Adaptive Clustering-based Malicious Traffic Classification at the Network Edge

Alec F. Diallo and Paul Patras
School of Informatics, The University of Edinburgh
Scotland, United Kingdom

***Abstract*—The rapid uptake of digital services and Internet of Things (IoT) technology gives rise to unprecedented numbers and diversification of cyber attacks, with which commonly-used rule-based Network Intrusion Detection Systems (NIDSs) are struggling to cope. Therefore, Artificial Intelligence (AI) is being exploited as second line of defense, since this methodology helps in extracting non-obvious patterns from network traffic and subsequently in detecting more confidently new types of threats. Cybersecurity is however an arms race and intelligent solutions face renewed challenges as attacks evolve while network traffic volumes surge. In this paper, we propose Adaptive Clustering-based Intrusion Detection (ACID), a novel approach to malicious traffic classification and a valid candidate for deployment at the network edge. ACID addresses the critical challenge of sensitivity to subtle changes in traffic features, which routinely leads to misclassification. We circumvent this problem by relying on low-dimensional embeddings learned with a lightweight neural model comprising multiple kernel networks that we introduce, which optimally separates samples of different classes. We empirically evaluate our approach with both synthetic and three intrusion detection datasets spanning 20 years, and demonstrate ACID consistently attains 100% accuracy and F1-score, and 0% false alarm rate, thereby significantly outperforming state-of-the-art clustering methods and NIDSs.**

***Index Terms*—network intrusion detection; kernel-based clustering; deep learning**

I. INTRODUCTION

The adoption of Internet of Things (IoT) devices and cloud-based services continues to grow sharply [1], leading to a pressing need for robust and efficient defense mechanisms to safeguard the networking infrastructure and users' private data. This is particularly critical as attackers continue to discover new system/software vulnerabilities on a daily basis [2]. As a result, cybercrime costed businesses and individuals in the United States alone $3.5 billion in 2019 [3]. Meanwhile, traditional security measures such as firewalls, anti-viruses, and rule-based Network Intrusion Detection Systems (NIDSs) are unable to keep up with the most recent and sophisticated attacks that exploit loopholes to bypass the perimeter defenses set by these measures [4]. In particular, widely-deployed NIDSs, including Snort [5], Zeek [6], or Suricata [7] present a number of disadvantages. Namely, they *(i)* require frequent updates of signature databases; *(ii)* exhibit high false alarm rates when classifying traffic with evolving behavior; and *(iii)* depend on considerable levels of human expert intervention for system tuning and manual decision making.

This research was supported by Arm Ltd.

In this context, Artificial Intelligence (AI)- and Machine Learning (ML)-based techniques such as Artificial Neural Networks, Clustering, and Ensemble Learning are increasingly appealing for building automatic network threat or anomaly detection systems [8], [9]. This is largely due to the unique ability of neural models to discover hidden patterns in vast amounts of data, which helps boosting classification accuracy, as already demonstrated in several research areas including speech recognition [10], computer vision [11], and wireless and mobile networking [12].

However, despite the rapid progress of AI-based approaches to Network Intrusion Detection (NID), existing solutions (e.g., [13]–[15]) remain extremely sensitive to small changes in individual features of network traffic flows, which dilutes their effectiveness in the face of continuous software updates and evolving traffic landscapes, as we reveal. Specifically, since these techniques learn from features of individual samples, training them on small subsets of carefully crafted features, unwittingly mislabelled samples, or unbalanced datasets negatively impacts on their generalization abilities, thereby rendering the detection of new malicious network activity very difficult. Additionally, current NIDSs introduce application latency due to their complexity, while their architectures are usually fixed, thus requiring retraining for every new task.

To tackle these problem, in this paper, we propose ACID, a classifier-agnostic and highly-effective Adaptive Clustering-based Intrusion Detection system. Our design incorporates an original multi-kernel based neural network that enables our NIDS to generalize well, regardless of any small changes incurred by small groups of packets or the unbalanced nature of the training dataset. We achieve this by means of a clustering algorithm that learns low-dimensional embeddings from linear and non-linear combinations of network flow features, which makes it possible to unambiguously separate these flows. By combining the cluster centers learned through our clustering network with statistical and semantic features extracted from packet sequences, and feeding the resulting feature vectors to a classifier, we effectively improve the detection performance of the NIDS while minimizing the false alarm rate. In a nutshell, our **key contributions** can be summarized as follows:

- **[C1]** We introduce a novel supervised Adaptive Clustering (AC) technique to learn cluster centers that can be used to expand the features of a given dataset. This approach improves the robustness against outliers and the generalization abilities of any classification model.

The solution is generally applicable to classification tasks in any domain, but is particularly useful for NID.

[C2] We study the ability of our approach to discover low-dimensional representations that can simplify separating a dataset into distinct classes, and show that our algorithm quickly self-adapts to complex, intertwined, and evolving structures, such as non-linearly separable data in multi-dimensional spaces, e.g., network traffic flows.

[C3] Building on our clustering approach, we design a NIDS that extracts a range of features from raw packets, extends these with the learned cluster centers, and feeds them to a classifier of choice to achieve binary and multi-label classification with high reliability.

[C4] To demonstrate the effectiveness of our solution, we evaluate ACID on three different network traffic datasets containing illicit flows that encompass 40 attack types spanning 20 years (i.e., KDD Cup'99 [16], ISCX-IDS 2012 [17], and CSE-CIC-IDS 2018 [18]) and reveal it consistently achieves F_1-scores of 100.0%, thereby outperforming existing NID approaches by up to 47%.

[C5] We assess the complexity of our approach in terms of computational requirements and runtime, and show that our model only requires 80ms per inference instance, making the case for ACID as a fast and lightweight candidate for deployment on constrained edge devices.

To the best of our knowledge, ACID is the first Deep Learning (DL)-based NIDS that exploits adaptive clustering to minimize false alarm rates, it is robust to a range of malicious traffic types, and it is amenable to prototyping on commodity gateways for real-time threat detection at the network edge.

II. Related Work

We briefly overview DL-based NID approaches directly related to our ACID system, and highlight their shortcomings.

A. Deep Learning-based Intrusion Detection

Most DL-based NIDSs attempt to match observed network flows against previously learned patterns. Despite increasing adoption, they produce unacceptably high false alarm rates for relatively small gains in detection performance. This significantly limits their applicability to real-life scenarios. Auto-Encoders (AEs) can learn latent representations of features and reduce their dimensionality in order to minimize memory consumption, which motivates their use for anomalous traffic detection [19]–[21]. Tan et al. apply Convolutional Neural Networks (CNNs) to learn spatial representations of packets, followed by image classification methods to identify malware traffic [22]. Wang et al. combine CNNs and Long Short-Term Memory (LSTM) structures to learn both spatial and temporal correlations between features [23]. Despite the effectiveness of these techniques, they completely ignore time-based statistical features that can be inferred from packets and the semantic relationships within packet payloads. Min et al. use these ignored attributes and apply Natural Language Processing techniques to process packet payloads [24]. This boosts detection performances, yet still presents several important weaknesses, including ignoring dataset imbalance and exhibiting very high processing times when dealing with large datasets. Under- and oversampling methods [25], [26] can mitigate this class imbalance problem, but these techniques either reduce the number of training data samples or use additional artificially generated data, both of which negatively impact on the classification performance, as they restrain the ability of ML models to learn accurate representations.

B. Clustering based Intrusion Detection

Ideally, any Intrusion Detection System (IDS) should *(i)* have learning and hierarchical feature representation abilities; *(ii)* handle high-dimensional data and extract valuable patterns. Since clustering methods group data into meaningful sub-classes, seeking to separate members of different clusters, several IDSs build on this approach. Jianliang et al. use k-means clustering to detect unknown attacks and separate large data spaces effectively [27]. However, their approach suffers from degeneracy and cluster dependence, which could be overcome with the Y-means clustering algorithm proposed in [28]. Mingqiang et al. introduce the concept of graph-based clustering for anomaly detection, whereby a Local Deviation Coefficient Graph Based (LDCGB) approach identifies outliers [29]. Li et al. use a Particle Swarm Optimization (PSO) algorithm based on swarm intelligence [30]. This solution avoids falling into local minima, while providing good overall convergence. Multi-stage techniques improve NID by *(i)* generating meta-alerts through clustering and *(ii)* reducing false alarm rates via classification of these meta-alerts [31].

However, these approaches are frequently unable to discriminate superficially similar but in essence different attacks (e.g., U2R and R2L vs. benign), present high misclassification rates due to their unsupervised nature, and/or are computationally expensive, making them unfit for deployment on constrained devices. We overcome these issues through a simple and effective adaptive clustering approach, while offering significant improvements in detection rate and minimizing false alarms.

III. Threat Model

We consider both home networks and enterprise environments subject to two offensive scenarios. First, we envisage attackers located outside the target network, to which they attempt to gain access, compromise victim devices, or retrieve sensitive data. In this scenario, the attacker would scan all the devices connected to the network to find weaknesses that would allow access. In the second scenario we consider, the attacker is either located outside the target network, which was already compromised (giving them the ability to control the hijacked hosts), or is internally connected to that network. In both cases, we assume an NIDS is deployed on an edge device, where all incoming and outgoing network traffic packets can be captured. Furthermore, we consider the possibility where an attacker has acquired sufficient knowledge of the network infrastructure, including target IP addresses, open port numbers, etc. However, we assume the deployed NIDS is hardened and attackers do not have the ability to access or alter its behaviour.

Our proposed system is best suited to networks where most user traffic belongs to a finite set of known applications. Therefore, as we monitor all incoming and outgoing communications, we are able to train our neural model in a supervised manner on a labelled dataset.

IV. System Architecture

We propose a novel Deep Learning (DL)-based NIDS that maximizes the probability of detecting malicious network flows and minimizes the false alarm rate. Our approach aims to *(i)* quickly adapt to complex data structures and patterns, by discovering low-dimensional embeddings of networks flows, which optimally separate samples of different types; and *(ii)* be deployable on devices with limited computational capabilities. These are essential features any NIDS must meet to be suitable for practical real-life environments. We fulfill these goals by combining three key components shown in the high-level overview of our ACID system in Fig. 1, namely:

- **Feature Extractor module**: transforms raw network packets into vectors of header and statistical features, and (optionally) semantic representations of payloads (i.e., additional feature vectors);
- **Adaptive Clustering module**: builds low-dimensional embeddings of network flow features and computes a set of abstract attributes that are common to samples belonging to the same traffic type;
- **Classification module**: uses features extracted from both network flows and by the clustering module to improve detection rate and minimize the impact of outliers. This module corrects any misclassifications made via clustering and can exploit further correlations in the inputs.

In what follows we detail the operation of each of these modules, then demonstrate how the synergies among them lead to remarkable malicious traffic detection performance.

A. Feature Extractor

ACID handles very large amounts of traffic by processing streams of raw network packets into feature-based representations of bidirectional flows corresponding to communications between (source, destination) pairs, over specific applications or protocols. These are subsequently used for clustering and classification. Our feature extractor comprises two parallel processing pipelines: (1) a *header analyzer* logic that builds a set of header and statistical features (including source/destination port numbers, packet inter-arrival time, total number of packets in a flow, etc.), which provide a compact representation of traffic behaviors; and (2) an optional *word embedding* logic that builds vectors of semantic representation of the payloads, though word2vec [32] and Text-CNN [33] techniques, similarly to the methodology previously used in [24]. While this additional payload features extractor unit significantly increases the computation costs, it can improve the performance of the classifier. This is because the semantic representations of payload features encode important information about contents of payload-based attacks, such as in the case of SQL injection.

Depending on the computational power of the device where the NIDS is deployed, the feature extraction module may introduce some latency. Regardless, online operation is easily achievable if running the NIDS separately from the packet forwarding unit. It is also worth noting that by aggregating packets into bidirectional flows, we drastically reduce the size of the training and evaluation sets, which in turn reduces the inference time of ACID.

B. Adaptive Clustering Module

One of our key contributions is a novel technique that improves the generalization abilities and robustness of any DL-based classification engine. In essence, we propose a clustering algorithm that produces cluster centers to be used as extensions of the input features being clustered. As our Adaptive Clustering (AC) approach is designed to be end-to-end differentiable, the training is performed on mini-batches, whereby the network learns low-dimensional representations of the inputs and computes the corresponding kernel centers. This operation is performed online (in an iterative manner) and the final layer of the kernel networks yields the probabilities of each sample in the inputs belonging to all possible classes. For our NIDS, we retrieve the computed cluster centers of each class and use them to expand their samples' features, thereby obtaining more features for each data point. These combined features are then given to the classification module for final decisions about the nature of the traffic observed.

Since we aim to deploy our NIDS in a live environment, we aim at a clustering algorithm which:

- *Handles very large amounts of high-dimensional data points* – This requirement prevents us from using clustering approaches that need all training data at once.
- *Handles incoming streams of data at random time intervals* – New incoming data should not require retraining the entire clustering algorithm; instead, data distributions should be learned on the fly.
- *Quickly and effectively clusters even the most complex data points in multi-dimensional spaces* – Having semantically good clusters would boost the performance of the classifier, as proven by our experiments.
- *Produces cluster centers* – This is an essential requirement for improving the robustness and generalization abilities of the classifier.

These set of desired properties lead to designing a *plug & play* type DL-based clustering algorithm that can be trained with batches of data at a time, which we detail in Section V.

C. Classification Module

Finally, ACID employs a classification module that processes the combined extracted features and cluster centers for each sample, and outputs the inferred traffic class to which that flow belongs. While the classifier's architecture can be designed to obtain any desired property (e.g., exploit spatio-temporal correlations or perform binary/multi-label classification), the additional features provided by our AC algorithm

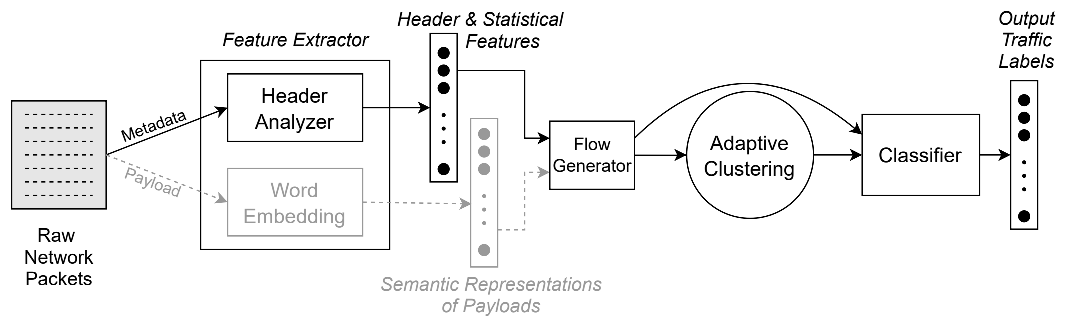

Fig. 1: Overview of the proposed ACID system. Header and statistical features, and (optionally) payload features are extracted from raw network packets, then grouped into bidirectional flows. Our AC module takes these flows, computes cluster centers, and appends them to the extracted features, before feeding the resulting vectors to a classifier for final decisions.

improve classification performance regardless of the specifics of the classifier, as they reduce the divergence between samples of the same class. To further reduce this divergence, we use each sample's corresponding cluster center instead of its low-dimensional representation, when extending its features vector. This not only improves the accuracy of the NIDS, but also minimizes the impact of outliers. To support this claim, we provide a theoretical proof of the gain introduced by using cluster centers as additional features.

Theorem. — *Clustering-based Divergence Reduction:* *Let K_c be the set of features obtained from the cluster center of a given class C. For any two samples $\chi_i, \chi_j \in C$,*

$$distance(\chi_i \cup K_c, \chi_j \cup K_c) \leq distance(\chi_i, \chi_j),$$

Proof. Let χ_i and χ_j be n-dimensional real-valued vectors:

$$\begin{aligned} \chi_i &= \{x_{i1}, x_{i2}, ..., x_{in}\} \in \mathbb{R}^n, \\ \chi_j &= \{x_{j1}, x_{j2}, ..., x_{jn}\} \in \mathbb{R}^n, \end{aligned}$$

where x_{it} and x_{jt} correspond to individual features of the sample data, with $t \in \{0, 1, ..., n\}$. Assume that χ_i and χ_j both belong to the C-th cluster according to our clustering module, and the C-th cluster center is defined as

$$K_c = \{k_{c1}, k_{c2}, ..., k_{cm}\} \in \mathbb{R}^m.$$

The aggregated features obtained after clustering are then

$$\begin{aligned} \chi'_i &= \{x_{i1}, x_{i2}, ..., x_{in}, k_{c1}, k_{c2}, ..., k_{cm}\} \in \mathbb{R}^{n+m}, \\ \chi'_j &= \{x_{j1}, x_{j2}, ..., x_{jn}, k_{c1}, k_{c2}, ..., k_{cm}\} \in \mathbb{R}^{n+m}. \end{aligned}$$

An intuitive way to calculate the impact of the clustering on robustness is to compare the similarities between the original and aggregated feature vectors, i.e., $Q_1 = distance(\chi_i, \chi_j)$ vs $Q_2 = distance(\chi'_i, \chi'_j)$. Expanding each yields

$$\begin{aligned} Q_1 &= \frac{1}{n} \sum_{\alpha=1}^{n} (x_{i\alpha} - x_{j\alpha})^2, \\ Q_2 &= \frac{1}{n+m} \left(\sum_{\alpha=1}^{n} (x_{i\alpha} - x_{j\alpha})^2 + \sum_{\alpha=1}^{m} (k_{c\alpha} - k_{c\alpha})^2 \right) \\ &= \frac{1}{n+m} \sum_{\alpha=1}^{n} (x_{i\alpha} - x_{j\alpha})^2. \end{aligned}$$

$$Q_1 - Q_2 = \frac{m}{n+m} \cdot Q_1 \implies Q_2 = \beta \cdot Q_1,$$

where $\beta = n/(n+m) \leq 1, \forall n > 0, \forall m \geq 0$, is the impact measure obtained by using the cluster centers as additional features. This implies that $Q_2 \leq Q1, \forall m \geq 0$. □

Hence, using cluster centers as additional features can only improve the decision capability of the classifiers by reducing the differences between items of the same group. The experimental results we present in Section VII fully support this finding. In our experiments, we will use a Random Forest structure [34] to perform the final classification of traffic flows.

It is worth noting that our clustering approach is not limited to NID, but can be applied to any other classification task.

V. Adaptive Clustering

Clustering algorithms learn some notion of similarity within a dataset, to group similar samples together. They usually heavily depend on individual data points and *(i)* bear significant time complexity when handling large amounts of data or large sample dimensionality; *(ii)* require an explicit measure of "distance" (even in multi-dimensional spaces); and *(iii)* return multiple possible interpretations of clustering results. Neural Networks (NNs) tackle these issues, yet often at the cost of reduced clustering performance. This is particularly problematic when dealing with complex datasets, where the NN, in seeking to generalize to all possible clusters (or classes), ends up only able to correctly classify a small subset of unambiguous samples. Unfortunately, most realistic datasets are multi-dimensional, wherein there are no obvious distinct patterns between members of different classes.

To tackle this problem, the Adaptive Clustering (AC) method we propose builds on multiple kernel networks, each learning to tailor itself to one of the possible clusters associated with a particular type of traffic which we want to classify, as illustrated in Fig. 2. For any observed sample, different encoders discover its optimal low-dimensional embedding and kernel networks learn to extrapolate a general representation (i.e., cluster center) of its group's members. By using encoders, our model reduces the dimensionality of the input features to any desired dimension. While the encoder architecture can be selected according to the task at hand, using a simple multi-layer, feed-forward NN allows the model to achieve

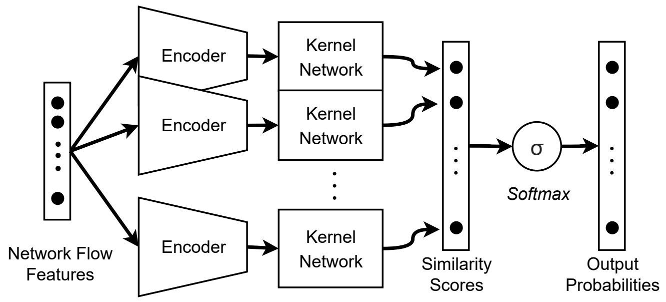

Fig. 2: Architecture of our Adaptive Clustering network.

optimal clustering while minimizing the overall computational overhead, as our results in Section VII demonstrate.

Formally, let us consider the set of N samples of n features $\boldsymbol{D} \subset \mathbb{R}^{N\times n}$. Let $\psi_{\theta_e}(x) : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be an embedding of a sample x mapped by a fully-connected feed-forward NN parametrized by $\theta_e \in \mathbb{R}^e$, onto $\mathbb{R}^m$, where m is the target dimension. The embedding of our entire dataset via ψ_{θ_e} can be expressed by $\Psi_{\theta_e} : \mathbb{R}^{N\times n} \longrightarrow \mathbb{R}^{N\times m}$. In this setting, we also use the same dimension m for the kernels to be learned by the kernel networks. We now define $\Psi_{\theta_k}(x) : \mathbb{R}^{N\times m} \longrightarrow \mathbb{R}^N$ to be our kernel functions with parameters $\theta_k \in \mathbb{R}^k$. For every target class, a new kernel network is generated such that each network learns to represent a unique cluster from the embedding output by the encoder.

In our design, each layer of the encoders implements an activation-like sine function $y_t = w_a \sin(2\pi w_f y_{t-1})$, where y_{t-1} is the output of the previous layer, and w_a and w_f are weight vectors of respective sizes $|w_a| = 1$ and $|w_f| = |y_{t-1}|$, which are learned by the network. The chosen sine activation-like functions enable the networks to learn faster and adapt to complex data structures. For the kernel networks, we use fully-connected neural models, whose outputs are passed through a softmax function $\sigma(y_i) = \exp(y_i)/\sum_{j=1}^{N}\exp(y_j)$, thereby returning the set of probabilities that a given sample belongs to different clusters. As such, with the kernel networks, we aim to map any embedding from the encoder to single values representing the likelihood estimations that samples belong to their respective clusters. For each sample, this mapping is done through a deep NN while simultaneously the kernel weights defined by these networks, i.e., cluster centers, are computed from the embeddings of all samples.

To train our AC network, we combine two distinct loss functions, i.e., $\mathcal{L} = \mathcal{L}_p + \mathcal{L}_c$, where $\mathcal{L}_p$ is the Mean Squared Error (MSE) between the estimated probabilities of samples belonging to clusters and the ground truth, and $\mathcal{L}_c$ is a contrastive loss that aims to control the distance between the different clusters. The latter is important when the number of clusters grows, hence they become closer to each other and harder to separate. To compute the MSE, we first perform one-hot encoding of the target cluster ids (c_i) as follows:

$$p_i = \begin{cases} 1, & \text{if } i = c_i, \\ 0, & \text{otherwise,} \end{cases}$$

then compute

$$\boldsymbol{\mathcal{L}_p} = \frac{1}{N}\sum_{i=1}^{N}(p_i - o_i)^2,$$

where $o = \{o_1, ..., o_N\} \in \mathbb{R}^N$ denotes the output probabilities of the model. $\mathcal{L}_c$ is designed to optimize the assignments of the clusters and is defined as

$$\mathcal{L}_c = Y \cdot d_S + (1 - Y) \cdot \max(0, \delta - d_D),$$

where d_S is the distance between all pairs of similar points and d_D is the distance between all pairs of dissimilar points, Y is a binary label indicating whether the pairs are similar (i.e., 1 if they should be deemed similar, 0 otherwise), and $\delta > 0$ is a margin defining the radius around the embedding space of a sample, so that dissimilar pairs only contribute to the loss if $d_D \leq \delta$. The training of the neural model combining the encoders and kernel networks is then performed end-to-end by running back-propagation over a suitable number of iterations.

By this approach, the embeddings learned by our AC algorithm are easily separable even by the simplest classifiers, as exemplified in Fig. 3. This indicates that the AC network automatically discovers optimal kernels to be learned, which constitutes one of its main advantages as compared to existing clustering methods. Additionally, using cluster centers provided by our approach as features given to classifiers inherently enhances privacy, since all members of the same class would possess the same feature values. In our experiments, we use the same NN structure and the same hyper-parameters. Only the number of sub-networks changes, as this is task-specific and corresponds to the number of output classes.

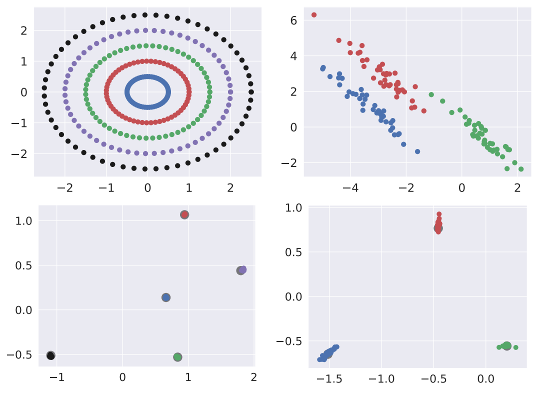
Fig. 3: Illustration of embeddings in a two-dimensional kernel space learned with our AC method: sample datasets (above) and their representations along with cluster centers (below).

VI. Implementation

We implement ACID in Python 3.7 using the PyTorch [35] and Scikit-Learn [36] libraries. For our AC algorithm we employ a set of encoders that are fully-connected NNs with 3 hidden layers comprising 500, 200, and 50 neurons, respectively. The number of neurons in the output layers is equal to the desired dimensionality of the kernels, which we set to 10 in our experiments. The kernel networks are also fully-connected, with 3 hidden layers of size 100, 50 and 30 neurons, respectively.[1] Inputs are processed in mini-batches of size 256 and we train the model using the Adam optimizer [37] with a learning rate of $1e^{-4}$. We adopt a Random Forest (RF) classifier due to its performance and computational efficiency, using default parameters, except the number of trees, which we set to 200. Our complete NIDS is trained over 100 iterations. To mimic computationally constrained edge devices, we execute all our experiments on a virtual machine running Ubuntu 18.04 LTS, with 4GB RAM, 50GB storage space, and a quad-core Intel(R) Celeron(R) N4100 CPU operating at 1.1 GHz. For the same reason, we perform no parallelization and no specific optimization of our clustering algorithm.

VII. Performance Evaluation

We first demonstrate the performance of our AC algorithm on general clustering tasks using five synthetic datasets, then evaluate ACID with three publicly-available network intrusion datasets. To completely cover all aspects of the NID task, we perform both binary and multi-label classifications on all datasets. We further use one of these datasets for a performance comparison with state-of-the-art NIDSs. Additionally, we perform a complexity and runtime analysis of our solution to understand its deployability on constrained edge devices.

A. Datasets

Synthetic Datasets: We generate five different artificial datasets covering a range of scenarios with different levels of complexity, including number of clusters/groups, shape, ambiguity, and distributions. Specifically, we consider

- *Two-circles:* A binary classification task with samples that fall into concentric circles. This is suitable for testing if an algorithm can learn complex non-linear manifolds.
- *Five-circles:* A multi-label classification problem with samples that fall into concentric circles. Similarly to the Two-circles dataset, this is also suitable for testing if an algorithm can learn complex non-linear manifolds.
- *Two-moons:* A binary classification problem consisting of samples falling into two interleaved half-circles. This dataset is suitable for testing if an algorithm can learn non-linear and intertwined class boundaries.
- *Blobs:* Groups of data-points with Gaussian distributions, which are suitable for assessing the ability of algorithms to solve linear classification problems.
- *Sine/Cosine:* The samples consist of sine and cosine data points. This dataset is suitable for testing if an algorithm can learn complex, non-linear, and intertwined class boundaries.

With these, we are able to compare the performance of our clustering approach against three popular clustering methods, and ascertaining its universal learning abilities. For visualization purposes, each of these synthetic datasets consists of samples of 2-dimensional data points, whose features correspond to their Cartesian coordinates, and each sample is assigned a label corresponding to its cluster ID.

[1]The source code of our implementation is available at `https://github.com/Mobile-Intelligence-Lab/ACID`

Intrusion Detection Datasets: To showcase the performance of our NIDS, we use three datasets that capture a total of 40 types of network attacks collected over a span of 20 years, namely KDD Cup'99 [16], ISCX-IDS 2012 [17], and CSE-CIC-IDS 2018 [18]. The KDD Cup'99 dataset was produced by MIT Lincoln Labs in a LAN operated similarly to an Air Force environment over the course of 9 weeks, during which raw TCP data was collected [38]. The CSE-CIC-IDS 2018 dataset is the result of a controlled attacks campaign run by the Canadian Institute for Cybersecurity using 50 machines that targeted a victim organization with 5 departments, involving 420 machines and 30 servers [39].

For comparison purposes, we evaluate all the benchmarks considered on a random subset of the ISCX-IDS 2012 dataset, which was released by the University of New Brunswick and consists of seven days of raw network data, including benign and four types of malicious network traffic, namely BruteForce SSH, DDoS, HttpDoS, and Infiltration [40].

B. Preprocessing

Based on the features extracted from raw packets (including fields in the packet headers and statistical attributes), we generate bidirectional flows, thereby incorporating more temporal information than what can be observed from individual packets (e.g., the time interval between two sequential packets). The first packet of each flow determines its direction, i.e., forward (source to destination) or backward (destination to source).

To obtain a relatively balanced ISCX-IDS 2012 dataset for benchmarking, we randomly select a predefined number of malicious and benign traffic samples. The preprocessed dataset is then divided into a training and a testing set using a 70/30 split ratio. As an example, we illustrate the preprocessing results obtained in Table I.

Category	Count	Percentage	Training Count	Testing Count
Benign	10,000	28.28%	7,040	2,960
BFSSH	7,042	19.92%	4,960	2,082
DDoS	4,963	14.03%	3,476	1,487
HttpDoS	3,427	9.70%	2,431	996
Infiltration	9,925	28.07%	6,903	3,022
Total	35,357	100%	24,810	10,547

TABLE I: ISCX-IDS 2012 dataset after preprocessing.

C. Evaluation Metrics

To measure the performance of our NIDS, we use the held out testing set to compute confusion matrices, based on which we calculate the number of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) inferences. With these, we derive a number of metrics that allow us to assess the quality of the classification results of ACID and those produced by the benchmarks considered, namely:

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

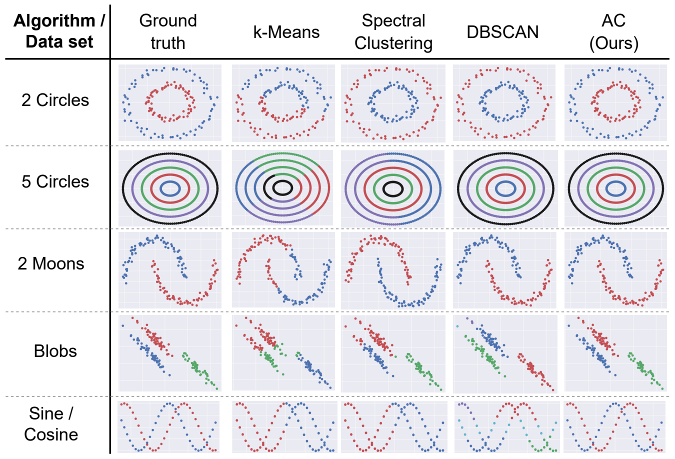

Fig. 4: Comparison of clustering results of the proposed AC method and three popular benchmarks on two-circles, five-circles, two-moons, blobs, and sine/cosine datasets.

$$\textbf{Precision} = \frac{TP}{TP+FP}; \quad \textbf{Recall} = \frac{TP}{TP+FN};$$
$$\textbf{FAR} = \frac{FP}{FP+TN}; \quad \textbf{F}_1\textbf{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

In the above, FAR is the false alarm rate.

D. General Clustering Results

We first compare the clustering performance of our AC network against that of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [41], Spectral Clustering [42], and k-Means [43], which are widely-used clustering approaches. For fairness, we specifically tune the parameters of each of the selected benchmark to optimize their performance. As observed in Fig. 4, Spectral Clustering obtains optimal results with the two-circles and the two-moons datasets, while DBSCAN performs very well on three out of the five datasets, i.e., two-circles, five-circles, and two-moons. Both approaches partially misclassify the blobs and consistently fail to cluster correctly the sine/cosine dataset. The k-Means clustering algorithm, however, systematically fails on all these tasks.

In contrast, our AC approach flawlessly clusters the data points in all the datasets considered, regardless of shape, distribution, or complexity (Fig. 4, rightmost column). This demonstrates the key advantage of using kernel networks to identify cluster centers and augmenting the feature set with information about these centers in view of classification.

E. Network Intrusion Detection Results

Next, we evaluate the performance of our complete NIDS. Recall that ACID extrapolates meaningful low-dimensional representations from header and statistical features extracted from raw network traffic data. Using these automatically learned features, we determine different cluster centers and use them to extend the header and statistical attributes. With these additional features, we expect our classifier to be more accurate and easily distinguish even the most similar patterns. To verify our hypothesis, we perform binary and multi-label classification on the three real-world datasets mentioned above, where we distinguish benign/malicious traffic flows, and respectively identify the type of every single traffic flow.

We also experimentally compare our approach with recent NIDS designs based on neural networks. In particular, we perform classification using DAGMM [13], which employs Gaussian Mixture Models (GMMs) to learn low-dimensional

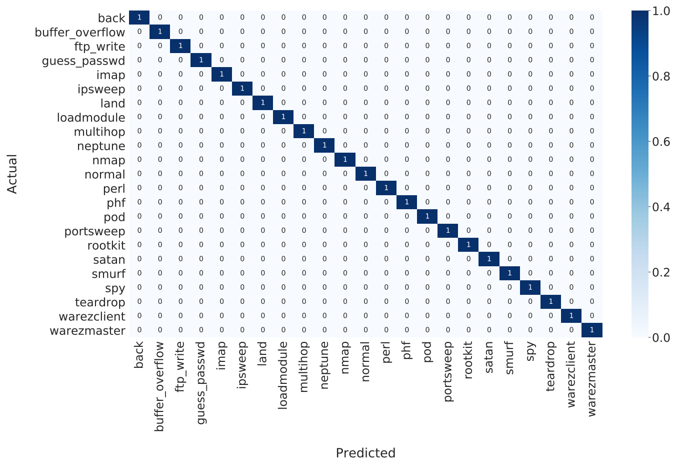

Fig. 5: Normalized confusion matrix for multi-label classification using ACID on the KDD Cup'99 dataset.

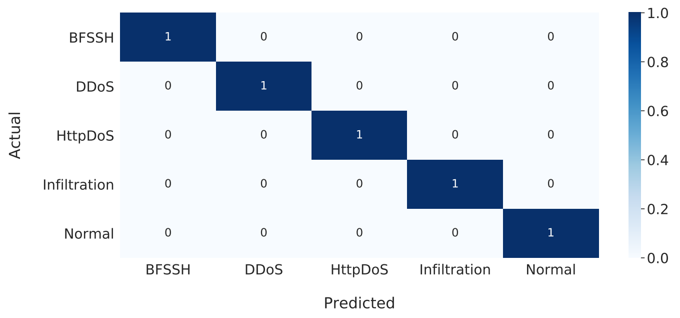

Fig. 6: Normalized confusion matrix for multi-label classification using ACID on the ISCX-IDS 2012 dataset.

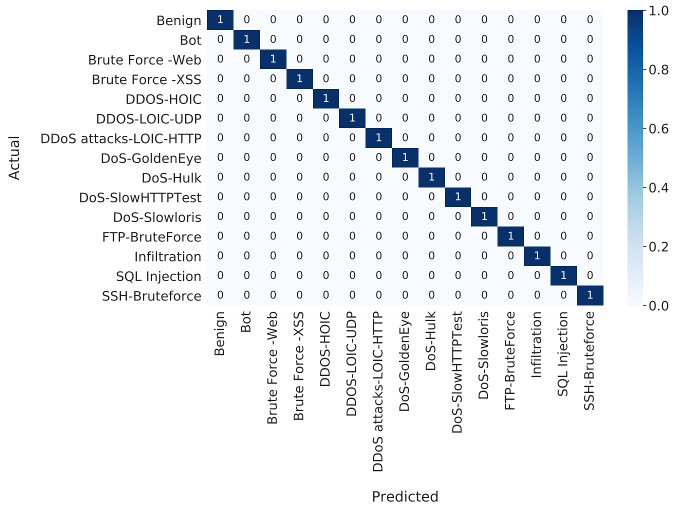

Fig. 7: Normalized confusion matrix for multi-label classification using ACID on the CSE-CIC-IDS 2018 dataset.

Metric Dataset	Accuracy (%)	FAR (%)	F_1 (%)	Classes	Samples
KDD CUP'99	100.0	0.00	100.0	23	43,510
ISCX-IDS 2012	100.0	0.00	100.0	5	10,547
CSE-CIC-IDS 2018	100.0	0.00	100.0	15	144,772

TABLE II: Performance summary of ACID on the KDD CUP'99, the ISCX-IDS 2012, and the CSE-CIC-IDS2018 datasets for the multi-label classification task.

embeddings of complex data structures while avoiding undesired local optima; N-BaIoT [14], which relies on Auto-Encoders (AEs) to discriminate IoT traffic; Deep NNs [15], which offer provably high performance despite using a simple architecture; and TR-IDS [24], which exploits payload contents to enhance NID performance.

In Figs. 5, 6, and 7, we provide normalized confusion matrices obtained with ACID, demonstrating its performance on the KDD Cup'99, ISCX-IDS 2012, and CSE-CIC-IDS 2018 datasets, respectively. Observe that our approach produces perfect results in the multi-label classification task, even where some network traffic flows may be very similar, e.g., Distributed Denial-of-Service (DDoS) attacks and high volume benign traffic. ACID correctly classifies 100% of the the traffic flows when using both kernel and payload features. Accuracy degrades only marginally when payload features are not employed for classification, specifically 99.41% accuracy is attained on the CSE-CIC-IDS 2018 dataset in this scenario.

We also compute the accuracy, precision, recall, F_1-score, and FAR for all datasets. The results confirm that ACID attains 100% accuracy, 0% FAR, and 100% F_1-score, when performing both binary and multi-label classification, irrespective of the number of classes. Multi-label classification results are summarized in Table II. These remarkable performance can be attributed to the manner in which our approach acts on the data, which is akin to a two-stage classification process, where the first stage corresponds to classifying network traffic via clustering, and the second corrects the misclassified samples through a further classifier. These results also confirm that our learned features consist of transferable knowledge across all samples in the respective datasets.

We further juxtapose ACID with the benchmark NIDSs considered, when classifying traffic in the ISCX-IDS 2012 dataset. We limit this comparison to binary classification, which is the intended goal of most of these methods. The obtained results are shown in Table III, which reveals that ACID outperforms existing solutions by up to 47% in terms of F_1-score. By combining a Text-CNN and RF, TR-IDS attains very good performance on the binary classification task, but unlike ACID, it struggles to discriminate malicious traffic flows of different types that are superficially similar. Specifically, in multi-label classification, TR-IDS misclassifies $\sim$1% of DDoS and $\sim$1% Infiltration as benign flows, while flagging more than 1% of benign flows as attacks. In practice, this would not only lead to manual inspection of large numbers of flows, but allow attacks with dramatic consequences (e.g., Infiltration) to succeed, and potentially block traffic with commercial value.

Approach	Payload-based Features	Accuracy (%)	FAR (%)	F_1 (%)
DAGMM [13]	No	62.91	30.65	53.07
N-BaIoT [14]	No	89.19	10.80	89.19
Deep NN [15]	No	88.14	7.41	70.35
TR-IDS [24]	Yes	98.88	1.12	98.87
ACID (ours)	No	99.78	0.23	99.44
ACID (ours)	Yes	**100.0**	**0.00**	**100.0**

TABLE III: Comparison of our ACID with existing methods on binary classification with ISCX-IDS 2012 dataset.

Payload Features	Number of Parameters	Batch size	Model Complexity (MFLOP)	Execution Time (seconds)
No	789,855	1	1.49	0.08 ± 0.01
		128	191.68	0.10 ± 0.02
Yes	942,460	1	25.71	0.19 ± 0.04
		128	3291.43	18.59 ± 0.74

TABLE IV: Computational complexity of our clustering approach to NIDS, with and without payload features. Experiments on an emulated constrained device as defined in Sec. VI.

F. Complexity

Having demonstrated exceptional NID performance, we now study the complexity of our ACID approach, both from computational and runtime perspectives. To this end, we count the number of parameters of our neural model and the number the floating point operations (FLOP) performed per inference. We also measure the inference time for a *single sample* and a batch of *128 samples*, respectively, accounting for all the processing undergone by packet through ACID's complete pipeline. The results obtained are reported in Table IV, where we also assess the additional complexity incurred when using payload features. Given that we use an edge device emulation set-up (described in Section VI) and inference times can be as low as 80ms, we conclude that it is feasible to deploy our ACID system on constrained edge devices for intrusion detection purposes. Further, increasing the batch size to 128 reduces the runtime per sample by 100×. We also note that payload features incur $\sim$2× higher execution time per single flow inference, while they can prove orders of magnitudes more costly when working with proportionally larger batches, which needs to be accounted for when deploying at the network edge.

G. Sensitivity Analysis

We conclude with a sensitivity analysis of our AC algorithm wrt. *(i)* the dimensionality of the learned representations, and *(ii)* the importance of cluster centers in the final classification.

1) Kernel Size: In the first experiment, without changing any other parameter of our model, we vary the kernel size and evaluate its impact on the quality of the clustering. Specifically,

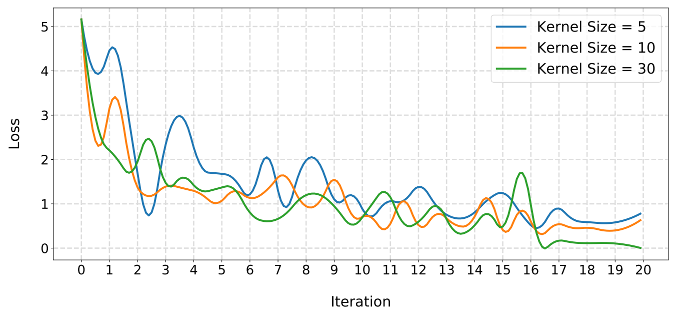

Fig. 8: Evolution of loss function values for different kernel sizes when training our AC on the ISCX-IDS 2012 dataset.

we examine the loss curve during training when the kernel size is respectively 5, 10, and 30. The results are shown in Fig. 8.

Observe that the impact of the dimension of the kernels is relatively negligible, as our AC approach converges rapidly to small loss values that lead to efficient separation of data samples into different clusters. This observation is particularly valuable when considering deploying our NIDS on constrained devices. For this reason, working with 10 as the NIDS kernel size, as we did in all experiments reported in this paper, is reasonable. Also note that training our clustering method for only 20 iterations is sufficient to obtain state-of-the-art performance and reduce the False Alarm Rate (FAR) to zero.

2) Feature Importance: To better understand the impact of different features on the classification results, we analyze their degree of contribution to this process. Since we use the RF classifier in the final stage of ACID, its implementation in the Scikit-Learn library directly provides the importance weight of each feature, thereby making it easy to rank all features according to their importance score. From these aggregated features, we select the top-15 ones according to their relative importance to the decision process and plot them in Fig. 9. To appreciate the importance of the cluster centers relative to all header and statistical features extracted, as well as payload based features, we extract 50 features from the payloads, using two modern Natural Language Processing (NLP) techniques (word embedding and Text-CNN), as also performed in [24]. Recall that these payload features can help to detect malicious contents, such as those seen with payload-based attacks, i.e. SQL injection, cross-site scripting (XSS), and shell-code.

Finally, we perform the same experiment excluding the payload features and observe that the cluster centers extracted from our clustering algorithm significantly outweigh all other features during the decision process. More specifically, the cluster centers contribute to the decision process by 5.77% to 9.03% (almost 2 to 3.6 times more than the most important of all header and statistical features combined). Furthermore, the clusters obtained by our AC approach to NID provide perfectly separable representations of our data points for all network traffic categories, which is confirmed by the t-distributed Stochastic Neighbor Embedding (t-SNE) representation [44] of the clusters shown in Fig. 10. Indeed, reducing the dimension of embedded representations obtained by our model to 2 through this method reveals that the clusters

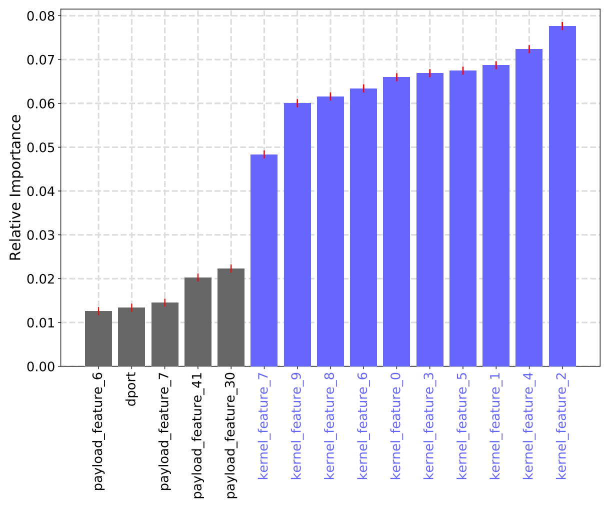

Fig. 9: Most important 15 features in the classification process on the ISCX-IDS 2012 dataset. 50 features extracted from the payloads in total.

corresponding to all types of attacks and benign traffic are clearly distinguishable to the clustering algorithm.

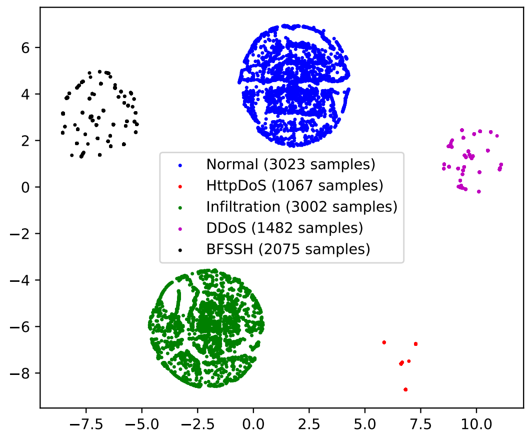

Fig. 10: (t-SNE) 2-D projections of clusters obtained by our AC approach for multi-label classification of ISCX-IDS 2012.

VIII. Conclusions and Future Directions

In this paper, we introduced a novel approach to Network Intrusion Detection (NID) based on an Adaptive Clustering (AC) neural network that achieves exemplary performance on three different datasets, in both binary and multi-label traffic classification tasks. Our design hinges on multiple kernel networks to learn optimal embeddings of data samples, thereby acquiring the ability to easily distinguish different types of network traffic. Through extensive experiments, we have proved the superiority of our clustering method over existing alternatives, and made the case for a lightweight and effective Network Intrusion Detection System (NIDS) that can be deployed on devices with limited computational resources, thereby strengthening defenses at the network edge.

References

[1] The Economist Intelligence Unit, “The IoT Business Index: A steep change in adoption,” Feb. 2020.

[2] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, “Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-scale IoT Exploitations,” *IEEE Comms Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[3] Federal Bureau of Investigation (FBI), “2019 Internet Crime Report,” Feb. 2020.

[4] Z. Inayat, A. Gani, N. B. Anuar, M. K. Khan, and S. Anwar, “Intrusion response systems: Foundations, design, and challenges,” *Journal of Network and Computer Applications*, vol. 62, pp. 53–74, 2016.

[5] Cisco, “Snort,” https://talosintelligence.com/snort.

[6] “Zeek,” https://zeek.org/.

[7] “Suricata,” https://suricata-ids.org/.

[8] H. Liu and B. Lang, “Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey,” *Applied Sciences*, vol. 9, no. 20, p. 4396, 2019.

[9] A. L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cyber security intrusion detection,” *IEEE Comms Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[10] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, “Speech recognition using deep neural networks: A systematic review,” *IEEE Access*, vol. 7, pp. 19 143–19 165, 2019.

[11] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[12] C. Zhang, P. Patras, and H. Haddadi, “Deep learning in mobile and wireless networking: A survey,” *IEEE Comms Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[13] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, “Deep autoencoding gaussian mixture model for unsupervised anomaly detection,” in *ICLR*, Mar. 2018.

[14] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, “N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders,” *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, May 2018.

[15] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, “Deep learning approach for intelligent intrusion detection system,” *IEEE Access*, vol. 7, pp. 41 525–41 550, Apr. 2019.

[16] “KDD Cup’99,” http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[17] “ISCX-IDS 2012,” https://www.unb.ca/cic/datasets/ids.htm.

[18] “CSE-CIC-IDS 2018,” https://registry.opendata.aws/cse-cic-ids2018.

[19] Y. Yu, J. Long, and Z. Cai, “Session-Based Network Intrusion Detection Using a Deep Learning Architecture,” in *International Conference on Modeling Decisions for Artificial Intelligence*, Sept. 2017, pp. 144–155.

[20] ——, “Network Intrusion Detection through Stacking Dilated Convolutional Autoencoders,” *Security and Communication Networks*, pp. 1–10, Nov. 2017.

[21] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, “Autoencoder-based feature learning for cyber security applications,” in *IEEE International joint conference on neural networks*, May 2017, pp. 3854–3861.

[22] Z. Tan, A. Jamdagni, X. He, P. Nanda, R. Liu, and J. Hu, “Detection of Denial-of-Service Attacks Based on Computer Vision Techniques,” *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2519–2533, May 2014.

[23] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, “HAST-IDS: Learning Hierarchical Spatial-Temporal Features using Deep Neural Networks to Improve Intrusion Detection,” *IEEE Access*, Dec. 2017.

[24] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, “TR-IDS: Anomaly-Based Intrusion Detection through Text-Convolutional Neural Network and Random Forest,” *Security and Communication Networks*, pp. 1–9, July 2018.

[25] S. Nejatian, H. Parvin, and E. Faraji, “Using sub-sampling and ensemble clustering techniques to improve performance of imbalanced classification,” *Neurocomputing*, vol. 276, pp. 55–66, 2018.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[27] M. Jianliang, S. Haikun, and B. Ling, “The Application on Intrusion Detection Based on K-means Cluster Algorithm,” *IEEE Information Technology and Applications*, vol. 1, pp. 150–152, May 2009.

[28] Y. Guan, A. Ghorbani, and N. Belacel, “Y-means: a clustering method for intrusion detection,” in *IEEE Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology*, vol. 2, June 2003, pp. 1083– 1086.

[29] Z. Mingqiang, H. Hui, and W. Qian, “A graph-based clustering algorithm for anomaly intrusion detection,” in *IEEE Computer Science & Education*, July 2012, pp. 1311–1314.

[30] Z. Li, Y. Li, and L. Xu, “Anomaly Intrusion Detection Method Based on K-Means Clustering Algorithm with Particle Swarm Optimization,” in *IEEE Information Technology, Computer Engineering and Management Sciences*, vol. 2, Sept. 2011, pp. 157–161.

[31] F. Hachmi and M. Limam, “A two-stage technique to improve intrusion detection systems based on data mining algorithms,” in *IEEE International Conference on Modeling, Simulation and Applied Optimization*, Apr. 2013, pp. 1–6.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” *ICLR Workshop*, 2013.

[33] T. He, W. Huang, Y. Qiao, and J. Yao, “Text-attentional Convolutional Neural Network for Scene Text Detection,” *IEEE Transactions on Image Processing*, vol. 25, no. 6, pp. 2529–2541, 2016.

[34] L. Breiman, “Random Forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[35] “PyTorch,” https://pytorch.org/.

[36] “Scikit-Learn,” https://scikit-learn.org/stable/.

[37] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” 2014.

[38] J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, “Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection,” *Results from the JAM Project by Salvatore*, pp. 1–15, 2000.

[39] A. Shiravi, H. Shiravi, M. Tavallaee, and A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *Computers & Security*, vol. 31, no. 3, p. 357–374, May 2012.

[40] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” in *International Conference on Information Systems Security and Privacy (ICISSP)*, 2018, pp. 108–116.

[41] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” in *KDD*, vol. 96, no. 34, 1996, pp. 226–231.

[42] F. R. Chung and F. C. Graham, *Spectral Graph Theory*. American Mathematical Soc., 1997, no. 92.

[43] J. MacQueen *et al.*, “Some Methods for Classification and Analysis of Multivariate Observations,” in *Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14, 1967, pp. 281–297.

[44] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.