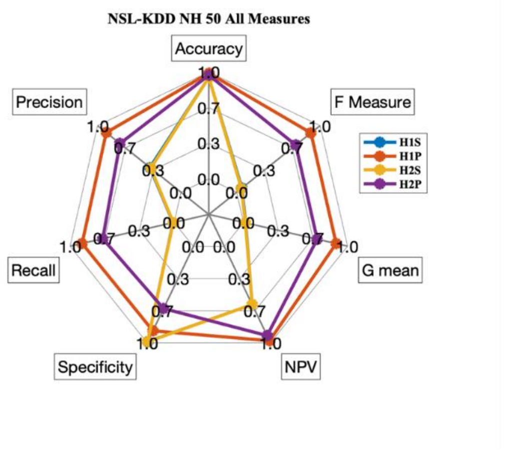


Figure 4. The performance metrics for H1S, H2S, H1P and H2P based on KDD99 -a- 100 neurons -b- 50 neurons -c- 25 neurons

a



b



c

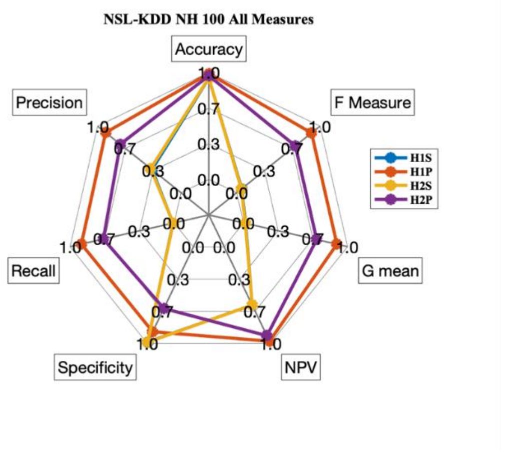
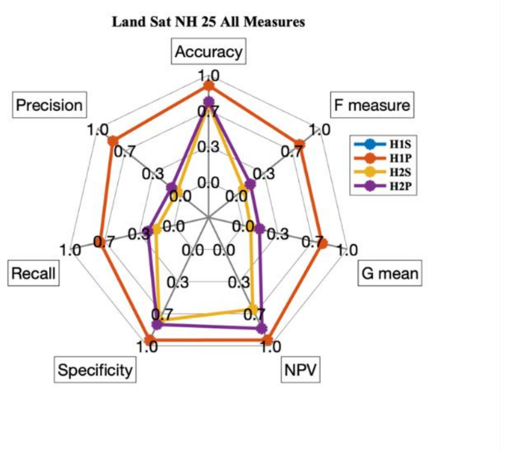


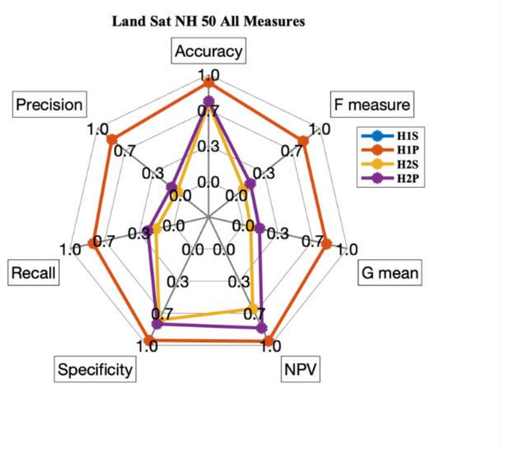
Figure 5. The performance metrics for H1S, H2S, H1P and H2P based on NSL-KDD -a- 100 neurons -b- 50 neurons -c- 25 neurons

Similarly, we present the prediction metrics of H1P and H2P compared with H1S and H2S for the other two datasets, namely, NSL-KDD and Landsat as it is presented in Figures 5 and 6, respectively. The figures emphasize on the superiority of the parallel structure over the single one. In addition, we observe that H1P has accomplished higher values of classification metrics comparing with H2P, which might be caused by the additional calling of genetic in the former which leads to over-fitting when the dynamic of the data is high. Also, we observe that both H1S and H2S have provided similar performance but less that H2P and H1P despite the number of neurons in the hidden layer. Consequently, we find that the effectiveness of the parallel weights is more than the number of neurons to reach the best optimality of knowledge capturing in the model.

a



b



c

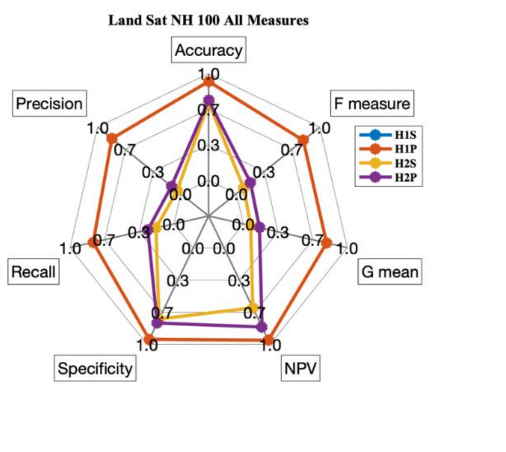


Figure 6. The performance metrics for H1S, H2S, H1P and H2P based on LandSat -a- 100 neurons -b- 50 neurons -c- 25 neurons

5.2 Comparison with benchmarks

The evaluation of our developed models is compared with the benchmark [22] which uses also parallel classifier, but it adds to it an opt0imizer for increasing the optimality of the weights . The benchmark choice of the PSO-FLN algorithm is due to its accuracy in optimizing neural networks. It boosts performance by increasing the connection quantity within the neural network while tuning the weights through Particle Swarm Optimization (PSO). Although it enhances predictive performance, it adds heavy computational burden. The focus on global optimization approaches makes PSO-FLN algorithms less suitable for environments with real-time or streaming data where quick adaptation is crucial.

In comparison, our proposed approach sticks to the same principle of optimizing the neural network's working memory by increasing its neural connections, solely tailored for online learning. The model learns step by step in real-time, removing the need for time-intensive global optimization. This structure provides quick adaptability with stream-fashioned data while keeping precision, providing an edge over PSO-FLN designs that rely on static parameters, making it more effective in freestyling shifts with real-time data.

To observe the overall difference in the performance between the single connection hyper structures and the parallel connection hyper structure, we presented the results with respect to different number of neurons, namely, 25, 50 and 100 neurons. On the other side, we only operated a hyper-heuristic parallel structure on number of neurons 25. We see that both hyper 1 and hyper 2 when operating the system on 25 neurons have yielded higher accuracy than the single connection.

In Table 2, which shows the results of KDD99 it is observed that hyper 1 with parallel connection has reached an accuracy of 99.4% while the single hidden layer connection has reached an accuracy of 79.8% with number of neurons 50 and the accuracy has decreased to 79.5% when the number of neurons was 100. Also, we observe similar results of hyper 2 structure as the accuracy was found to be 97.5% for parallel connection hyper 2, while it was only 79.8% for single connection hyper 2 with a number of neurons of 100. Similarly, we observe for Table 3, which shows the results of NSL-KDD an accuracy of 99% and 97.3% for hyper 1 and hyper 2 parallel connection structure respectively, compared with only 94.8% and 94.9% as the maximum achieved accuracy of hyper 1 and hyper 2 single connection, respectively. This even was observed more in Table 4, which shows the results of Landsat where the accuracy was 90.3% and 75.0% for hyper 1 and hyper 2 parallel connection structure, respectively with number of neurons of 25, compared with only 72.5% as maximum achieved accuracy for both hyper 1 and hyper 2 single connection, even with a number of neurons of 100.

For benchmark evaluation, we compared all hyper-heuristic variants with the benchmark PSO-FLN with 25 neurons, it was found that hyper model for KDD 99 in Table 2 has accomplished competitive performance to H1P with slight superiority of H1P over PSO-FLN in terms of accuracy which was found to be for the former as 99.4% and for the latter 99.3%. This accuracy change is because of better recall for H1P compared with better precision of PSO-FLN. The similar performance between H1P and PSO-FLN is also observed in NSL-KDD 99 in Table 3. However, we observe a clear superiority of H1P over PSO-FLN for the LandSat dataset for all metrics in Table 4.

Table 2. The classification metrics of IDS for hyper-heuristics 1, 2 single and parallel structure for KDD 99

KDD99		**H1S**		**H1P**		**H2S**		**H2P**	**PSO-FLN**
Hidden neurons	25	50	100	25	25	50	100	25	25
Accuracy	0.795	0.798	0.795	0.994	0.796	0.797	0.798	0.975	0.993
Precision	0.658	0.668	0.668	0.988	0.636	0.658	0.661	0.952	0.990
Recall	0.009	0.015	0.007	0.986	0.011	0.009	0.010	0.939	0.982
Specificity	0.990	0.985	0.993	0.995	0.987	0.990	0.989	0.983	0.992
NPV	0.332	0.332	0.332	0.973	0.332	0.332	0.332	0.841	0.993
G_mean	0.018	0.029	0.014	0.986	0.021	0.018	0.020	0.946	0.986
F_Measure	0.077	0.099	0.069	0.986	0.082	0.077	0.081	0.946	0.986

Table 3. The classification metrics of IDS for hyper-heuristics 1, 2 single and parallel structure for NSL-KDD 99

NSL-KDD99		**H1S**		**H1P**		**H2S**		**H2P**	**PSO-FLN**
Hidden neurons	25	50	100	25	25	50	100	25	25
Accuracy	0.948	0.949	0.948	0.990	0.949	0.950	0.949	0.973	0.990
Precision	0.355	0.372	0.349	0.870	0.361	0.362	0.364	0.815	0.915
Recall	0.010	0.010	0.010	0.876	0.010	0.009	0.009	0.709	0.899
Specificity	0.988	0.988	0.987	0.869	0.989	0.990	0.989	0.941	0.962
NPV	0.602	0.603	0.602	0.975	0.603	0.603	0.603	0.940	0.953
G_mean	0.020	0.020	0.020	0.873	0.019	0.018	0.017	0.758	0.907
F_Measure	0.060	0.062	0.060	0.873	0.059	0.057	0.057	0.760	0.907

Table 4. The classification metrics of IDS for hyper-heuristics 1, 2 single and parallel structure for Land Sat

Land Sat		**H1S**		**H1P**		**H2S**		**H2P**	**PSO-FLN**
Hidden neurons	25	50	100	25	25	50	100	25	25
Accuracy	0.725	0.725	0.725	0.903	0.725	0.725	0.725	0.750	0.744
Precision	0.045	0.045	0.045	0.816	0.045	0.045	0.045	0.112	0.054
Recall	0.174	0.174	0.174	0.709	0.174	0.174	0.174	0.251	0.234
Specificity	0.738	0.738	0.738	0.942	0.738	0.738	0.738	0.778	0.765
NPV	0.621	0.621	0.621	0.941	0.621	0.621	0.621	0.820	-
G_mean	0.072	0.072	0.072	0.759	0.072	0.072	0.072	0.155	0.113
F_Measure	0.089	0.089	0.089	0.761	0.089	0.089	0.089	0.168	0.088

As we saw in the previous Tables 2 and 4, the differences in performance and accuracy (79% on the KDD99 dataset for both) between H1S and H2S are minor in nature. There are several factors that can describe why these differences exist, as elaborated below.

1. Algorithmic Structure:

The core principles in both H1S and H2S frameworks are somewhat identical. The difference arises with H2S because, in the optimization phase, extra heuristics or specific parameters are added. This may lead to differences in convergence speed or stability. From the results, it is clear that the differences are minimal, especially for complex datasets like KDD99.

2. Dataset Complexity:

Another factor that contributes to the approximate 79% accuracy on both models is the pronounced and sophisticated nature of the KDD99 dataset. Although KDD99 is broadly accepted, it encompasses several diversified attacks as well as normal instances. Because the dataset is inherently complex, it is plausible that the models were compelled to converge in the same manner. The accuracy implies that both frameworks are capable of learning from the dataset and that the optimization processes on either of the frameworks do not meaningfully influence the performance.

5.3 Computation evaluation

Another evaluation was conducted to explore the computational performance of our developed H1P and H2P and to compare them with the benchmark. As it is shown in Tables 5, 6 and 7 the computational time of H1P is similar to H1S when the number of neurons equal 25 and it represents the least computational time comparing with the other algorithm of H1S or H2P of higher number of neurons, namely, 50 and 100. Furthermore, it is less than the computational time of the benchmarks of PSO-FLN. This is observed for the three datasets, and it is interpreted by the fact that the number of neurons in the models of HH-F, namely, H1S, H2S, H1P and H2P are the most consuming factor of computation and the nature of PSO-FLN in consuming computational cost at the searching in the weights. Considering the classification metrics reported in the previous sections of H1P for number of neurons of 25 and its outperformance to H1S and H2S for number of neurons of 50 and 100, and the light computation of H1P for number of neurons of 25, we state that H1P has accomplished both efficiency as well as accurate prediction. Again, this is interpreted by the embedded knowledge in the parallel weight and its less harm on the computational load on one side and it is fine tuning of the topology in a way that prevents overfitting, or any degradation of performance caused by extra connections or weights caused from higher number of neurons.

Table 5. Computation evaluation for our approach and the benchmark with number of neurons equals to 25 for KDD 99

KDD 99		H1S		H1P		H2S		H2P	PSO-FLN
Hidden neurons	25	50	100	25	25	50	100	25	25
Computation time	796	6245	49476	796	2020	7470	50701	2020	6919

Table 6. Computation evaluation for our approach and the benchmark with number of neurons equals to 25 for NSL-KDD 99

NSL-KDD99		H1S		H1P		H2S		H2P	PSO-FLN
Hidden neurons	25	50	100	25	25	50	100	25	25
Computation time	325	2550	20200	325	825	3050	20700	825	2825

Table 7. Computation evaluation for our approach and the benchmark with number of neurons equals to 25 for LandSat

LandSat		H1S		H1P		H2S		H2P	PSO-FLN
Hidden neurons	25	50	100	25	25	50	100	25	25
Computation time	72	565	4479	72	182	676	4590	182	626

6. CONCLUSION AND FUTURE WORK

The hyper-heuristic framework is composed of three main distinct blocks, namely, neural network (NN), core-clustering with online and offline phase and GA. Its core-classification block is NN, which contains a single hidden layer. This article has developed this core to include parallel connections. This development has enabled a greater degree of freedom for enabling optimal learning with an avoidance of sub-optimality due to under-fitting or over-fitting. In addition, it provides capability of reaching better performance with a lower number of neurons which leads to more efficiency in the model. The evaluation has been conducted based on three datasets, namely, KDD, NSL-KDD and LandSat. The reached accuracy for KDD 99 was 99% for hyper 1 and was 97% for hyper 2. On the other side, we observe that the single connection has generated only an accuracy of 79% with 100 neurons. For NSL-KDD, we observe that the accuracy was 98.8% and 97% for hyper 1 and hyper 2 parallel connection structure respectively, compared with only 94.8% and 94.9% as the maximum achieved accuracy of hyper 1 and hyper 2 single connection, respectively. In addition, we observe similar outperformance of parallel variants for LandSat dataset. From the computational cost perspective, both H1P and H2P have shown low computational cost that outperforms PSO-FLN. The reason is their low number of neurons and support for online learning that PSO-FLN lacks.

While the methodology outlines a beneficial approach, there are several considerations that impact its effectiveness, accuracy, and practical application, which need to be resolved. First, an increasing number of feature counts is inflexible for dealing with high-dimensional data. The model is prone to the curse of dimensionality, which results in longer training periods and reduced generalization capability resulting from overfitting. The second includes some bias in the imbalance class problem, most common in many IDS datasets, where the model might become overfitted to prevalent attack patterns. For the methods proposed, an assumption is made that the data is supported in steps conducted adequately, concentrating on verifying class subset caps misclassified as spam. Around the detection of rare classes, there still remains an issue of data-centric sharpening, which goes beyond data manipulation.

With other proposed enhancements, self-perturbate transformations for the modification of increasingly specific data with online IKD techniques, or hyper-heuristic derivation techniques utilizing kinesics, which also exploit ensemble and debris techniques that enhance sparse data add-back methods not constrained by static ID and IP value.

REFERENCES

[1] Alladi, T., Chamola, V., Sikdar, B., Choo, K.K.R. (2020). Consumer IoT: Security vulnerability case studies and solutions. IEEE Consumer Electronics Magazine, 9(2): 17-25. https://doi.org/10.1109/MCE.2019.2953740

[2] Hussain, F., Hussain, R., Hassan, S.A., Hossain, E. (2020). Machine learning in IoT security: Current solutions and future challenges. IEEE Communications Surveys & Tutorials, 22(3): 1686-1721. https://doi.org/10.1109/COMST.2020.2986444

[3] Atlam, H.F., Wills, G.B. (2020). IoT security, privacy, safety and ethics. Digital Twin Technologies and Smart Cities, 123-149. https://doi.org/10.1007/978-3-030-18732-3_8

[4] Xu, Y. (2020). A review of cyber security risks of power systems: From static to dynamic false data attacks. Protection and Control of Modern Power Systems, 5(1): 19. https://doi.org/10.1186/s41601-020-00164-w

[5] Alqahtani, H., Sarker, I.H., Kalim, A., Minhaz Hossain, S.M., Ikhlaq, S., Hossain, S. (2020). Cyber intrusion detection using machine learning classification techniques. In Computing Science, Communication and Security: First International Conference, COMS2 2020, Gujarat, India, pp. 121-131. https://doi.org/10.1007/978-981-15-6648-6_10

[6] Almiani, M., AbuGhazleh, A., Al-Rahayfeh, A., Atiewi, S., Razaque, A. (2020). Deep recurrent neural network for IoT intrusion detection system. Simulation Modelling Practice and Theory, 101: 102031. https://doi.org/10.1016/j.simpat.2019.102031

[7] Sarker, I.H., Abushark, Y.B., Alsolami, F., Khan, A.I. (2020). Intrudtree: A machine learning based cyber security intrusion detection model. Symmetry, 12(5): 754. https://doi.org/10.3390/sym12050754

[8] Dutta, V., Choraś, M., Kozik, R., Pawlicki, M. (2021). Hybrid model for improving the classification effectiveness of network intrusion detection. In 13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020) 12, pp. 405-414. https://doi.org/10.1007/978-3-030-57805-3_38

[9] Jiang, S., Song, X., Wang, H., Han, J.J., Li, Q.H. (2006). A clustering-based method for unsupervised intrusion detections. Pattern Recognition Letters, 27(7): 802-810. https://doi.org/10.1016/j.patrec.2005.11.007

[10] Zhang, Y., Li, P., Wang, X. (2019). Intrusion detection for IoT based on improved genetic algorithm and deep belief network. IEEE Access, 7: 31711-31722. https://doi.org/10.1109/ACCESS.2019.2903723

[11] Adnan, A., Muhammed, A., Abd Ghani, A.A., Abdullah, A., Hakim, F. (2020). Hyper-heuristic framework for sequential semi-supervised classification based on core clustering. Symmetry, 12(8): 1292. https://doi.org/10.3390/sym12081292

[12] Neal, B., Mittal, S., Baratin, A., Tantia, V., Scicluna, M., Lacoste-Julien, S., Mitliagkas, I. (2018). A modern take on the bias-variance tradeoff in neural networks. arXiv preprint arXiv:1810.08591. https://doi.org/10.48550/arXiv.1810.08591

[13] Badotra, S., Panda, S.N. (2021). SNORT based early DDoS detection system using Opendaylight and open networking operating system in software defined networking. Cluster Computing, 24(1): 501-513. https://doi.org/10.1007/s10586-020-03133-y

[14] Hyde, R., Angelov, P., MacKenzie, A.R. (2017). Fully online clustering of evolving data streams into arbitrarily shaped clusters. Information Sciences, 382: 96-114. https://doi.org/10.1016/j.ins.2016.12.004

[15] Islam, M.K., Ahmed, M.M., Zamli, K.Z. (2019). A buffer-based online clustering for evolving data stream. Information Sciences, 489: 113-135. https://doi.org/10.1016/j.ins.2019.03.022

[16] Zhou, Y., Mazzuchi, T.A., Sarkani, S. (2020). M-AdaBoost-a based ensemble system for network intrusion detection. Expert Systems with Applications, 162: 113864. https://doi.org/10.1016/j.eswa.2020.113864

[17] Jabbar, M.A., Aluvalu, R. (2017). RFAODE: A novel ensemble intrusion detection system. Procedia Computer Science, 115: 226-234. https://doi.org/10.1016/j.procs.2017.09.129

[18] Manzoor, I., Kumar, N. (2017). A feature reduced intrusion detection system using ANN classifier. Expert Systems with Applications, 88: 249-257. https://doi.org/10.1016/j.eswa.2017.07.005

[19] Qaiwmchi, N.A.H., Amintoosi, H., Mohajerzadeh, A. (2020). Intrusion detection system based on gradient corrected online sequential extreme learning machine. IEEE Access, 9: 4983-4999. https://doi.org/10.1109/ACCESS.2020.3047933

[20] Lv, L., Wang, W., Zhang, Z., Liu, X. (2020). A novel intrusion detection system based on an optimal hybrid kernel extreme learning machine. Knowledge-Based Systems, 195: 105648. https://doi.org/10.1016/j.knosys.2020.105648

[21] Zhao, J., Wang, Z., Park, D.S. (2012). Online sequential extreme learning machine with forgetting mechanism. Neurocomputing, 87: 79-89. https://doi.org/10.1016/j.neucom.2012.02.003

[22] Ali, M.H., Al Mohammed, B.A.D., Ismail, A., Zolkipli, M.F. (2018). A new intrusion detection system based on fast learning network and particle swarm optimization. IEEE Access, 6: 20255-20261. https://doi.org/10.1109/ACCESS.2018.2820092

[23] Tong, Q., Liu, S., Lu, Q., Chai, J. (2009). A fast algorithm of Moore-Penrose inverse for the symmetric Loewner-type matrix. In 2009 International Conference on Information Engineering and Computer Science, Wuhan, China, pp. 1-4. https://doi.org/10.1109/ICIECS.2009.5364014

[24] Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A. (2009). A detailed analysis of the KDD CUP 99 data set. In 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, pp. 1-6. https://doi.org/10.1109/CISDA.2009.5356528

[25] Amini, A., Saboohi, H., Herawan, T., Wah, T.Y. (2016). MuDi-Stream: A multi density clustering algorithm for

evolving data stream. Journal of Network and Computer Applications, 59: 370-385. https://doi.org/10.1016/j.jnca.2014.11.007

[26] Brownfield, B., Lemos, T., Kalivas, J.H. (2018). Consensus classification using non-optimized classifiers. Analytical Chemistry, 90(7): 4429-4437. https://doi.org/10.1021/acs.analchem.7b04399

[27] Hong, X., Chen, S., Harris, C.J. (2007). A kernel-based two-class classifier for imbalanced data sets. IEEE Transactions on Neural Networks, 18(1): 28-41. https://doi.org/10.1109/TNN.2006.882812

[28] Joshi, M.V. (2002). On evaluating performance of classifiers for rare classes. In 2002 IEEE International Conference on Data Mining, Maebashi City, Japan, pp. 641-644. https://doi.org/10.1109/ICDM.2002.1184018

[29] Lan, Y., Wang, Q., Cole, J.R., Rosen, G.L. (2012). Using the RDP classifier to predict taxonomic novelty and reduce the search space for finding novel organisms. PLoS One, 7(3): e32491. https://doi.org/10.1371/journal.pone.0032491

[30] Seliya, N., Khoshgoftaar, T.M., Van Hulse, J. (2009). A study on the relationships of classifier performance metrics. In 2009 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, NJ, USA, pp. 59-66. https://doi.org/10.1109/ICTAI.2009.25