RESEARCH

Open Access

# A multi-agent adaptive deep learning framework for online intrusion detection

Mahdi Soltani[1], Khashayar Khajavi[1], Mahdi Jafari Siavoshani[1] and Amir Hossein Jahangir[1*]

## Abstract

The network security analyzers use intrusion detection systems (IDSes) to distinguish malicious traffic from benign ones. The deep learning-based (DL-based) IDSes are proposed to auto-extract high-level features and eliminate the time-consuming and costly signature extraction process. However, this new generation of IDSes still needs to overcome a number of challenges to be employed in practical environments. One of the main issues of an applicable IDS is facing traffic concept drift, which manifests itself as new (i.e. , zero-day) attacks, in addition to the changing behavior of benign users/applications. Furthermore, a practical DL-based IDS needs to be conformed to a distributed (i.e. , multi-sensor) architecture in order to yield more accurate detections, create a collective attack knowledge based on the observations of different sensors, and also handle big data challenges for supporting high throughput networks. This paper proposes a novel multi-agent network intrusion detection framework to address the above shortcomings, considering a more practical scenario (i.e., online adaptable IDSes). This framework employs continual deep anomaly detectors for adapting each agent to the changing attack/benign patterns in its local traffic. In addition, a federated learning approach is proposed for sharing and exchanging local knowledge between different agents. Furthermore, the proposed framework implements sequential packet labeling for each flow, which provides an attack probability score for the flow by gradually observing each flow packet and updating its estimation. We evaluate the proposed framework by employing different deep models (including CNN-based and LSTM-based) over the CIC-IDS2017 and CSE-CIC-IDS2018 datasets. Through extensive evaluations and experiments, we show that the proposed distributed framework is well adapted to the traffic concept drift. More precisely, our results indicate that the CNN-based models are well suited for continually adapting to the traffic concept drift (i.e. , achieving an average detection rate of above 95% while needing just 128 new flows for the updating phase), and the LSTM-based models are a good candidate for sequential packet labeling in practical online IDSes (i.e. , detecting intrusions by just observing their first 15 packets).

**Keywords** Deep learning, Intrusion detection, Continual learning, Online IDS, Federated learning, Adaptable IDS, Zero-day attacks, Machine learning

## Introduction

Nowadays, the growth of cyber threats highlights the importance of security devices such as intrusion detection systems (IDSes). The network security analyzers use IDSes to monitor the network data, analyze them, and detect any kind of intrusions. There are mainly two categories of intrusion detectors: signature-based and machine learning-based (ML-based) (Labonne 2020).

The main advantage of ML-based IDSes over signature-based ones is the absence of the costly and time-consuming signature extraction process in the former. Consequently, ML-based IDSes, especially deep learning ones, are considered the new generation of IDS devices. The ability of deep learning-based (DL-based) IDSes to

*Correspondence:
Amir Hossein Jahangir
jahangir@sharif.ir
[1] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

auto-extract high-level features and classify different attack/benign traffic flows is their main advantage compared to the traditional ML-based IDSes. Moreover, due to the high-dimensional processing ability of DL models, the DL-based IDSes are good candidates for inspecting traffic content, as suggested in the recently proposed Deep Intrusion Detection (DID) framework (Soltani et al. 2022).

Many studies in the literature have applied deep learning methods to offline IDSes (Thakkar and Lohiya 2021; Soltani et al. 2023). Nevertheless, in this paper, we focus on simultaneously adapting DL-based IDSes for the following three practical challenges of online intrusion detection.

The first challenge is related to the continuous adaptability of a DL-based IDS to an organization's traffic since both attack and benign traffic patterns might encounter concept drift with the passage of time. For example, switching between semester and vacation times in the universities, adding new services to the web servers, and the emergence of new popular applications and protocols are examples of the content and behavior changes of benign user/traffic over time. Moreover, the characteristics and content of attack traffic also change continuously. This is due to the fact that the number of revealed vulnerabilities is increasing (NIST 2021), and additionally, novel attacks are devised on the existing vulnerabilities.

The second challenge in this scope stems from the distributed nature of anomaly detection. While DL-based IDSes have proved themselves to be accurate, there is still a need to suit them to a distributed architecture from two practical points of view:

1. It has been well discussed that relying solely on a single instance or sensor of an IDS will often yield inaccurate intrusion detection (Bhargavi and Vaidehi 2013). Large and complex network architectures require an ensemble of IDSes, each strategically placed in a specific location, ensuring optimal security and robustness (Iyengar 2020; Seresht and Azmi 2014). Furthermore, the collective knowledge of these scattered IDSes can be shared with a central unit to produce more comprehensive information and awareness regarding the network (Chai et al. 2021).
2. While relying on DL models, handling concurrent flows is not trivial. In most large networks, online traffic consists of many concurrent and interleaving flows. Each flow has a different start, end, and duration time. Consequently, considering a specific time window, the traffic consists of packets belonging to different flows. On the other hand, DL models need the sequence of a particular flow's packets to determine the flow label. As a result, these interleaving packets cannot be fed into a single DL model, and the flows should be separated beforehand.

The third and last challenge is that the performance of an online IDS depends on its ability to determine the correct flow label by inspecting fewer packets (i.e. , early attack detection). A reliable and fast attack detection can stop the attack earlier and mitigate its full impact on the target organization. Similar to the applicable traditional online IDSes, the aim is to determine the flow's label with some confidence per each packet arrival. When the IDS analyzes more flow packets, it increases its confidence score of the flow label. Security administrators can determine the thresholds of acceptable confidence scores according to the sensibility of the organization's assets.

To summarize, the contributions of this paper to make the DL-based IDSes more practical are as follows:

- We design a practical method for adapting DL-based IDSes to the network concept drift and new traffic patterns. A multi-stage deep continual learning algorithm is devised for this manner.
- We propose a novel multi-agent framework suitable for a distributed intrusion detection environment. The different agents can detect intrusions simultaneously (i.e. , in a multi-sensor environment) and also continuously adapt themselves to the traffic changes in their local sub-network. Furthermore, the agents can share and exchange their local knowledge through a proposed federated learning approach.
- We also take into account the requirements for a practical online IDS by analyzing the incoming traffic on the packet level while considering the flow concept (i.e. , determining the attack probability of a flow by observing each incoming packet)
  We conduct extensive experiments and analyses to demonstrate the effectiveness of the proposed framework from different perspectives. We show that by exploiting deep continual learning methods, the proposed framework can adapt the IDS to new patterns in the network with a relatively small number of new flows (i.e. , 128). Additionally, by utilizing LSTM models, the proposed framework is able to detect the intrusions of the state-of-the-art datasets CIC-IDS2017 and CSE-CIC-IDS2018 by just observing their first 15 packets. Furthermore, we show that the proposed framework performs well in a multi-agent environment, and different IDSes are able to effectively share their obtained attack knowledge, resulting in more reliable and robust intrusion detection.

The rest of this paper is organized as follows. In the next section, we first review the related works in DL-based intrusion detectors, deep continual learning methods, packet labeling, and deep federated learning. In section "Framework", we describe the proposed framework for online intrusion detectors. "Experimental evaluation" Section presents details of experiments, dataset preprocessing, and evaluation results of the framework implementations. "Discussion and future directions" Section discusses and analyzes the results of the experiments and explores some possible future directions. Finally, "Conclusion" Section concludes the paper.

## Related works

In this section, we briefly review both the previous approaches that have exploited deep learning for intrusion detection systems and also concepts that will aid us in designing our proposed online anomaly-based IDS (i.e. , sequence labeling, continual learning, and federated learning).

### Deep learning-based intrusion detection

Due to the capabilities of deep learning algorithms, including auto-extraction of suitable features, processing high dimensional data (e.g., content bytes of a flow), and supporting the time-series nature of the data, many studies have applied them in the scope of network intrusion detection. In the following, we review some of these research studies.

In Yin et al. (2017), the authors employ recurrent neural networks (RNN) for intrusion detection and evaluate binary and multi-class classification performance over the NSL-KDD dataset. In Vinayakumar et al. (2017), the intrusion detection application of different architectures of CNN-based DL models (e.g. , CNN, CNN-RNN, CNN-LSTM, and CNN-GRU) are evaluated using the KDDCup 99 dataset. Similarly, in Saba et al. (2022), the authors have exploited CNN models to design an Anomaly-Based IDS for IoT networks.

In Alghamdi and Bellaiche (2023), the authors use an ensemble-based deep learning technique to design intrusion detection systems for IoT networks. Their approach consists of an initial binary LSTM model that indicates whether the input traffic is normal or an attack. In the latter's case, a voting mechanism is conducted between three classifiers, i.e. LSTM, CNN, and artificial neural network (ANN) to perform multi-class classification on the input traffic and infer its corresponding attack type. Moreover, their proposed system processes the data in two modes: batch mode for training the models and stream mode to deal with the traffic stream in real-time.

Distributed intrusion detection using mobile agents is discussed in Riyad et al. (2019). Each mobile agent analyses the traffic and detects the threats independently. Consequently, this distribution operation evades the single point of failure problem. Additionally, they propose algorithms for reducing false positives by using inter-agent communications. They use the principal component analysis (PCA) algorithm to select the traffic features. Then, an ensemble of support vector machines (SVM), ANN, and RF algorithms classify the input traffic. The evaluation has been done on the KDD99 dataset.

In Abou El Houda et al. (2022), The authors propose an explainable IDS for IoT. Using Explainable Artificial Intelligence (XAI) techniques, they aim to design a framework in which the decisions of the Dl-based IDS are interpretable.

Employing reinforcement learning (RL), particularly deep Q-learning, in network intrusion detection systems is the main contribution of the proposed framework in Kim and Park (2019). The authors use two deep auto-encoder in their RL framework. One is for training the Q-learning model, and the other is for updating the model. The framework periodically applies mini-batch updates or Q-learning updates to make the model more adaptable to the continual evolution of cyber-attacks.

A deep learning self-adaptive approach is presented in Papamartzivanos et al. (2019). This approach consists of a transformation layer (the encoder) and a supervised learning deep model. It depends highly on the *change signals* from the network mapper modules. Such entities should determine any network changes, such as running services, available hosts, the operating system, and potential vulnerabilities. The approach learns a new auto-encoder model based on the stored traffic related to the signal period time and an archived initial labeled dataset. Then, it uses the encoder part as the new transformation layer by receiving the change signal. As a result, the model adapts itself to the new traffic distributions. A weakness of the mentioned approach is that, in many cases, receiving change signals from a network mapper is not a reasonable assumption for changing the model. For example, sensing a change in the network load may result from a DDoS/DoS attack. More generally, the model should not adapt its transformation layer according to the change signals triggered by attacks.

CSE-IDS (Gupta et al. 2022) focuses on the imbalanced nature of classes in the network security scope. It proposes a three-layer deep learning-based IDS and assumes three traffic categories: benign traffic, majority attacks with frequent samples, and minority attacks that represent infrequent ones. A cost-sensitive deep neural network (DNN) separates the benign traffic from the malicious ones in the first layer. The cost-sensitive loss function handles the imbalanced number of attacks and benign traffic. Then, a boosting ensemble, namely eXtreme Gradient Boosting, separates the suspicious samples into the benign class, different majority attack classes, and a single class representing all minority classes. Finally, an RF classifies the minority attacks into their respective classes. Besides, layer 2 and layer 3 use two oversampling techniques, namely, random oversampling and SVM-SMOTE. Their evaluation is based on the pre-extracted features of the NSL-KDD, CIDDS-001, and CIC-IDS2017 datasets.

In Wang et al. (2021), the authors integrate the stacked denoising auto-encoder (SDAE) (for reducing the noise of network traffic) and the extreme learning machine (ELM) (for increasing the IDS speed) as the SDAE-ELM model. This model is presented for a network intrusion detection system (NIDS). Besides, they propose to integrate the deep belief networks (DBN) (for extracting features from the log files of each host) and the softmax classifier (for determining the attack types) as the DBN-Softmax for the host-based intrusion detection system (HIDS). Their models use unsupervised data for the pretraining phase (learning the DAE and DBN layers of the NIDS and HIDS, respectively). Then, the fine-tuning phase uses supervised learning for training the SDAE-ELM and DBN-Softmax. The authors evaluate the NIDS based on the pre-extracted features of KDD99, NSL-KDD, UNSW-NB15, and CIDDS-001 datasets. Additionally, the AFDA-LD dataset is used to evaluate the HIDS model.

Cretu-Ciocarlie et al. (2009) propose an ensemble of n-gram based anomaly detectors (i.e. , micro-models). The voting scheme determines the predicted label of the evaluation traffic. They use time-delimited slices of the dataset for training the disjoint micro-models. Additionally, the model updates itself by generating new models according to the recently received traffic. The new micro-models take the place of the oldest ones. Accordingly, the intrusion detector can be adaptable to the traffic concept drift.

In Soltani et al. (2023), the authors propose DOC++ as a deep novelty-based classifier to detect not-seen traffic (both the zero-day attacks and new benign behaviors). In addition, using a joint deep clustering algorithm, enough pieces of each new novel class evidence are gathered and used in the supervised labeling process and corresponding updating phase. The update process that is responsible for learning the newly labeled concepts uses an active-passive strategy as the following steps:

1. Clone the existing active model to a passive model.
2. Run the cloned model's training, clustering, and post-training phases.
3. Migrate the traffic to the new model.

Even though the above-mentioned and many other similar research studies use terms like deep learning-based online/real-time NIDS, most of them solely focus on improving the detection speed and accuracy (i.e. , detection rate) of their models in comparison with the other approaches. Speed and accuracy are critical parameters in a real-world NIDS, but there are many other practical challenges in online NIDSes. For example, packet interleaving is an issue in real network traffic: packets of different flows are interleaved, and the proposed system should consider this challenge. Furthermore, network traffic concept drift is a prevalent phenomenon, and a practical IDS should adapt itself to these continuous changes. Additionally, a practical NIDS should determine the flow label upon receiving each packet and declare a confidence score for its decision. Measuring the performance of an online IDS is based on its capability to determine the true flow label with acceptable confidence by observing fewer packets of a flow.

However, to the best of our knowledge, the above challenges have not been investigated yet in most of the research studies related to online deep learning-based NIDSes.

### Sequence labeling

As mentioned before, an ideal characteristic that an IDS should possess is the ability to determine whether a flow is categorized as a possible threat in a gradual manner.

To be more precise, since the packets corresponding to a flow do not arrive simultaneously with the arrival of the first packet of a flow, the IDS presents an initial probability regarding the possibility of whether that flow is an attack. As time progresses, with the emergence of further packets, the IDS should produce a more accurate likelihood regarding that flow.

One should bear in mind that in conjunction with adjusting more to real-world scenarios, this scheme tends to be more efficient since there is no need to allocate time and computational resources to accumulate all packets of a flow (Hwang et al. 2019). For this purpose, the IDS needs to perform two essential tasks:

1. Produce labels for each packet individually, rather than yielding a single label for the flow.
2. Use temporal features for estimating the probability. In other words, the IDS should also consider the previous packets of a flow in the inference process for a new packet.

Due to their ability to preserve memory over sequential inputs, RNN networks, specifically LSTMs, have been widely exploited in several domains (since they excel in circumventing the vanishing gradient problem (Hochreiter and Schmidhuber 1997)). For instance, in the field of natural language processing (NLP), the research studies (e.g. , Ma and Hovy (2016) and Huang et al. (2015)) have used LSTMs to tackle sequence labeling tasks like *part of speech tagging* and *chunking*.

Similarly, some researchers have utilized LSTMs for network traffic classification. In Hwang et al. (2019), network traffic classification is done at the packet level by mapping this task to a sentence classification problem in NLP. This approach considers packets and their headers as sentences and words, respectively. The headers of a packet are used to construct a 64-dimensional word vector, which is used as the input for an LSTM model to perform the classification. In Lopez-Martin et al. (2017), although several networks comprising LSTM segments have been designed to classify packets sequentially, they require the entire flow for classification.

In Ansari et al. (2022), the authors employ deep models with gated recurrent units (GRU) to generate alerts for malicious sources. In their approach, a model is trained to learn the dependencies between previously generated alerts and predict future alerts for a malicious source.

In Gao et al. (2019), both a many-to-many and a many-to-one LSTM are designed to address intrusion detection systems for the supervisory control and data acquisition (SCADA) protocol, and their results are compared.

Since a many-to-many LSTM model can classify packets individually and sequentially, our approach utilizes this technique as one of the base DL models inside the proposed adaptive framework. Furthermore, LSTMs can work with variable length input sequences (i.e. , flows) (Lee et al. 2021), thus making them more efficient and practical.

## Deep continual learning

In the proposed framework, our primary attention has been devoted to a specific family of online learning algorithms named *Continual learning (CL)*, defined as the ability to learn new tasks that arrive sequentially by efficiently exploiting the knowledge acquired in previous tasks (Van de Ven and Tolias 2019). The main dilemma in CL is a phenomenon called *catastrophic forgetting*, characterized by the model performing poorly on the old tasks when trained on the new ones.

In recent years, valuable methods have been proposed to mitigate the problem of catastrophic forgetting for continual learning. In the following, we will review the two main categories related to our research.

### *Continual learning based on regularization*

A prevalent technique for continual learning is to exploit different regularization terms and constraints to avoid detrimental weight changes when training on new tasks. One naive solution would be to use an L2-Regularization term, but this approach will prevent the model from efficiently learning new tasks.

A ground-breaking technique known as elastic weight consolidation (EWC) is proposed in Kirkpatrick et al. (2017), which uses a regularization term based on the diagonals of a set of Fisher information matrices to reduce the plasticity of the weights of greater importance to the previous tasks. The values on the diagonal of the Fisher information matrix measure the amount of information that the training samples provide for each parameter (i.e. , weight) of the trained DL model, thus representing an importance factor for each weight. To be more precise, based on the definitions in Martens (2020); Van de Ven and Tolias (2019), the $i_{th}$ element of the Fisher information matrix diagonal is proportional to the expected value (i.e. , based on the training data distribution) regarding the Hessian of the model's output with respect to the $i_{th}$ weight. Consequently, a high Hessian for a weight signifies the plasticity of the gradient of the model output based on that weight. Note that in a given task, the weights obtained from the training (i.e. , optimization) procedure often represent a local minimum for the desired loss function. As a result, changes made to parameters with a high hessian would result in a substantial drift from that minima, resulting in a performance decline of the model on the mentioned task.

Since in EWC, the number of quadratic terms would increase linearly with the advent of new tasks, online EWC is proposed in Schwarz et al. (2018), which uses a single Fisher information matrix and updates it each time it learns a new task. Another method named synaptic intelligence (SI) is proposed in Zenke et al. (2017). Instead of the Fisher information matrix, it tries to compute an online importance factor for each weight, which describes its importance across all previously learned tasks.

### *Continual learning based on expansion*

A number of approaches focus on the main idea to expand the network capacity by adding new layers or extending the previous layers to accommodate the knowledge associated with the new task (Rusu et al. 2016; Yoon et al. 2017; Jain and Kasaei 2021).

Progressive neural networks (PNN), as described in Rusu et al. (2016), are models comprised of columns that each preserve a connection with all of their predecessors. Each column can be considered an individual network with a fixed architecture that includes blocks representing a network layer. A new column is added to the model with the arrival of new data, and training is done via freezing the previous columns. The main drawback of this approach is the constant, substantial increase of the network size for every new task, thus making it infeasible to maintain in the long run. Several methods have been proposed to circumvent this flaw by expanding the network as efficiently as possible.

As described in Yoon et al. (2017), dynamically expandable networks (DEN) try to design an architecture that dynamically increases the network capacity when faced with new training data. At its core, a DEN first aims to modify the current network to perform well on the new data. In case of failure, each layer will be augmented by adding a fixed number of nodes, and the whole expanded network will be trained on the new data with the group sparse regularization (Scardapane et al. 2017). Due to this regularization term, some added nodes will be considered redundant after training and be pruned, thus preventing the network from becoming too large. In the end, if the weights of some previous nodes experience significant alteration during training, a duplicate of those nodes will be added to the network, and the network will be trained again.

One recent variation of DEN named 3d_DEN is proposed in Jain and Kasaei (2021) for continual multi-class classification. Each task represents a new class, and a corresponding output node will be added to the network. In this approach, when training the network after expansion, only the added segments are trained, and the previous parts of the network are frozen, thus protecting them from catastrophic forgetting.

Since DEN and its variations rely on multiple sparse regularization terms, the high number of hyperparameters will make tuning the ideal network arduous. For this means, reinforced continual learning (RCL) is introduced in Xu and Zhu (2018). In this method, for expanding the network, an LSTM network is used via reinforcement learning and policy gradient to predict the optimal number of nodes that should be added to each layer, with respect to both the detection rate and size of the network.

Although the approaches mentioned above try to expand the network as efficiently as possible, the network's size will still grow after each task, which is considered a drawback in the long run. An approach for fully compressing the network after the expansion is proposed in the "regularize, expand, and compress" (REC) framework (Zhang et al. 2020). Similar to RCL, REC exploits reinforcement learning (AutoML Sutton et al. (2000)) to expand the network. The whole network is trained on the new task with regularizations based on multi-task learning and the Fisher information matrix. After that, the compression is done using the knowledge distillation approach (Hinton et al. 2015) and soft labels; thus, the network ia reshaped to its original architecture.

### Deep federated learning

Federated learning (FL) is an ML approach for training a model by utilizing distributed devices that contribute to the training process based on their local data. Both synchronous and asynchronous methods have been proposed to this end, but since the nature of our problem requires an asynchronous setting, we will mainly focus on the latter.

In Gimpel et al. (2010), an asynchronous distributed optimization algorithm is designed, which despite a minor error in the training procedure, performs well when evaluated on NLP tasks. In Xie et al. (2019), an asynchronous federated learning scheme is proposed in which each worker independently trains a model with a regularization term that prevents any significant drift from the main-model. Also, the main-model is updated via weighted averaging with the worker model.

In Chai et al. (2021), a federated learning system is designed based on dividing the clients into different groups called tiers. In this approach, a tiering module partitions the clients into tiers based on their performance (e.g., response latency). In each tier, the updating process is synchronously performed by the tier members via gradient computation and optimization. Furthermore, The main-model gets updated asynchronously based on the weighted averaging of the models obtained from the tiers.

In Diro and Chilamkurti (2018), the authors propose a distributed attack detection mechanism for IoT based on fog computing (Yi et al. 2015). In their approach, the fog nodes are responsible for locally training DL models that act as intrusion detectors at the network edge. Furthermore, a coordinator master is used to propagate the local updates and parameters between the fog nodes, and this optimization procedure is conducted via distributed SGD.

These proposed schemes and designs provide a solid foundation for designing our multi-agent framework as described in "Framework" Section.

## Framework

In this section, our proposed online anomaly-based intrusion detector is described. This framework aims to address three of the current main challenges for an applicable and realistic intrusion detection system:

- *Continuos adaptation to new traffic* The first challenge relates to the emergence of new attacks and benign user/traffic behavior changing over time. To address this continuous adaption challenge, we use deep continual learning methods, as discussed in "Continuous adaption to network concept drift" Section.

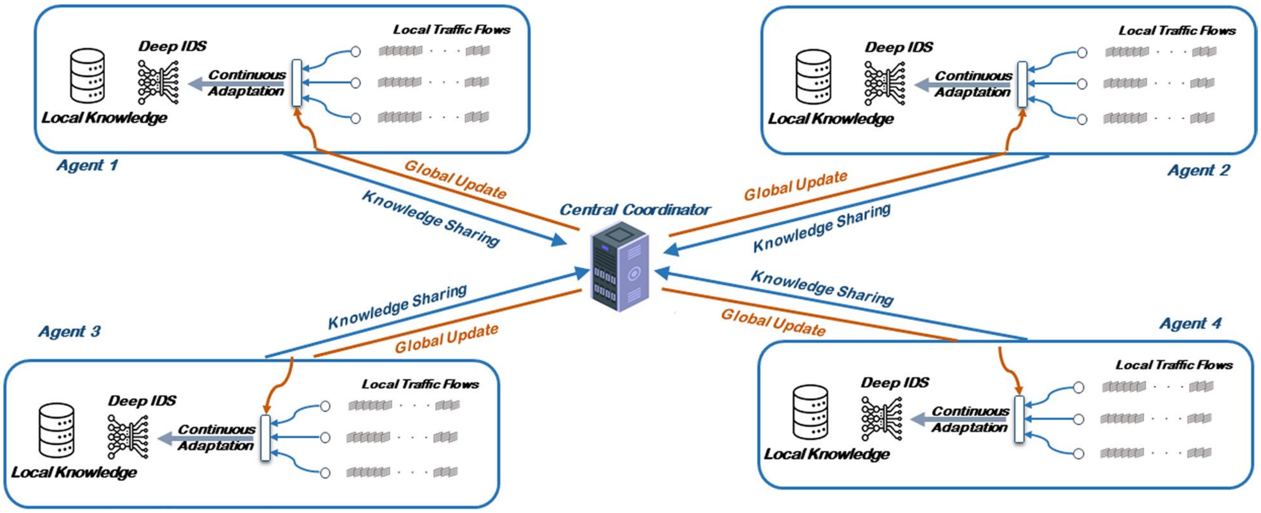


**Fig. 1** An overview of the proposed multi-agent IDS framework

- *Online intrusion detection* As mentioned before, another challenge of an online IDS is making a progressive decision about a flow by observing the stream of its packets. The reason is that the best online IDSes are the ones that can detect an attack with fewer packets. In other words, threat detection should be done before the attacker completes the attack.
- *Multi-agent architecture* The third challenge of an online IDS corresponds to the interleaving nature of the packets of different flows in the network traffic. In particular, to address this issue and to consider a high throughput network, we propose to use a distributed architecture for handling all packets of each flow in an agent. Each agent implements a sub-model of the main DL model in this architecture. Then, to update the model, the distributed sub-models are aggregated in the main deep anomaly-based model.

Our proposed framework aims to collectively address the mentioned practical challenges, as mentioned in the following sections.

## Overview

The proposed framework is a multi-agent IDS depicted in Fig. 1. Each agent can be strategically placed in a different section of a network (or, as discussed in "Multi-agent IDS" Section , these agents can be dispersed in a geo-distributed manner on a global scale), and individually perform intrusion detection on their associated area using their deep IDSes. Each agent independently detects intrusion within its assigned area using deep IDS capabilities. This distributed setup ensures scalability for high throughput and facilitates knowledge sharing. Detection methods can operate at both flow and packet levels, depending on the chosen deep model architecture for the IDS.

Furthermore, each agent continuously adapts itself to the new flows and patterns in its local sub-network to update its local benign/attack knowledge. In "Continuous adaption to network concept drift" Section , we analyze and propose an optimal strategy for updating a single deep IDS. The subsequent description of the proposed multi-agent architecture (see "Multi-agent IDS" Section) outlines a system where agents exchange local knowledge through a central coordinator. This coordinator accumulates shared knowledge from all agents and updates them regularly. To be more precise, each agent can receive an update from the central coordinator and update its knowledge accordingly. Conversely, the central coordinator can also receive an update from each agent and share it with other agents. The details of these procedures will be provided in detail in "Multi-agent IDS" Section.

## Continuous adaption to network concept drift

With the advent of a new attack, we expect our IDS to conform itself to the new data, and while preserving its ability to detect previously learned abnormalities, it should extend its knowledge to recognize the new one. To achieve this goal, in this section, we propose a continual learning-based algorithm that best satisfies the needs and constraints of an IDS.

### *General IDS model architecture*

The proposed framework assumes that the DL models used in an IDS are comprised of the base and dense parts. The base part usually consists of either LSTM or convolutional (CNN) layers and is followed by the dense part that comprises multiple fully-connected (FC) layers. Ensuing from the deductions made in Jain and Kasaei (2021) and Yosinski et al. (2014), the base parts serve as a pre-trained and frozen section of our network, whereas the FC layers will change and train continually on new anomalies. This approach has two main benefits:

1. The base part will determine the general features of our inputs (may it be flows or individual packets, as described in "Experimental evaluation" Section) and learn useful representations that facilitate the classification procedure, which is an integral phase in many DL-based IDSes (Choi et al. 2020). On the other hand, the FC layers will both learn new specific features and better classify the general features by training on new data.
2. Each continual training will require less computation since the pre-trained network will not be involved.

### *Proposed continual learning approach*

The proposed continual learning algorithm, similar to those mentioned in "Deep continual learning" Section, is based on the expansion approach, i.e. , each FC layer is augmented with a set of nodes. More specifically, each added node will have inputs from all nodes in the previous layer (including the augmented ones), but its outputs will only be connected to the new nodes in the next layer, thus allowing it to capture new features while not altering the nodes from older tasks (i.e., attacks in the security scope) (Jain and Kasaei 2021). In this expansion phase, based on prior work, there are two options:

- Adding a fixed number of nodes to each layer (denoted as $k$) (Yoon et al. 2017; Jain and Kasaei 2021).
- Designing a controller for configuring the optimal numbers of nodes for each layer based on RL approaches (which have been used prevalently in the network anomaly detection scope (Adawadkar and Kulkarni 2022)). To be more precise, each time the controller generates the number of nodes corresponding to each layer, it receives a reward and updates itself via policy gradient techniques. This process is repeated several times until the best result is achieved (Xu and Zhu 2018; Zhang et al. 2020).

Although the latter approach tends to discover a more efficient expanded network, our analysis indicated that the former would better suit our domain, as explained below.

First, the latter approach requires a substantial amount of time to find the optimal network, which is a significant flaw since the IDS is expected to perform on a real-time basis. Each time the controller predicts the number of added nodes, training has to be conducted on the corresponding child network to yield a reward for the controller. This process might be carried out several times to yield the best result. On the other hand, using a fixed number of nodes will require training the expanded network only once.

Second, as the expansion procedure is ensured by compression (as described in "Multi-agent IDS" Section), finding the minimum number of nodes in each layer is not necessary. Also, in contrast to Yoon et al. (2017) and Jain and Kasaei (2021), there is no need to perform $l_1$ -norm or group sparsity regularization and tuning their corresponding hyperparameters for training on the new task since compressing the network will not rely on this technique, as explained in the next section.

After adding $k$ nodes to each FC layer, the training phase consists of two sections:

1. The nodes pertaining to the previous tasks are frozen, and while only the added nodes are kept trainable, training is done on the data of the new task (i.e. , new attack). As mentioned above, there will be no need for any kind of regularization. Hence, the training can be described as optimizing a single loss function, i.e. ,

$$\min_{W^{\mathrm{Add}}} \mathcal{L}\left(W^{\mathrm{Add}} \middle| W^{\mathrm{Prv}}, \mathcal{D}_{\mathrm{train}}\right), \tag{1}$$

where $\mathcal{L}$ is our desired loss function (e.g. , binary cross-entropy), $W^{\mathrm{Add}}$ describes the weights of the newly added nodes to the network, $W^{\mathrm{Prv}}$ represents the (frozen) weights of previous nodes, and $\mathcal{D}_{\mathrm{train}}$ is the dataset comprising the new traffic for training.
2. After the first step, the expanded model's performance is measured on a validation set $\mathcal{D}_{\mathrm{val}}$, and in the case its detection rate is below a preset threshold $\tau$, instead of solely training the added nodes, the whole network is trained under the following equation (Zhang et al. 2020)

$$\min_{W^{\mathrm{Exp}}} \left[ \begin{array}{r} \mathcal{L}(W^{\mathrm{Exp}}|\mathcal{D}_{\mathrm{train}}) + \lambda_1 \sum_{i=1}^{\mathcal{N}_{\mathrm{params}}} \mathcal{F}_{ii}^{\mathrm{Prv}}(\theta_i^{\mathrm{Exp}} - \theta_i^{\mathrm{Prv}}) + \\ \lambda_2 \left\| [W^{\mathrm{Exp}}; W^{\mathrm{Prv}}] \right\|_{2,1} + \lambda_3 \left\| W^{\mathrm{Add}} \right\|_1 \end{array} \right], \tag{2}$$

where $\mathcal{N}_{\text{params}}$ is the number of weights in the model prior to expansion, $W^{\text{Prv}} = \{\theta_i^{\text{Prv}}\}_{i=1}^{\mathcal{N}_{\text{params}}}$, as introduced above, are the weights of the model before expansion, $W^{\text{Add}}$ are the weights of the newly added nodes, and $W^{\text{Exp}}$ are all the weights of the expanded model. In the expanded model, using the Fisher information matrix diagonal, the term $\sum_{i=1}^{\mathcal{N}_{\text{params}}} \mathcal{F}_{ii}^{\text{Prv}}(\theta_i^{\text{Exp}} - \theta_i^{\text{Prv}})$, is enforced on the weights corresponding to the previous task to avoid catastrophic forgetting (as discussed in Setion 2.3.1). The term $\|[W^{\text{Exp}}; W^{\text{Prv}}]\|$ is an $l_{2,1}$-norm regularization (Zhang et al. 2020) (i.e. , $\|\|\|W^{\text{Exp}}\|_2, \|W^{\text{Prv}}\|_2\|_1$) term derived from multi-task learning, aiming to learn the shared representations between the weights of the model prior to and after expansion, and $\|W^{\text{Add}}\|_1$ is a sparsity-inducing regularization term (Gong et al. 2012) imposed solely on the new nodes for efficient learning of the features specific to the new traffic.

Furthermore, for practically computing the diagonal of the Fisher information matrix, we employ the method proposed in Van de Ven and Tolias (2019). Namely, for the $i_{th}$ element of the diagonal we have:

$$\mathcal{F}_{ii} = \frac{1}{|S|} \sum_{(x,y)\in S} \frac{\delta \log p(Y = y|x, \theta)}{\delta \theta_i}, \tag{3}$$

where $\mathcal{F}_{ii}$ is the $i$th element of the Fisher information matrix diagonal (i.e. , corresponding to the $i$th weight of the model) and $S$ is the data set used for training the model. Furthermore, $\theta$ are the weights of the model after training, $(x, y)$ represents any labeled sample from $S$, and $p(Y = y|x, \theta)$ is the produced probability by the model for the correct class label. Moreover, a proposed strategy is discussed for updating the Fisher information diagonal throughout the continual learning procedure in "Multi-Agent IDS" Section.

Algorithm 1 describes the proposed continual learning procedure.

**Algorithm 1** Continual Learning Algorithm.

```
Input :
    $\mathcal{D}_{\text{train}}$ : New dataset to train on
    $\mathcal{D}_{\text{val}}$ : Validation dataset
Output :
    $W^{\text{Exp}}$ : The weights of the expanded network

Add $k$ units to all layers
Obtain $W^{\text{Exp}}$ by training the network based on (1)
if detection rate of $W^{\text{Exp}}$ model on $\mathcal{D}_{\text{val}} < \tau$ then
    Obtain $W^{\text{Exp}}$ by training the network based on (2)
end if
```

### *Data sampling*

In some cases, incrementally training solely on a new set of data samples from unknown traffic might make our model biased towards new traffic, which will be an instance of catastrophic forgetting. As suggested in Jain and Kasaei (2021); Soltani et al. (2023), with the advent of new data, we will constitute a training set that possesses the new data in conjunction with samples corresponding to the previous attacks and benign flows that the model has been previously trained on.

To implement this approach, the collective number of data samples belonging to the previous attacks should be equal to the number of the new attack samples. Since our model is a binary classifier between benign and attack flows, the number of benign samples should be equal to the total number of attacks (i.e. , including the old and new attacks). Algorithm 2 describes this procedure in detail.

One should bear in mind that sustaining all the previous instances is evidently unfeasible for practical scenarios. However, as discussed in "Discussion and future directions" Section, the proposed updating strategy is able to adapt the model to new traffic with a small number of instances. Consequently, it suffices to preserve a limited number of instances from previous flows to prevent bias (i.e. , set a threshold for the maximum number of previous benign/attack samples). Furthermore, another practical approach for reproducing previous samples would be using Generative Adversarial Networks (GAN) that can support continuous updating to new data (Andresini et al. 2021; Liang et al. 2018; Seff et al. 2017; Varshney et al. 2021).

**Algorithm 2** Data Sampling Algorithm.

```
Input :
    $\mathcal{R}$ : raw samples of the new traffic
Output :
    $\mathcal{D}_t$ : augmented dataset for the new traffic (i.e., new task)

$\mathcal{B}$ = dataset containing benign samples
$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{t-1})$: datasets of previous attacks
$\mathcal{D}_t = \mathcal{R}$
Split $\mathcal{R}$ to $\mathcal{A}_t$ (new attack samples) and $\mathcal{B}_t$ (new benign samples)
$s_A = \text{len}(\mathcal{A}_t)$
for $i = 1, 2, \ldots, t-1$ do
    Choose $s_A$ samples from $\mathcal{A}_i$ and add to $\mathcal{D}_t$
end for
$s_B = \text{len}(\mathcal{D}_t) - \text{len}(\mathcal{B}_t)$
Choose $s_B$ samples from $\mathcal{B}$ and add to $\mathcal{D}_t$
```

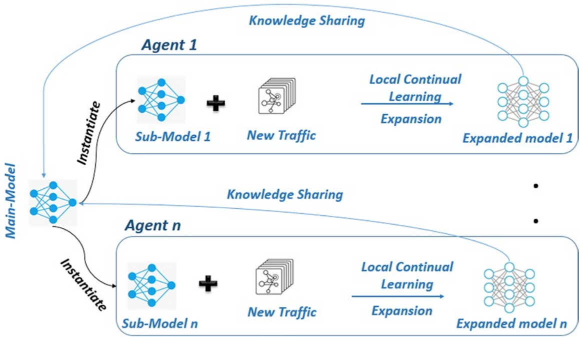


**Fig. 2** An overview of how different agents function in the proposed architecture. *Agent*$_1, \ldots,$ *Agent*$_n$ (e.g. , different IDSes) contain each a sub-model which is initialized with the weights of the main-model. After learning a new anomaly based on Algorithm 1, the main-model is updated via knowledge sharing (i.e. , federated distillation)

## Multi-agent IDS

To address the distributed requirements of an IDS (as discussed in "Introduction" Section), we have proposed to employ a multi-agent federated learning architecture. Each agent is assigned a part of the traffic flows, captures the new abnormalities and benign traffic concept drift based on the assigned traffic, and then updates itself.

To be more precise, each agent consists of a *sub-model* that continually learns new traffic behavior. Once an agent has finished its continual learning procedure, it asynchronously updates the *main-model* through knowledge distillation (Hinton et al. 2015). Thus, the collective knowledge obtained and shared by all the agents will be incrementally integrated into the main-model.

An overview of the proposed federated learning architecture is shown in Fig. 2. The main-model acts as the central coordinator (as mentioned in "Federated learning" Section), which gathers the collective knowlege of the agents and updates the agents accordingly. Each agent initializes its sub-model weights with the main-model's latest weights prior to its continual learning procedure. After the learning phase, in order to update the main-model, each agent engages in an asynchronous optimization with the loss function using a combination of the logits (i.e. , the input vector of the final softmax layer as the soft labels) and the actual labels (i.e. , hard labels). In addition, in order to prevent catastrophic forgetting, a regularization term based on the diagonal of the Fisher information matrix of the main-model is exploited. Thus, in order to update the main-model through knowledge distillation, we propose the $\ell$th agent computes and sends to the main-model the gradients of the following loss function

$$f_{\mathsf{dist}}(W_{\mathsf{main}}) = \mathcal{L}(W_{\mathsf{main}}; \mathcal{D}_\ell) + \mathcal{L}_{\mathsf{kd}}(W_{\mathsf{main}}; \mathcal{Z}_\ell) + \lambda \sum_{i=1}^{\mathcal{N}_{\mathsf{params}}} \mathcal{F}_{ii}(\theta^i_{\mathsf{main}} - \theta^i_{\mathsf{init}}), \tag{4}$$

where again, $\mathcal{N}_{\mathsf{params}}$ is the total number of parameters in the main-model, $W_{\mathsf{main}} = \{\theta^i_{\mathsf{main}}\}_{i=1}^{\mathcal{N}_{\mathsf{params}}}$ is the new weights of the main-model, $W_{\mathsf{init}} = \{\theta^i_{\mathsf{init}}\}_{i=1}^{\mathcal{N}_{\mathsf{params}}}$ and $\mathcal{F}$ are the weights and the Fisher information diagonal of the main-model prior to distillation, $\mathcal{D}_\ell$ is the training data observed by the $\ell$th agent, and $\mathcal{Z}_\ell$ are the logits received through the expanded model.

In order to asynchronously update the main-model, an agent first acquires the latest version of $W_{\mathsf{init}}$ and $\mathcal{F}$ from the main-model. Then, the main-models' parameters are updated through the following update rule (Gimpel et al. 2010)

$$W'_{\mathsf{main}} = W_{\mathsf{main}} - \mu \sum_{\ell \in M} \nabla_\ell (f_{\mathsf{dist}}(W_{\mathsf{main}})), \tag{5}$$

where $\nabla_\ell(f_{\mathsf{dist}}(W_{\mathsf{main}}))$ is the gradient of $f_{\mathsf{dist}}(W_{\mathsf{main}})$ computed by the $\ell_{th}$ agent on its own batch. Also, $M$ is the set of agents that have sent a gradient in the time interval between the last two updates.

Once an agent's federated distillation procedure comes to an end, it also computes the Fisher information matrix diagonal based on the latest version of $W_{\mathsf{main}}$ and its own data, using Eq. 2. This matrix is sent to the main-model, updating the main Fisher information matrix diagonal based on the following equation

$$\mathcal{F}'_{\mathsf{main}} = \alpha \mathcal{F}_{\mathsf{main}} + (1 - \alpha)\mathcal{F}_{\mathsf{agent}}, \tag{6}$$

where $\mathcal{F}'_{\mathsf{main}}$ is the new Fisher information matrix diagonal of the main-model, $\mathcal{F}_{\mathsf{main}}$ is the diagonal of the previous Fisher information matrix of the main-model, $\mathcal{F}_{\mathsf{agent}}$ is the Fisher information matrix diagonal sent by the agent, and $\alpha$ is an aggregation weight.

Based on the proposed federated learning architecture, the procedure that an agent undertakes to update the main-model is described in Algorithm 3. Note that the proposed approach has the practical benefit of not expanding the main-model; thus, the main-model will not grow infinitely and can be practically applied in the long term without needing additional memory. Furthermore, the federated distillation procedure also functions as a compression mechanism for the agents. As a result, an agent's expanded model can be replaced with the updated main-model at the end of this process.

**Table 1** Summary of the parameters used in desinging the proposed architecture and their influence

| Parameter | Influence |
|---|---|
| $\lambda_1$ | Preventing catastrophic forgetting during update in Eq. 2 |
| $\lambda_2$ | Amount of learned shared representations prior and after update in Eq. 2 |
| $\lambda_3$ | Regularization of expanded parts of the network in 2 |
| $\lambda$ | Preventing catastrophic forgetting during update of main-model in Eq. 4 |
| $\mu$ | Amount of change applied to the main-model during update in Eq. 5 |
| $\alpha$ | Amount of change applied to the Fisher diagonal after update in Eq. 6 |

**Algorithm 3** Agent Learning Procedure.

```
Input :
    $\mathcal{R}$ : Flows pertaining to the new traffic
Obtain $\mathcal{D}_t$ by using $\mathcal{R}$ as input to Algorithm 2.
Split $\mathcal{D}_t$ to $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ for training and validation.
Get $W_{\text{main}}$ and $\mathcal{F}$ from the main-model.
Obtain $W_{\text{Exp}}$ from Algorithm 1 using $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $W_{\text{main}}$, and $\mathcal{F}$.
Update $W_{\text{main}}$ and $\mathcal{F}$ from the main-model (in case that other agents have updated the main-model).
Set $W_{\text{init}} = W_{\text{main}}$
for each training step do
    Get mini-batch and labels from $\mathcal{D}_{\text{train}}$ and the logits from $W^{\text{Exp}}$.
    Compute the gradient of (4) and send it to the main-model.
    Wait for the main-model to send $W_{\text{main}}$
end for
Compute $\mathcal{F}_{\text{agent}}$ based on $\mathcal{D}_{\text{train}}$ and sent to the main-model in order to compute (3.3).
```

In the end, in Table 1, we summarize the parameters used in our architecture to further clarify the design of the proposed online anomaly-based deep IDS.

## Experimental evaluation

This section describes the evaluation details of the proposed framework to reproduce the experiments. First, the Experimental details, including evaluation infrastructure, the preprocessing phase, evaluated datasets, and hardware specifications, are described ("Experimental details" Section). These are the common infrastructure for all the following experiments. Then, different deep online anomaly detectors' implementations are evaluated ("Deep Adaptive Anomaly Detectors" Section). Next the proposed distributed architecture for implementing a DL-based NIDS is evaluated ("Federated learning" Section). Finally, in "Early attack detection through packet assessment" Section we evaluate the the online IDS challenge of progressively determining the flow label upon each packet's arrival.[1]

### Experimental details

#### *Evaluation infrastructure*

In this work, the deep intrusion detection (DID) framework, introduced in Soltani et al. (2022), is used in the preprocessing phase of all experiments. The DID approach is selected for its ability to self-extract appropriate features and the capability of detecting a wide range of attacks, including content-based ones like SQL injection and Heartbleed attacks. The content-based attacks are the main segment of the threats with high malicious impacts on the targeted organizations. Consequently, this preprocessing phase can significantly affect the applicability of the proposed framework.

#### *Datasets*

As the DID approach is designed for the applicable IDSes, it requires the pure content of traffic flows (e.g. , in PCAP format). Consequently, the scope of applicable datasets for evaluating deep IDSes is constrained to those including the labeled traffic content. The privacy issues restrict the dataset developers from publishing the details of the real network traffic. As a result, datasets with entire traffic content such as DARPA 1999 (Lippmann et al. 2000) (which is the base of the KDD99 (KDD 2021) and NSL-KDD (Tavallaee et al. 2009) dataset), CIC-IDS2017 (Sharafaldin et al. 2018), and CSE-CIC-IDS2018 (CSE-CIC 2021) are all generated in an emulated network.

In this work, to properly evaluate the proposed framework, we have used the more up-to-date datasets (CIC-IDS2017 and CSE-CIC-IDS2018), which have

**Table 2** The system specification of the experimental environment

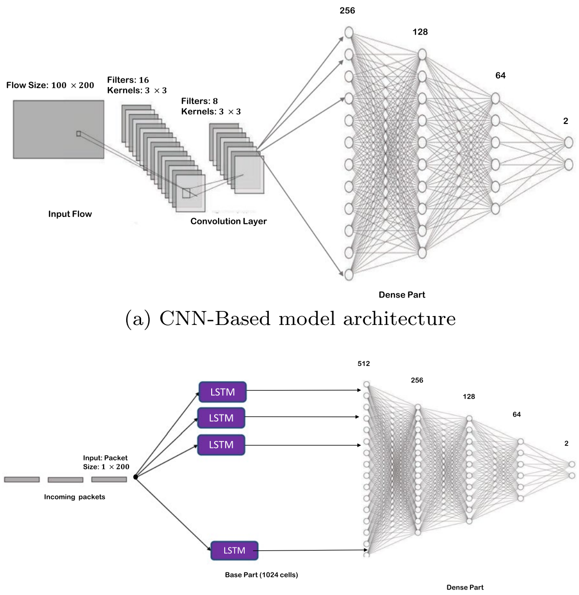
| | |
|---|---|
| OS | Ubuntu Version 20.04.3 LTS with Kernel 5.4.0-81-generic |
| CPU | Intel(R) Core(TM) i7-6900K 3.20GHz with 16 virtual cores |
| RAM | 32 GB |
| GPU | GeForce GTX 1080 Ti |
| GPU frame buffer | 8 GB |

[1] The implementations of all evaluated models are available at https://github.com/INL-Laboratory/Continual-Federated-IDS.

implemented the more recent attack types like SSH brute force botnet, DoS, DDoS, web, and infiltration attacks. Most importantly, they contain content-based attacks like SQL injection, XSS attacks, and Heartbleed. Additionally, benign profiles are extracted based on the abstract behavior of 25 users over the HTTP, HTTPS, FTP, SSH, and email protocols. Besides detecting the anomalies with a high detection rate, an IDS should also produce low false-negative rates. As a result, in addition to anomaly flows, we use benign traffic in our experiments.

In order to prepare the data to feed into the DL models, we use a packet size of 200 bytes and a flow size of 100 packets, resulting in a 20,000-dimensional input vector (which we will refer to as the flow matrix). This selection is based on the analysis of the correspondent datasets investigated in Soltani et al. (2022). To implement the proposed framework, we employ the Keras library (Chollet 2017) with Tensorflow (Abadi et al. 2015) as its backend. The characteristics of our experimental environment are shown in Table 2.

Throughout the experiments, we have exploited the labeled flows of the CIC-IDS2017 and CSE-CIC-IDS2018 datasets. These datasets have been split so that 64% of the overall flows are used for training the models in the experiments, 16% for validating the best hyperparameters, and 20% for testing and evaluating the different approaches in the proposed framework.



(a) CNN-Based model architecture

(b) LSTM-Based model architecture

**Fig. 3** The CNN-Based and LSTM-Based model architectures used in our experiments. The CNN-based models take an entire flow as an input, whereas the LSTM-based models work on the packet level and process the packets of a flow one by one

**Table 3** The hyperparameters used in the evaluations

| Parameter | Usage | Search space | Chosen value |
|---|---|---|---|
| $\lambda_1$ | Equation 2 | $[10^{-6}, 10^{-3}, 1, 10]$ | 1 |
| $\lambda_2$ | Equation 2 | $[10^{-6}, 10^{-3}, 1, 10]$ | $10^{-3}$ |
| $\lambda_3$ | Equation 2 | $[10^{-6}, 10^{-3}, 1, 10]$ | $10^{-3}$ |
| $\lambda$ | Equation 4 | $[10^{-6}, 10^{-3}, 1, 10]$ | 1 |
| $\alpha$ | Equation 3.3 | [0.4, 0.6, 0.8, 0.9] | 0.9 |
| $\mu$ | Equation 3.3 | [0.1, 0.5, 1] | 1 |
| Batch size | Initial Training | [8, 16, 32, 64, 128] | 32 |
| Epochs | Initial Training | [30, 50, 80] | 50 |
| Batch size | Continual Learning | [8, 16, 32] | 16 |
| Epochs | Continual Learning | [10, 20 , 30, 40] | 20 |
| $k$ | Continual Learning | [5, 10 , 12, 15] | 10 |
| Batch size | Federated Learning | [8, 16, 32] | 16 |
| Epochs | Federated Learning | [10, 20, 30 , 40] | 20 |

### *Model architectures*

We evaluate our proposed framework with two different architectures (i.e., CNN and LSTM). In the following, we describe each architecture's base and dense parts, as discussed in "Continuous adaption to network concept drift" Section.

In the first architecture (CNN-based), the base part comprises two consecutive 2D convolution layers with 8 and 16 filters, a 3 × 3 kernel size, a stride of 1 × 1, and no padding. The dense part comprises four layers with 256, 128, 64, and 2 neurons, respectively.

The second architecture (LSTM-based) consists of a single, many-to-many LSTM layer with 1024 cells as the base part. Many-to-many LSTMs can generate separate outputs for each of the corresponding sequential inputs. The dense part has five layers with 512, 256, 128, 64, and 2 neurons.

Note that the above-mentioned architectures use different input vectors. The first architecture uses the entire flow matrix as the input (i.e. , the input is a matrix of size 200 × 100). In contrast, the second architecture takes individual packets as the input (i.e. , the input is a vector of size 200, however, a sequence of 100 such vectors are fed into the model) and estimates a probability for the flow label after processing each packet. Consequently, the second architecture is more applicable to early attack detection in IDSes. Figure 3 illustrates these two different architectures.

The ReLU activation function and a dropout of 0.2 are used in both architectures for all but the last layer. In the last layer of both architectures, a softmax function is implemented to compute the benign/anomaly probabilities. Finally, the Adam optimizer is used for training the different DL models.

### *Hyperparameter settings*

We employ a grid search procedure to obtain the best values of hyperparameters, including training batch size, epochs, and coefficients of the regularization terms. Moreover, for the continual learning algorithm, we adhere to the method used in Jain and Kasaei (2021) for determining the number of added nodes (i.e. , increasing the value of $k$ to the point where no improvement in the overall detection rate is witnessed). Table 3 presents the chosen values of all hyperparameters used throughout the experiments, in addition to their searched space.

## Deep adaptive anomaly detectors

In this section, we devise two scenarios for evaluating the ability of models to learn new anomalies. Note that in the following experiments, we use the term *known attack* for an attack class if a DL model has previously been adapted (i.e. , trained or updated) to that attack. Furthermore, the *zero-day attack* term is used for an attack class that the model has not been adapted to. In the first scenario, we use a pairwise evaluation: one known attack alongside one zero-day (i.e. , new) attack. In the second scenario, we aim to evaluate a model's ability to learn consecutive new anomalies over time, i.e. , some anomalies learned continually over time (as known attacks) and one zero-day attack.

In the first scenario's experiments, a model is initially trained with a sufficient number of flows (i.e. , about 3000–5000) from benign and one known attack (i.e. , anomaly). Afterward, for each of the remaining attacks (as the new anomalies), a set of 128 flows is used to train the expanded initial model (see Algorithm 1). Then, the expanded model is compressed back to its initial architecture (see Eq. 4). Note that to resemble a more practical circumstance, in the above, the number of unknown attack flows is selected as relatively small for evaluating the adaptive IDS.

In the evaluation phase, we report the models' detection rate for known and unknown attacks according to two separate datasets created from the original data, i.e. , known and zero-day datasets. The first one contains 500 known attack flows, and the second one includes 500 zero-day attack flows. Additionally, 500 benign flows are added to both datasets.

The results of the above-mentioned scenario's experiments are shown in Tables 4 and 5. The first column indicates the experiment's known attack, which will be used to train the initial model besides the model detection rate on the corresponding known attack. As mentioned above, the goal of this scenario is to adapt (i.e. , expand, train, and compress) the initial model to new anomalies separately and report the detection rate at different steps (called evaluation states). The second column represents the state of the reported detection rate and the rest of the columns indicate the detection rate of the model over new (i.e. , zero-day) anomalies for three evaluation states[2]: *Before Update (zero-day)*, *After Update (zero-day)*, and *After Update (initial known)*.

Prior to adapting the model to a new anomaly, the initial model detection rate is measured on the corresponding zero-day dataset and reported as *Before Update (zero-day)*. The compressed model detection rate on the same set is reported as *After Update (zero-day)* to indicate the model's improvement after continual learning. In addition, the *After Update (initial)* state represents how the updating procedure affects the model's previous knowledge (i.e. , catastrophic forgetting) by measuring the compressed model's detection rate on the anomaly on which the model was initially trained.

The results of Table 4 highlight the notable performance of CNN-based models in the adaptation process. Before being updated, most models exhibit low accuracy in detecting zero-day attacks (the "Before Update (zero-day)" accuracy for each attack is mostly below 0.45, indicating a lack of knowledge in distinguishing zero-day flows from benign ones). However, the detection rate after the update rises to an average of above 95% for all types of attacks. Additionally, CNN models demonstrate exceptional accuracy in detecting their initial attacks (i.e. , an average detection accuracy above 98%) both before and after being updated with the new zero-day attack.

Besides, based on the results in Table 5, it can be derived that LSTM-based models generally achieve acceptable detection performance during updates to new zero-day attacks. To be more precise, in most of the scenarios, the LSTM model detects its initial attack with an average rate above 90% after the updating phase. Moreover, the detection rate on the zero-day attack rises to above 80% after the update, except in cases where the model was initially trained on certain attacks like Portscan, FTP Patator, and Bruteforce Web. This implies that LSTM models rely more on their initial attack knowledge during the updating phase to extract the required features for detecting the new zero-day attacks.

In the second scenario, similar to the first one, an initial model is trained on a known anomaly. Then, considering the rest of the anomalies as zero-day attacks, the model is sequentially expanded, trained, and compressed on 128 flows of each of the remaining new attacks. The main difference between the first and second scenarios is that the

[2] Note that the goal of this evaluation is to investigate the effectiveness of the updating procedure when the model is faced with new (i.e. , zero-day) attacks (i.e. , different from its initial known attack). Hence, the experiments where the initial and zero-day attacks are the same are not reported.

**Table 4** CNN-based model

| Known attack (accuracy) | Zero-day attack | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **State** | **Botnet** | **DDOS** | **Portscan** | **DOS SlowHttpTest** | **DOS SlowLoris** | **DOS Hulk** | **DOS GoldenEye** | **FTP Patator** | **SSH Patator** | **Web BruteForce** | **Web XSS** |
| Botnet (0.96) | Before Update (zero-day) | – | 0.32 | 0.33 | 0.32 | 0.56 | 0.32 | 0.42 | 0.32 | 0.32 | 0.40 | 0.57 |
| | After Update (zero-day) | – | 0.99 | 0.99 | 0.98 | 0.98 | 0.99 | 0.94 | 1.00 | 0.97 | 0.99 | 0.98 |
| | After Update (initial) | – | 0.97 | 0.97 | 0.95 | 0.96 | 0.96 | 0.95 | 0.97 | 0.95 | 0.96 | 0.96 |
| DDoS (0.99) | Before Update (zero-day) | 0.45 | – | 0.33 | 0.34 | 0.33 | 0.45 | 0.39 | 0.33 | 0.33 | 0.42 | 0.59 |
| | After Update (zero-day) | 0.98 | – | 1.00 | 0.99 | 0.97 | 0.90 | 0.95 | 1.00 | 0.99 | 0.99 | 0.99 |
| | After Update (initial) | 0.99 | – | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.99 | 0.99 | 0.98 | 0.99 |
| Portscan (0.99) | Before Update (zero-day) | 0.44 | 0.33 | – | 0.33 | 0.58 | 0.33 | 0.33 | 0.33 | 0.33 | 0.41 | 0.58 |
| | After Update (zero-day) | 0.95 | 0.98 | – | 0.99 | 0.93 | 0.97 | 0.95 | 1.00 | 0.98 | 0.98 | 0.98 |
| | After Update (initial) | 0.98 | 0.99 | – | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.99 | 0.98 | 1.00 |
| DoS SlowHttpTest (0.98) | Before Update (zero-day) | 0.44 | 0.49 | 0.34 | – | 0.91 | 0.81 | 0.63 | 0.33 | 0.33 | 0.43 | 0.60 |
| | After Update (zero-day) | 0.97 | 0.98 | 1.00 | – | 0.99 | 0.99 | 0.97 | 1.00 | 0.99 | 0.99 | 0.98 |
| | After Update (initial) | 0.97 | 0.99 | 0.99 | – | 0.99 | 0.97 | 0.99 | 0.99 | 0.99 | 0.98 | 0.98 |
| DoS SlowLoris (0.99) | Before Update (zero-day) | 0.44 | 0.33 | 1.00 | 0.35 | – | 0.33 | 0.35 | 0.33 | 0.33 | 0.41 | 0.58 |
| | After Update (zero-day) | 0.97 | 0.98 | 1.00 | 0.96 | – | 0.99 | 0.92 | 1.00 | 0.98 | 0.99 | 0.99 |
| | After Update (initial) | 0.98 | 0.98 | 0.99 | 0.98 | – | 0.97 | 0.93 | 0.98 | 0.96 | 0.97 | 0.96 |
| DoS Hulk (0.97) | Before Update (zero-day) | 0.44 | 0.72 | 0.33 | 0.45 | 0.34 | – | 0.84 | 0.33 | 0.33 | 0.42 | 0.59 |
| | After Update (zero-day) | 0.96 | 0.99 | 1.00 | 0.98 | 0.98 | – | 0.97 | 1.00 | 0.99 | 0.99 | 0.99 |
| | After Update (initial) | 0.98 | 0.99 | 0.98 | 0.99 | 0.99 | – | 0.98 | 0.99 | 0.99 | 0.98 | 0.99 |
| DoS GoldenEye (0.98) | Before Update (zero-day) | 0.72 | 0.77 | 0.33 | 0.42 | 0.65 | 0.99 | – | 0.33 | 0.33 | 0.44 | 0.59 |
| | After Update (zero-day) | 0.97 | 0.98 | 1.00 | 0.98 | 0.97 | 0.99 | – | 1.00 | 0.99 | 1.00 | 0.99 |
| | After Update (initial) | 0.99 | 0.98 | 0.97 | 0.99 | 0.97 | 0.98 | – | 0.99 | 0.99 | 0.98 | 0.99 |

**Table 4** (continued)

| Known attack (accuracy) | Zero-day attack | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **State** | **Botnet** | **DDOS** | **Portscan** | **DOS SlowHttpTest** | **DOS SlowLoris** | **DOS Hulk** | **DOS GoldenEye** | **FTP Patator** | **SSH Patator** | **Web BruteForce** | **Web XSS** |
| FTP Patator (0.99) | Before Update (zero-day) | 0.45 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | – | 0.33 | 0.42 | 0.59 |
| | After Update (zero-day) | 0.76 | 0.82 | 1.00 | 0.97 | 0.97 | 0.91 | 0.95 | – | 0.99 | 0.98 | 0.98 |
| | After Update (initial) | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 1.00 | 0.98 | – | 1.00 | 0.99 | 0.99 |
| SSH Patator (0.99) | Before Update (zero-day) | 0.45 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | – | 0.42 | 0.59 |
| | After Update (zero-day) | 0.97 | 0.99 | 0.99 | 0.98 | 0.97 | 0.98 | 0.95 | 1.00 | – | 1.00 | 0.98 |
| | After Update (initial) | 0.99 | 0.99 | 0.99 | 1.00 | 0.99 | 0.98 | 0.99 | 1.00 | – | 0.99 | 0.99 |
| BruteForce Web (0.97) | Before Update (zero-day) | 0.44 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | 0.33 | – | 0.98 |
| | After Update (zero-day) | 0.96 | 0.98 | 1.00 | 0.97 | 0.93 | 0.97 | 0.94 | 1.00 | 0.99 | – | 0.99 |
| | After Update (initial) | 0.95 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 | 0.99 | 0.99 | - | 0.99 |
| XSS Web (0.94) | Before Update (zero-day) | 0.66 | 0.32 | 0.32 | 0.76 | 0.33 | 0.32 | 0.32 | 0.32 | 0.32 | 0.92 | – |
| | After Update (zero-day) | 0.91 | 0.98 | 0.98 | 0.99 | 0.90 | 0.98 | 0.97 | 0.99 | 0.99 | 0.98 | – |
| | After Update (initial) | 0.98 | 0.99 | 0.97 | 0.99 | 0.96 | 0.98 | 0.98 | 0.98 | 0.99 | 0.97 | – |

The detection rate of continual learning for each pair of anomalies is evaluated on the CIC-IDS2017 dataset

**Table 5** LSTM-based model

| Known attack (accuracy) | Zero-day attack | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **State** | **Botnet** | **DDoS** | **Portscan** | **DoS SlowHttpTest** | **DoS SlowLoris** | **DoS Hulk** | **DoS GoldenEye** | **FTP Patator** | **SSH Patator** | **BruteForce Web** | **XSS Web** |
| Botnet (0.93) | Before Update (zero-day) | – | 0.31 | 0.31 | 0.31 | 0.31 | 0.32 | 0.45 | 0.31 | 0.31 | 0.43 | 0.57 |
| | After Update (zero-day) | – | 0.93 | 0.95 | 0.94 | 0.92 | 0.92 | 0.93 | 0.93 | 0.95 | 0.91 | 0.87 |
| | After Update (initial) | – | 0.91 | 0.90 | 0.92 | 0.90 | 0.90 | 0.91 | 0.91 | 0.94 | 0.91 | 0.90 |
| DDoS (0.92) | Before Update (zero–day) | 0.69 | – | 0.29 | 0.44 | 0.30 | 0.77 | 0.83 | 0.29 | 0.95 | 0.40 | 0.52 |
| | After Update (zero-day) | 0.90 | – | 0.95 | 0.78 | 0.81 | 0.91 | 0.89 | 0.90 | 0.96 | 0.87 | 0.87 |
| | After Update (initial) | 0.93 | – | 0.91 | 0.92 | 0.89 | 0.93 | 0.93 | 0.88 | 0.93 | 0.90 | 0.90 |
| Portscan (0.98) | Before Update (zero-day) | 0.63 | 0.50 | – | 0.50 | 0.50 | 0.49 | 0.50 | 0.50 | 0.49 | 0.59 | 0.74 |
| | After Update (zero-day) | 0.64 | 0.50 | – | 0.62 | 0.55 | 0.50 | 0.50 | 0.50 | 0.50 | 0.90 | 0.93 |
| | After Update (initial) | 1.00 | 1.00 | – | 0.97 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.96 | 0.97 |
| DoS slowHttpTest (0.98) | Before Update (zero-day) | 0.51 | 0.57 | 0.34 | – | 0.75 | 0.43 | 0.40 | 0.78 | 0.33 | 0.79 | 0.82 |
| | After Update (zero-day) | 0.80 | 0.94 | 0.98 | – | 0.75 | 0.43 | 0.77 | 0.99 | 0.92 | 0.92 | 0.96 |
| | After Update (initial) | 0.86 | 0.97 | 0.98 | – | 0.99 | 0.99 | 0.87 | 0.99 | 0.98 | 0.99 | 0.99 |
| DoS slowLoris (0.97) | Before Update (zero-day) | 0.45 | 0.33 | 0.38 | 0.84 | – | 0.44 | 0.40 | 0.34 | 0.33 | 0.42 | 0.59 |
| | After Update (zero-day) | 0.73 | 0.66 | 0.82 | 0.93 | – | 0.94 | 0.84 | 0.77 | 0.63 | 0.77 | 0.60 |
| | After Update (initial) | 0.95 | 0.93 | 0.97 | 0.98 | – | 0.97 | 0.98 | 0.97 | 0.93 | 0.96 | 0.99 |
| DoS hulk (0.99) | Before Update (zero-day) | 0.45 | 0.34 | 0.33 | 0.36 | 0.43 | – | 0.60 | 0.36 | 0.33 | 0.43 | 0.59 |
| | After Update (zero-day) | 0.92 | 0.92 | 0.97 | 0.97 | 0.97 | – | 0.95 | 0.99 | 0.98 | 0.92 | 0.92 |
| | After Update (initial) | 0.97 | 0.96 | 0.96 | 0.97 | 0.99 | – | 0.98 | 0.99 | 0.97 | 0.96 | 0.97 |
| DoS GoldenEye (0.99) | Before Update (zero-day) | 0.48 | 0.97 | 0.33 | 0.40 | 0.67 | 0.99 | – | 0.33 | 0.33 | 0.42 | 0.60 |
| | After Update (zero-day) | 0.74 | 1.00 | 0.98 | 0.95 | 0.91 | 0.99 | – | 1.00 | 0.93 | 0.94 | 0.96 |
| | After Update (initial) | 0.99 | 0.99 | 0.98 | 0.99 | 0.98 | 0.99 | – | 0.99 | 0.93 | 0.95 | 0.98 |

**Table 5** (continued)

| Known attack (accuracy) | Zero-day attack | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **State** | **Botnet** | **DDoS** | **Portscan** | **DoS SlowHttpTest** | **DoS SlowLoris** | **DoS Hulk** | **DoS GoldenEye** | **FTP Patator** | **SSH Patator** | **BruteForce Web** | **XSS Web** |
| FTP patator (0.99) | Before Update (zero-day) | 0.45 | 0.33 | 0.45 | 0.33 | 0.34 | 0.33 | 0.33 | – | 0.33 | 0.42 | 0.60 |
| | After Update (zero-day) | 0.81 | 0.39 | 0.99 | 0.75 | 0.54 | 0.33 | 0.33 | – | 0.98 | 0.91 | 0.93 |
| | After Update (initial) | 0.96 | 0.98 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | – | 1.00 | 0.98 | 0.98 |
| SSH patator (0.92)) | Before Update (zero-day) | 0.49 | 0.80 | 0.87 | 0.55 | 0.37 | 0.29 | 0.29 | 0.92 | – | 0.81 | 0.83 |
| | After Update (zero-day) | 0.59 | 0.92 | 0.92 | 0.77 | 0.68 | 0.37 | 0.33 | 0.92 | – | 0.81 | 0.83 |
| | After Update (initial) | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 | 0.92 | – | 0.92 | 0.92 |
| BruteForce web (0.95) | Before Update (zero-day) | 0.48 | 0.32 | 0.36 | 0.41 | 0.57 | 0.31 | 0.31 | 0.33 | 0.31 | – | 0.96 |
| | After Update (zero-day) | 0.92 | 0.81 | 0.98 | 0.85 | 0.65 | 0.72 | 0.71 | 0.97 | 0.81 | – | 0.95 |
| | After Update (initial) | 0.95 | 0.89 | 0.97 | 0.93 | 0.93 | 0.90 | 0.93 | 0.96 | 0.94 | – | 0.97 |
| XSS web (0.95) | Before Update (zero-day) | 0.65 | 0.62 | 0.98 | 0.88 | 0.63 | 0.33 | 0.36 | 0.98 | 0.31 | 0.94 | – |
| | After Update (zero-day) | 0.89 | 0.92 | 0.99 | 0.91 | 0.91 | 0.89 | 0.85 | 0.99 | 0.98 | 0.95 | – |
| | After Update (initial) | 0.94 | 0.94 | 0.97 | 0.97 | 0.87 | 0.89 | 0.91 | 0.97 | 0.96 | 0.97 | – |

The detection rate of continual learning for each pair of anomalies is evaluated on the CIC-IDS2017 dataset

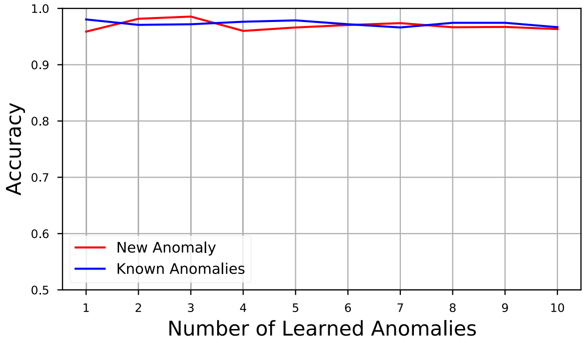


**Fig. 4** CNN-Based model detection rate after each step of learning a new anomaly on the CIC-IDS2017 dataset

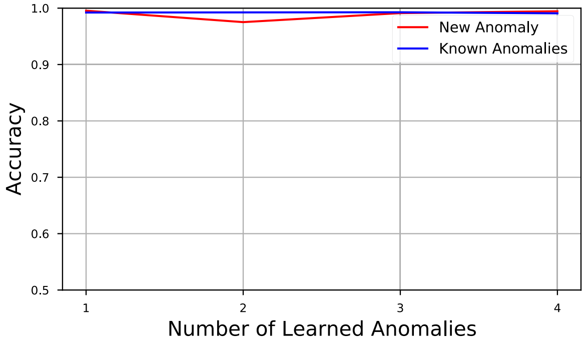


**Fig. 5** CNN-Based model detection rate after each step of learning a new anomaly on the CSE-CIC-IDS2018 dataset

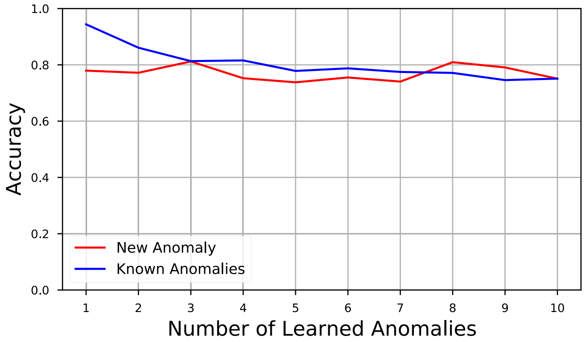


**Fig. 6** LSTM-Based model detection rate after each step of learning a new anomaly on the CIC-IDS2017 dataset

latter uses the previous step's compressed model as the initial model for the current training step. In other words, during the continual learning procedure, the model acquires knowledge about all the previous anomalies and considers them as known attacks.

Similar to the previous scenario, we perform different evaluation experiments. In each experiment, we use a different permutation for the attack sequence. Finally, the

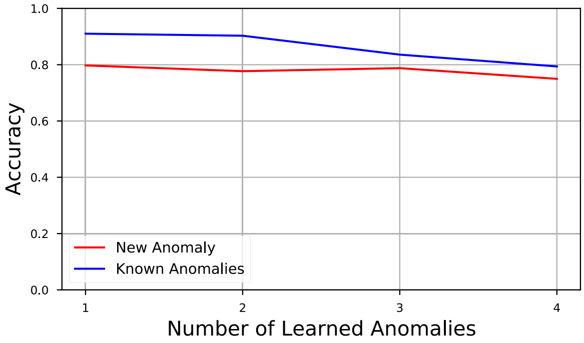


**Fig. 7** LSTM-Based model detection rate after each step of learning a new anomaly on the CSE-CIC-IDS2018 dataset

detection rate of each step is reported according to the average detection rate of all experiments' corresponding steps. As a result, this scenario does not rely on a particular attack sequence and yields more reliable results for real-world situations.

In order to evaluate the second scenario in the test phase, we prepare two datasets for each experiment's step. The first one, called *zero-day dataset*, includes 500 new attack flows and 500 benign flows. The second one, named the *known dataset*, consists of 500 attack flows for each previously known attack in addition to an equal number of benign flows for making the dataset balanced. Notice that the known dataset expands as the evaluation steps progress over the attack sequence.

Figures 4 and 5 depict the results of this experiment with CNN-based models over the CIC-IDS2017 and CSE-CIC-IDS2018 datasets, respectively. Similarly, Figs. 6 and 7 report the results on the same datasets with LSTM-based models. The results indicate that while the proposed adaptive deep IDS can continually adapt itself to the new zero-day attacks, it also preserves its ability to detect the previously observed attacks. Furthermore, the CNN-based models have a better average detection rate than LSTM-based models for detecting new anomalies (we will discuss more about the reasons for the different results produced by CNN and LSTM models in "Discussion and Future Directions" Section). To be more precise, the CNN-based models have an average detection rate above 95% both on new and previously known attacks (i.e. , after the updating procedure). On the other hand, based on Figs. 6 and 7, LSTM-based models tend to have a lower detection rate when updated on new attacks (78% at the end of the updating phase). However, it is worth mentioning that their previous knowledge is preserved during the updating procedure (i.e. , the detection rate on known anomalies does not decrease after learning a new attack).

**Table 6** CNN-based model detection rate in the federated learning approach on the CIC-IDS2017 and CSE-CIC-IDS2018 datasets

| Known attack | State | | |
|---|---|---|---|
| | Unknowns-before | Unknowns-after | Known-after |
| Botnet | 0.49 | 0.96 | 0.95 |
| DDos | 0.49 | 0.96 | 0.95 |
| Portscan | 0.51 | 0.92 | 0.92 |
| DoS SlowHttpTest | 0.65 | 0.97 | 0.95 |
| DoS SlowLoris | 0.54 | 0.96 | 0.96 |
| DoS Hulk | 0.58 | 0.95 | 0.95 |
| DoS GoldenEye | 0.63 | 0.96 | 0.95 |
| FTP Patator | 0.49 | 0.92 | 0.91 |
| SSH Patator | 0.49 | 0.95 | 0.95 |
| BruteForce Web | 0.49 | 0.94 | 0.94 |
| XSS Web | 0.61 | 0.90 | 0.91 |
| Botnet (2018) | 0.49 | 0.97 | 0.99 |
| DoS SlowLoris (2018) | 0.48 | 0.99 | 1.00 |
| DoS GoldenEye (2018) | 0.58 | 0.99 | 0.99 |
| FTP BruteForce (2018) | 0.49 | 0.98 | 1.00 |
| SSH BruteForce (2018) | 0.49 | 0.98 | 1.00 |

**Table 7** LSTM-based model detection rate in the federated learning approach on the CIC-IDS2017 and CSE-CIC-IDS2018 datasets

| Known attack | State | | |
|---|---|---|---|
| | Unknowns-before | Unknowns-after | Known-after |
| Botnet | 0.69 | 0.78 | 0.87 |
| DDos | 0.64 | 0.83 | 0.90 |
| Portscan | 0.48 | 0.50 | 0.98 |
| DoS slowHttpTest | 0.61 | 0.61 | 0.87 |
| DoS SlowLoris | 0.54 | 0.90 | 0.90 |
| DoS hulk | 0.79 | 0.89 | 0.89 |
| DoS goldenEye | 0.64 | 0.90 | 0.92 |
| FTP patator | 0.74 | 0.74 | 0.89 |
| SSH patator | 0.40 | 0.62 | 0.78 |
| BruteForce web | 0.80 | 0.81 | 0.94 |
| XSS web | 0.64 | 0.72 | 0.91 |
| Botnet (2018) | 0.51 | 0.59 | 0.98 |
| DoS slowLoris (2018) | 0.70 | 0.80 | 0.98 |
| DoS goldenEye (2018) | 0.64 | 0.80 | 0.98 |
| FTP bruteForce (2018) | 0.52 | 0.62 | 0.99 |
| SSH bruteForce (2018) | 0.48 | 0.5 | 0.91 |

### Federated learning

As discussed in "Multi-Agent IDS" Section, the federated learning technique is essential to a distributed DL-based IDS. In this section, we aim to evaluate the performance of the federated learning implementation of our proposed framework.

Although agents often may have encountered benign or known attacks in practice, we consider a more challenging case in which each agent analyzes a completely new zero-day attack for evaluating the proposed multi-agent architecture. In this scenario, the main-model is initially trained on an anomaly as the known attack. Then, a process thread is designated as an agent for each of the remaining anomalies. Each agent is responsible for learning a new anomaly and updating the main-model. When this (simultaneous) learning and (asynchronous) updating process is done, the performance of the final version of the main-model is evaluated and reported in Tables 6 and 7. Also, in our experiments, we set $\alpha$ in (6) as the ratio between the number of samples used in training the main-model and each sub-model, which was approximately 0.9.

The evaluation procedure is similar to Sect. 4.2. The main difference is that the zero-day dataset comprises a collective set consisting of 500 flows from each zero-day anomaly and a proportionate amount of benign flows. Consequently, the *Unknowns-After* state represents the model detection rate on all the unknown attacks after the federated updating phase.

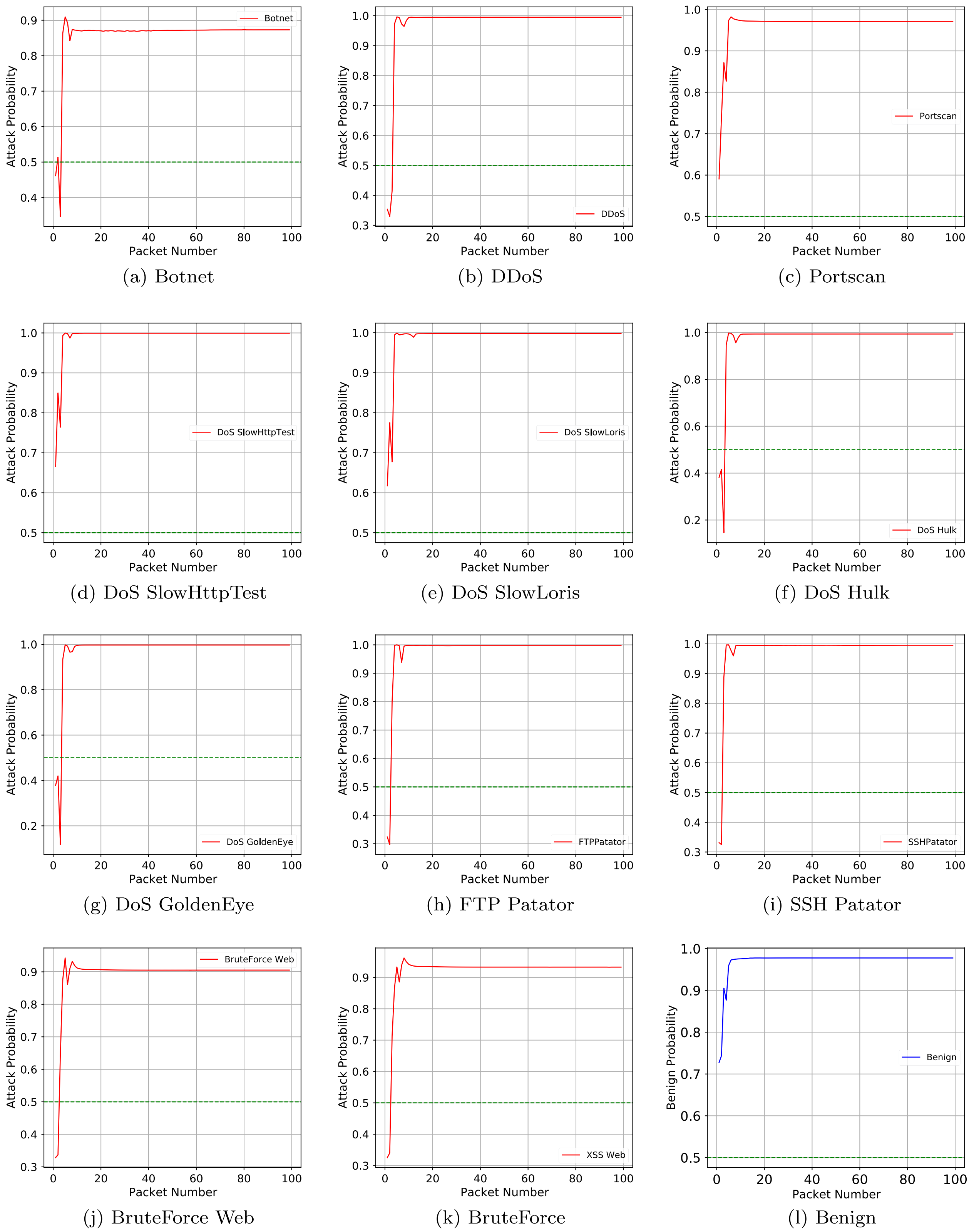
Based on Table 6, the CNN-based models prove to function well in adapting to new attack knowledge. With an average detection rate of 95% for the CIC-IDS2017 and 99% on the CSE-CIC-IDS2018 datasets, the CNN-based main-models learn to detect the new zero-day attack with the knowledge obtained through the sub-models.

Moreover, Table 7 indicate that the LSTM-based main-models tend to detect most zero-day attacks acceptably (i.e. , with a detection rate above 75%) except for attacks such as Portscan, DoS SlowHttpTest, and SSH Patator.

### Early attack detection through packet assessment

This section evaluates an LSTM model's ability to gradually assign a probability to each packet of an incoming flow. We consider a many-to-many LSTM-based model with the same architecture described in "Model Architectures" Section and train it on a collection of all the anomalies in the CIC-IDS2017 dataset. The model yields an anomaly probability per input packet. Finally, we have an output vector whose size equals the number of packets in the incoming flow.

The average probability the model assigns to a flow's true (actual) label, as a function of each incoming packet, is depicted in Fig. 8. The results demonstrate that with only 15 packets, the model can predict a flow's label with more than 80% detection rate.



(a) Botnet

(b) DDoS

(c) Portscan

(d) DoS SlowHttpTest

(e) DoS SlowLoris

(f) DoS Hulk

(g) DoS GoldenEye

(h) FTP Patator

(i) SSH Patator

(j) BruteForce Web

(k) BruteForce

(l) Benign

**Fig. 8** The average true label probability of each flow's packet sequence in the CIC-IDS2017 dataset

**Table 8** Resource and time consumption of CNN-based and LSTM-based architectures, where each number is averaged over different attacks

| | Initial model size (Memory) (MB) | Expanded model size (Memory) (MB) | Initial training (Time) (min) | Updating/ expansion (Time) (s) | Updating/ compression (Time) (min) | Validation (Time) (s) |
|---|---|---|---|---|---|---|
| CNN-based | 300 | 320 | 7 | 15 | 6 | 1.22 |
| LSTM-based | 20 | 23 | 13 | 2 | 2 | 2.97 |

## Discussion and future directions

In this section, we discuss and analyze the experiments' main results and mention possible directions for future research. The evaluations in "Deep Adaptive Anomaly Detectors" and "Federated Learning" Sections indicate that in terms of adaptability, CNN models tend to learn new traffic patterns better than LSTM models. This phenomenon could be explained by the fact that CNN layers extract features at the flow level, which capture the spatial characteristics of packets in a given flow. On the other hand, while LSTM layers are well-suited for obtaining the temporal relation between sequential packets, the feature vector extracted by them is based on the transferred history of the previous packets. Consequently, the direct data observation by CNN models can generate better features for representing the flows. While the classification patterns based on these features might change over time (according to the traffic concept drift), those features themselves embody a suitable representation of a flow. Thus, the dense layers in CNN-based models have a more straightforward task for tuning their weights when facing new traffic. The weakness of LSTM models in the case of learning new attacks (Table 5) is especially aggravated for attacks that use contents similar to benign flows (e.g. , portscan and FTP Patator[3]).

Furthermore, in our experiments, we have investigated the models' performances for adaptation to new traffic under strict constraints. To be more precise, the models are provided with a low amount of knowledge both at the initial training (i.e. , only one known anomaly is used in the initial training phase) and updating phase (i.e. , only 128 flows are used as the new traffic samples). According to our evaluations, by relaxing the above constraints, the results of LSTM-based models improve when trained with more data. On the other hand, based on the results of "Early Attack Detection Through Packet Assessment" Section, LSTM models can detect an anomaly with fewer packets, thus being more efficient and applicable to real-world scenarios. More precisely, the early detection capability of LSTM-based models can help mitigate the intrusion's impact on the target organization. Overall, the initial training of the LSTM-based models needs more effort, but they are more efficient in detecting with fewer packets and the updating process (see Table 8).

Regarding the catastrophic forgetting issue, the results in "Deep Adaptive Anomaly Detectors" Section indicate that regardless of how well the model adapts itself to new traffic, its performance on its previous knowledge will not deteriorate. Figures 4, 5, 6 and 7 indicate that after learning the new anomaly in each step, the model detection rate on previously learned anomalies is consistent with the previous step's detection rate on both new and old anomalies.

The IDS performance and its required resources are other determinative points in selecting the deep model architecture. According to Table 8, LSTM-based models are more well-suited for practical IDSes. Although they need more time for the initial training of the model, they update themselves faster in continual updating procedures and consume less memory for their models. As mentioned, the reported initial training time (in Table 8) is based on the average elapsed time for each of our different experiments with about 3000–5000 flows. The updating and validation times are reported according to processing 128 and 1000 flows, respectively.

One should also consider the efficiency of the updating procedure in an adaptive deep intrusion detection system. An IDS should update itself with the traffic concept drift as early as possible. Consequently, in this paper, we evaluate the updating procedure (Deep Adaptive Anomaly Detectors" Section) with only 128 flows of the new traffic, which is considered relatively low compared to the number of flows used to train an initial model (i.e., about 3000~5000 flows for each attack).

Considering the distributed implementation of the proposed framework (evaluated in "Federated Learning" Section), the federated distillation procedure yields acceptable results on both known and new anomalies while the agents learn novelty attacks and update the model asynchronously. As a result, the proposed multi-agent IDS framework can manage big data issues in practical situations. Furthermore, as discussed in

[3] Unlike FTP Patator, SSH Patator uses encrypted traffic. The randomness of the flow bytes makes it different from the benign traffic. Similarly, other attacks, such as web attacks and a variety of DOS attacks, use slightly different contents.

"Multi-Agent IDS" Section, the proposed framework can also improve an agent's data privacy.

Based on the obtained results, we discuss that the proposed multi-agent architecture is advantageous in several aspects:

1. In terms of privacy, since only gradients are exchanged between an agent and the main-model, the IDS can be shared between numerous organizations. Each can contribute to updating the main anomaly detection model while preserving their data privacy. Even on a geo-distributed scale, different IDSes and organizations scattered over various locations can all collaborate with the main-model (i.e. , sharing center) to securely adapt themself to new traffic patterns. An overall schematic of this scenario is depicted in Fig. 9.
2. With the emergence of Big Data, IDSes have to face colossal and highly fast generated data streaming into the network (Othman et al. 2018). A multi-agent architecture allows the dispersion of data among the sub-models in a parallel structure (i.e. , load balancing the traffic flows, as demonstrated in Fig. 10), improving the efficiency in both detecting intrusions and updating the IDS to new traffic behavior through a distributed training process.
3. Interleaving traffic packets can be tackled by assigning each flow's packets to a specific agent (note that each agent can be assigned multiple flows).

To extend this research, we mention possible directions for future studies. In the deep learning scope, it is observed that adversarial attacks are a critical challenge for DL models (Khamis et al. 2020; Madry et al. 2018; Akhtar and Mian 2018). In these types of attacks, the model is misled with deceptive data. Consequently, in future studies, one can evaluate the proposed framework against adversarial attacks and devise defense solutions for reducing this threat.

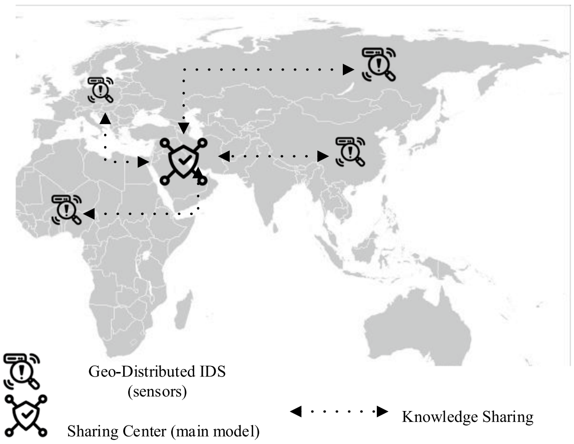


**Fig. 9** An illustration of the geo-distributed IDSes that can share their knowledge through a sharing center (i.e. , main-model)

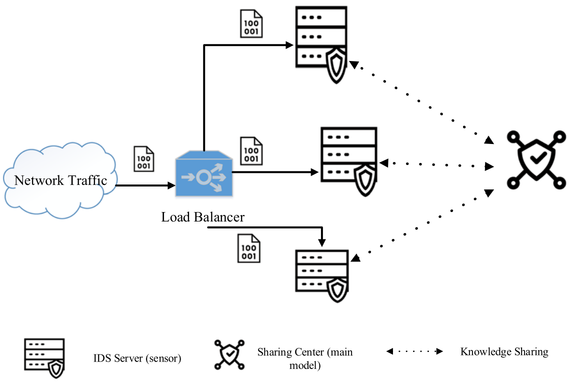


**Fig. 10** An illustration of load balancing in the proposed multi-agent framework

Moreover, the different traffic classes passed to agents are also an important issue. As discussed in Sect. 3.3, the multi-agent architecture approach proposed in this paper provides a practical resolution for IDSes from two aspects: (1) scalability in distributed IDSes for handling concurrent and high-throughput volumes of traffic and (2) knowledge sharing between differently-located agents. For the latter, one should consider the following challenge:

Generally, attack traffic can be divided into two main categories: statistical and content-based. Statistical attacks correlate highly with deployment circumstances (e.g., the topology of the target network, server capacity, geographic location, etc.). In contrast, content-based attacks are independent of the environmental characteristics of their target network. Although, according to the importance and impact of content-based attacks (Malware 2023), most traditional signature-based IDSes pivot on these attacks, the proposed framework has some challenges with statistical ones. For instance, based on the different attributes of the targeted servers, the velocity of sent request rates (e.g., for DDoS or DoS attacks) differ between different organization types. On the other hand, content-based attacks such as XSS, CSRF, and SQL injection tend to have the same signature regardless of their target domain. So, future research could analyze the aspects and challenges of sharing the attack knowledge of different attack types (statistical or content-based) between agents in inhomogeneous environments.

A straightforward solution is to disperse the agents in environments with the same characteristics. Another possible solution is to use a local, initial, threshold-based IDS for each agent to filter the statistical attacks, and only benign and content-based attacks flow to the agent model. Either way, to the best of our belief, this topic is worth further research.

In the end, to complete our analysis, we compare the proposed framework with previous related research studies from different aspects. As demonstrated in Table 9, the proposed framework simultaneously provides solutions for the three aforementioned challenges of DL-based IDSes: continuous adaption, multi-agent IDSes, and early attack detection. Furthermore, note that most proposed DL-based IDS frameworks depend on labeled datasets. However, for practical applications, future studies can develop an unsupervised version of our proposed online adaptive anomaly detection framework. We believe that, in addition to the suggestions provided in this work, accomplishing this last step will result in a DL-based IDS more suitable for real-world scenarios.

## Conclusion

This paper presented a novel framework for DL-based IDSes that mitigates three practical issues these systems are currently facing. Namely, we provided solutions for continuously adapting the IDS to network concept drift, early attack detection, and efficiently functioning in a multi-agent environment (e.g. , sharing the attack knowledge from different located IDS sensors, load-balancing the flows between different agents and managing interleaving flows).

The proposed framework exploits continual learning algorithms to update DL-based models for adapting to the concept drift in attack/benign traffic behaviors. Additionally, it uses federated learning for designing multi-agent IDSes and providing privacy and load balancing for big data traffic. Furthermore, the paper investigates the usage of Long Short-Term Memory networks (LSTMs) for packet labeling and early anomaly detection to design more practical IDSes. Finally, the framework is implemented and evaluated with two architectures: convolutional neural networks (CNNs) and LSTM-based models. The results indicate that while both architectures perform well, CNN models prevail in terms of detection rate, and LSTM models are more suitable for early anomaly detection with just a few packets.

**Table 9** Comparison between the previous related studies and the proposed framework

| | DL-based | Unsupervised | Continuous Adaptation | Multi-Agent | Early Attack Detection | Dataset |
|---|---|---|---|---|---|---|
| Yin et al. (2017) | ✓ | | | | | NSL-KDD |
| Vinayakumar et al. (2017) | ✓ | | | | | KDDCup 99 |
| Thakur et al. (2021) | ✓ | | | | | CIC-IDS2017 |
| Riyad et al. (2019) | | | ✓ | ✓ | | KDD99 |
| Kim and Park (2019) | ✓ | | ✓ | | | KDD99, NSL-KDD |
| Papamartzivanos et al. (2019) | ✓ | ✓ | ✓ | | | KDD99, NSL-KDD |
| Gupta et al. (2022) | ✓ | | | | | NSL-KDD, CIDDS-001, CIC-IDS2017 |
| Wang et al. (2021) | ✓ | | | | | KDD99, NSL-KDD, UNSW-NB15, CIDDS-001, ADFA-LD |
| Wang et al. (2022) | ✓ | | | | | UNSW NS2019, ISCX IDS 2012, CIC-IDS2017, CIC-ANDMAL2017 |
| Cretu-Ciocarlie et al. (2009) | | ✓ | ✓ | | | Network Traffic of Columbia University's Computer Science Department |
| Folino et al. (2021) | ✓ | Semi-supervised | ✓ | | | CIC-IDS2017, ISCXIDS2012 |
| Soltani et al. (2023) | ✓ | | | | | CIC-IDS2017, CSE-CIC-IDS2018 |
| Mirza and Cosan (2018) | ✓ | | | | ✓ | ISCXIDS2012 |
| Gao et al. (2019) | ✓ | | | | ✓ | SCADA simulated testbed |
| Proposed Framework | ✓ | | ✓ | ✓ | ✓ | CIC-IDS2017, CSE-CIC-IDS2018 |

### Author contributions

MS: Conceptualization, methodology, software, implementation, validation. KK: Conceptualization, methodology, software, implementation, validation. MJS: Conceptualization, methodology, supervision. AHJ: Conceptualization, supervision.

### Funding

No funding was received to assist with the preparation of this manuscript.

### Availability of data and materials

The datasets that have been used in this research are all publicly available. The two datasets CIC-IDS2017 and CSE-CIC-IDS2018 have been used for all the experiments in this research; they are publicly available.

### Code availability

The implementations of all evaluations and experiments are available here at Github repository.

## Declarations

### Ethical approval and consent to participate

This article does not contain any studies with human participants or animals performed by any of the authors.

### Competing interests

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Received: 5 April 2023 Accepted: 10 December 2023
Published: 1 May 2024

### References

Abadi M et al (2015) TensorFlow: large-scale machine learning on heterogeneous systems. http://tensorflow.org/. Software available from tensorflow.org

Abou El Houda Z, Brik B, Khoukhi L (2022) "why should i trust your ids?": an explainable deep learning framework for intrusion detection systems in internet of things networks. IEEE Open J Commun Soc 3:1164–1176

Adawadkar AMK, Kulkarni N (2022) Cyber-security and reinforcement learning-a brief survey. Eng Appl Artif Intell 114(105):116

Akhtar N, Mian A (2018) Threat of adversarial attacks on deep learning in computer vision: a survey. IEEE Access 6:14410–14430

Alghamdi R, Bellaiche M (2023) An ensemble deep learning based ids for IoT using lambda architecture. Cybersecurity 6(1):5

Andresini G, Appice A, De Rose L, Malerba D (2021) Gan augmentation to deal with imbalance in imaging-based intrusion detection. Fut Gener Comput Syst 123:108–127

Ansari MS, Bartoš V, Lee B (2022) Gru-based deep learning approach for network intrusion alert prediction. Fut Gener Comput Syst 128:235–247

Bhargavi R, Vaidehi V (2013) Semantic intrusion detection with multisensor data fusion using complex event processing. Sadhana 38(2):169–185

CSE-CIC-IDS2018 (2021) https://www.unb.ca/cic/datasets/ids-2018.html

Chai Z, Chen Y, Anwar A, Zhao L, Cheng Y, Rangwala H (2021) Fedat: a high-performance and communication-efficient federated learning system with asynchronous tiers. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, pp 1–16

Choi YH, Liu P, Shang Z, Wang H, Wang Z, Zhang L, Zhou J, Zou Q (2020) Using deep learning to solve computer security challenges: a survey. Cybersecurity 3(1):1–32

Chollet F (2017) keras. https://github.com/fchollet/keras

Cretu-Ciocarlie GF, Stavrou A, Locasto ME, Stolfo SJ (2009) Adaptive anomaly detection via self-calibration and dynamic updating. In: International workshop on recent advances in intrusion detection, pp 41–60

Diro AA, Chilamkurti N (2018) Distributed attack detection scheme using deep learning approach for internet of things. Fut Gener Comput Syst 82:761–768

Folino F, Folino G, Guarascio M, Pisani F, Pontieri L (2021) On learning effective ensembles of deep neural networks for intrusion detection. Inf Fus 72:48–69

Gao J, Gan L, Buschendorf F, Zhang L, Liu H, Li P, Dong X, Lu T (2019) Lstm for SCADA intrusion detection. In: 2019 IEEE pacific rim conference on communications, computers and signal processing (PACRIM), IEEE, pp 1–5

Gimpel K, Das D, Smith NA (2010) Distributed asynchronous online learning for natural language processing. In: Proceedings of the fourteenth conference on computational natural language learning, pp 213–222

Gong P, Ye J, Cs Zhang (2012) Multi-stage multi-task feature learning. Adv Neural Inf Process Syst 25:1997–2005

Gupta N, Jindal V, Bedi P (2022) CSE-IDS: using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems. Comput Secur 112(102):499

Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

Huang Z, Xu W, Yu K (2015) Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint arXiv:1508.01991

Hwang RH, Peng MC, Nguyen VL, Chang YL (2019) An LSTM-based deep learning approach for classifying malicious traffic at the packet level. Appl Sci 9(16):3414

Iyengar N (2020) Evaluation of network based IDS and deployment of multisensor IDS. arXiv preprint arXiv:2007.11654

Jain S, Kasaei H (2021) 3D_DEN: open-ended 3D object recognition using dynamically expandable networks. IEEE Trans Cognit Dev Sys. https://doi.org/10.1109/TCDS.2021.3075143

KDD Cup 1999 (2021) http://kdd.ics.uci.edu/databases/kddcup 99/kddcup99.html

Khamis RA, Shafiq MO, Matrawy A (2020) Investigating resistance of deep learning-based ids against adversaries using min-max optimization. In: ICC 2020—2020 IEEE international conference on communications (ICC), pp 1–7. https://doi.org/10.1109/ICC40277.2020.9149117

Kim C, Park J (2019) Designing online network intrusion detection using deep auto-encoder q-learning. Comput. Electr. Eng. 79:106460

Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu AA, Milan K, Quan J, Ramalho T, Grabska-Barwinska A et al (2017) Overcoming catastrophic forgetting in neural networks. Proc Natl Acad Sci 114(13):3521–3526

Labonne M (2020) Anomaly-based network intrusion detection using machine learning. Ph.D. thesis, Institut Polytechnique de Paris

Lee SW, Mohammadi M, Rashidi S, Rahmani AM, Masdari M, Hosseinzadeh M et al (2021) Towards secure intrusion detection systems using deep learning techniques: comprehensive analysis and review. J Netw Comput Appl 187(103):111

Liang KJ, Li C, Wang G, Carin L (2018) Generative adversarial network training is a continual learning problem. arXiv preprint arXiv:1811.11083

Lippmann R, Haines JW, Fried DJ, Korba J, Das K (2000) The 1999 DARPA off-line intrusion detection evaluation. Comput Netw 34(4):579–595. https://doi.org/10.1016/S1389-1286(00)00139-0

Lopez-Martin M, Carro B, Sanchez-Esguevillas A, Lloret J (2017) Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access 5:18042–18050

Malware Statistics in (2023) Frequency, impact, cost & more: comparitech.com. https://www.comparitech.com/antivirus/malware-statistics-facts/. Accessed 12 Sept 2023

Ma X, Hovy E (2016) End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. arXiv preprint arXiv:1603.01354

Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A (2018) Towards deep learning models resistant to adversarial attacks. In: 6th international conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30 –May 3, 2018, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=rJzIBfZAb

Martens J (2020) New insights and perspectives on the natural gradient method. J Mach Learn Res 21(1):5776–5851

Mirza AH, Cosan S (2018) Computer network intrusion detection using sequential lstm neural networks autoencoders. In: 2018 26th signal processing and communications applications conference (SIU), IEEE, pp 1–4

NIST security vulnerability trends in 2020 (2021) an analysis. https://www.redscan.com/media/Redscan_NIST-Vulnerability-Analysis-2020_v1.0.pdf

Othman SM, Ba-Alwi FM, Alsohybe NT, Al-Hashida AY (2018) Intrusion detection model using machine learning algorithm on Big Data environment. J Big Data 5(1):1–12

Papamartzivanos D, Mármol FG, Kambourakis G (2019) Introducing deep learning self-adaptive misuse network intrusion detection systems. IEEE Access 7:13546–13560

Riyad A, Ahmed MI, Khan RR (2019) An adaptive distributed intrusion detection system architecture using multi agents. Int J Electr Comput Eng 9(6):4951

Rusu AA, Rabinowitz NC, Desjardins G, Soyer H, Kirkpatrick J, Kavukcuoglu K, Pascanu R, Hadsell R (2016) Progressive neural networks. arXiv preprint arXiv:1606.04671

Saba T, Rehman A, Sadad T, Kolivand H, Bahaj SA (2022) Anomaly-based intrusion detection system for IoT networks through deep learning model. Comput Electr Eng 99(107):810

Scardapane S, Comminiello D, Hussain A, Uncini A (2017) Group sparse regularization for deep neural networks. Neurocomputing 241:81–89

Schwarz J, Czarnecki W, Luketina J, Grabska-Barwinska A, Teh YW, Pascanu R, Hadsell R (2018) Progress & compress: A scalable framework for continual learning. In: International conference on machine learning, PMLR, pp 4528–4537

Seff A, Beatson A, Suo D, Liu H (2017) Continual learning in generative adversarial nets. arXiv preprint arXiv:1705.08395

Seresht NA, Azmi R (2014) Mais-ids: a distributed intrusion detection system using multi-agent ais approach. Eng Appl Artif Intell 35:286–298

Sharafaldin I, Lashkari AH, Ghorbani AA (2018) Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Mori P, Furnell S, Camp O (eds) Proceedings of the 4th international conference on information systems security and privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22–24, 2018, pp 108–116. SciTePress. https://doi.org/10.5220/0006639801080116

Soltani M, Ousat B, Siavoshani MJ, Jahangir AH (2023) An adaptable deep learning-based intrusion detection system to zero-day attacks. J Inf Secur Appl 76(103):516

Soltani M, Siavoshani MJ, Jahangir AH (2022) A content-based deep intrusion detection system. Int J Inf Secur. https://doi.org/10.1007/s10207-021-00567-2

Sutton RS, McAllester DA, Singh SP, Mansour Y (2000) Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems, pp 1057–1063

Tavallaee M, Bagheri E, Lu W, Ghorbani AA (2009) A detailed analysis of the KDD CUP 99 data set. In: 2009 ieee symposium on computational intelligence for security and defense applications, CISDA 2009, Ottawa, July 8–10, 2009, IEEE, pp 1–6. https://doi.org/10.1109/CISDA.2009.5356528

Thakkar A, Lohiya R (2021) A review on machine learning and deep learning perspectives of ids for IoT: recent updates, security issues, and challenges. Arch Comput Methods Eng 28(4):3211–3243

Thakur S, Chakraborty A, De R, Kumar N, Sarkar R (2021) Intrusion detection in cyber-physical systems using a generic and domain specific deep autoencoder model. Comput Electr Eng 91(107):044

Varshney S, Verma VK, Srijith P, Carin L, Rai P (2021) Cam-gan: continual adaptation modules for generative adversarial networks. Adv Neural Inf Process Syst 34:15175–15187

Van de Ven GM, Tolias AS (2019) Three scenarios for continual learning. arXiv preprint arXiv:1904.07734

Vinayakumar R, Soman K, Poornachandran P (2017) Applying convolutional neural network for network intrusion detection. In: 2017 International conference on advances in computing, communications and informatics (ICACCI), IEEE, pp 1222–1228

Wang Z, Fok KW, Thing VL (2022) Machine learning for encrypted malicious traffic detection: approaches, datasets and comparative study. Comput Secur 113(102):542

Wang Z, Liu Y, He D, Chan S (2021) Intrusion detection methods based on integrated deep learning model. Comput Secur 103:102177

Xie C, Koyejo S, Gupta I (2019) Asynchronous federated optimization. arXiv preprint arXiv:1903.03934

Xu J, Zhu Z (2018) Reinforced continual learning. arXiv preprint arXiv:1805.12369

Yi S, Li C, Li Q (2015) A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 workshop on mobile big data, pp 37–42

Yin C, Zhu Y, Fei J, He X (2017) A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access 5:21954–21961

Yoon J, Yang E, Lee J, Hwang SJ (2017) Lifelong learning with dynamically expandable networks. arXiv preprint arXiv:1708.01547

Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? arXiv preprint arXiv:1411.1792

Zenke F, Poole B, Ganguli S (2017) Continual learning through synaptic intelligence. In: International conference on machine learning, PMLR, pp 3987–3995

Zhang J, Zhang J, Ghosh S, Li D, Zhu J, Zhang H, Wang Y (2020) Regularize, expand and compress: Nonexpansive continual learning. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision, pp 854–862

## Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.