Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.DOI

AD-CGAN: Anomaly Detection using Conditional Generative Adversarial Networks

OKWUDILI M. EZEME, (Member, IEEE), QUSAY H. MAHMOUD, (SENIOR MEMBER, IEEE) AND AKRAMUL AZIM (SENIOR MEMBER, IEEE)

Department of Electrical, Computer and Software Engineering
Ontario Tech University, Oshawa, ON L1G 0C5 Canada
(e-mail: {mellitus.ezeme, qusay.mahmoud, akramul.azim}@ontariotechu.net)

Corresponding author: Okwudili Ezeme (e-mail: mellitus.ezeme@ontariotechu.net).

ABSTRACT Whether in the realm of software or hardware, datasets that represent the state of systems are mostly imbalanced. This imbalance is because the reliability requirements of these systems make the occurrence of an anomaly a rare phenomenon. Hence, most datasets on anomaly detection have a relatively small percentage that captures the anomaly. Recently, generative adversarial networks (GAN) have shown promising results in image generation tasks. Therefore, in this research work, we build on the ability of conditional GANs (CGAN) to generate plausible distributions of a given profile to solve the challenge of data imbalance in anomaly detection tasks and present a novel framework for anomaly detection. Firstly, we learn the pattern of the minority class data samples using a *single class CGAN*. Secondly, we use the knowledge base of the single class CGAN to generate samples that augment the minority class samples so that *a binary class CGAN* can train on the typical and malicious profiles with a balanced dataset. This approach inherently eliminates the bias imposed on algorithms from the dataset and results in a robust framework with improved generalization. Thirdly, the binary class CGAN generates a knowledge base that we use to construct the cluster-based anomaly detector. During testing, we do not use the single class CGAN, thereby providing us with a lean and efficient algorithm for anomaly detection that can do anomaly detection on semi-supervised and non-parametric multivariate data. We test the framework on logs and image-based anomaly detection datasets with class imbalance. We compare the performance of AD-CGAN with similar cluster-based algorithms that have been tested on the same benchmark datasets, and AD-CGAN outperforms all the other algorithms in the standard metrics for measuring the efficacy of anomaly detection algorithms.

INDEX TERMS Anomaly Detection, Transfer Learning, Deep Learning, Generative Adversarial Networks

I. INTRODUCTION

In data mining and statistics, an outlier or anomaly refers to the characteristics of being different from others considered normal, and the field of anomaly detection is concerned with the detection of these deviants using a well defined statistical or machine learning procedure [1], [2]. According to [3], an anomaly differs from other samples by having features which suggest that the process that generated the anomalous sample is different from the process that generated the rest of the samples. Also, the presence of anomalies can be as a result of errors in software that generates the data, problems with the data collection methods or an outright attack on the application by agents that try to take control of the application. Therefore, an anomaly can be *point-based, radius-based* or *context-based* [4]. With the increasing integration of machine learning and deep learning frameworks into so many areas of computing and business intelligence, the anomaly detection practitioners still rely heavily on the use of signature-based anomaly detection mechanism because most of them trade *determinism* for *generalization*. However, deep learning based models offer the following advantages over the traditional algorithms [5], [6]: **a)** they scale better when the dataset is large, and because we are in the big data age, deep learning networks seem appropriate for anomaly detection especially in the online deviant behavior detection; **b)** there is no need for feature engineering by domain experts since the cascade of layers in the deep learning frameworks does automatic feature processing, thereby providing an end–

to-end model that takes raw data and returns a decision; **c)** deep learning models have outperformed traditional algorithms in processing time-series and image data. Despite all these advantages, the adoption of these deep learning models in anomaly detection is still not wide-spread because the evolving nature of the boundary between normal and deviant behaviors in several domains makes it difficult to demarcate with certainty, the anomalous and normal data from each other. Hence, the prevalence of the signature-based approaches despite the advantage of detecting zero-day vulnerabilities with anomaly models. Furthermore, considering that investigation and confirmation of an alarm to be a false alarm requires enormous resources both in terms of human and material capital, some companies often trade the risk of exploitation of zero-day vulnerabilities to committing resources into the investigation of false alarms. Some others use a combination of both the signature and model-based approaches depending on the criticality of the resource being protected and the financial outlay of the company. Recently, GANs [7] have emerged as an important area of deep learning with ground breaking results in mostly image, audio and video applications. There are different variants of GAN [8] and we build on [9] to design our AD-CGAN used for anomaly detection in imbalanced data. Since GANs draw samples from the latent space, this research has the ability to provide a more stable prediction since it does not suffer from cumulative errors that are inherent in linear models like the autoregressive integrated moving average (ARIMA) [10].

A. MOTIVATIONS AND CONTRIBUTIONS

Applications and cyber-physical systems are designed with high-reliability requirements. Therefore, most of the time, the data collected from these systems contains normal operating conditions and some occasional anomalous incidents. Because of the rarity of occurrence of these unusual incidents, many datasets used by researchers to model system profiles in this domain are highly imbalanced, resulting in the following consequences:

- **a)** bias is inherently introduced in the models built using the imbalanced data. Since random guess is guaranteed to return high accuracy due to the data imbalance, machine learning models appear redundant.
- **b)** adaptability of anomaly detection frameworks from one system to another with varying ratios of typical to anomalous samples returns poor performance because bias varies from one system to the other. This variability in the composition of the different datasets makes generalization difficult.

Therefore, our contributions in this research work are as follows;

- **a)** design and development of a CGAN for context modeling of *normal* and *malicious* profiles.
- **b)** creation of a multi-stage transfer learning framework using *CGANs* to generate distinct clusters in imbalanced data.
- **c)** introduction of a lightweight anomaly detector for anomaly detection in non-parametric multivariate data using the knowledge base of the CGANs.

We organize the rest of the paper into the following sections: Section II highlights some of the related works in this domain while Section III discusses the details of the AD-CGAN design. In Section IV, we conduct experiments on benchmark datasets to measure the performance of AD-CGAN and discuss the results of the experiments in Section V. Finally, we conclude the paper in Section VI and offer ideas for future work.

II. RELATED WORK

Intrusion and *anomaly* detection [2], [11], [12] are the two broad classifications of detecting deviant behaviors in a system or application. While intrusion detection techniques rely on the use of signatures to detect a misuse behavior by storing signatures of discovered anomalies, anomaly detection models construct contexts using known standard features, and labels any deviation from the constructed profile as anomalous. The obvious limitation of this signature-based approach is that zero-day vulnerabilities cannot be detected as it only searches for observed signatures, i.e. it emphasizes memorization over generalization. On the other hand, the anomaly-based approaches target both known and unknown anomalies and can detect zero-day vulnerability. Because our work centers on anomaly detection approach, we highlight mostly the works in this area.

In [13], [14], the authors used kernel events to build an offline anomaly detection model using some vector space model concepts and agglomerative clustering technique. Imputation techniques were also used to increase the scope of their model and reduce the incidents of false positives. However, the anomaly frameworks of [13], [14] lack temporal modeling, hence, does not capture the nature of the system process behavior which emits system calls in discrete, sequential mode. Reference [15] used deep LSTM models constructed from system logs to create anomaly detection models for detecting anomalies in logs from virtual machines. The LSTM model is augmented with a workflow model which helps to detect context switches but since it is a host-based anomaly framework, it lacks the fine granularity of process-based anomaly detection frameworks. Authors of [16] used textual information buried in console logs of a program to create an anomaly detection model using principal component analysis that analyzes user credentials and actions. Also, [12] constructed host-based anomaly detection models using Bayesian Networks that uses system call arguments and variables as its features. The model of [12] uses aggregate weighting to reduce the effect of inconsistent anomaly scores from different models, but they do not consider the temporal relationship amongst the sequence of the system calls. Also, in [17], [18], the hierarchical LSTM network is used to explore the understanding of relationships amongst the kernel event traces of an embedded system, but other features which ordinarily should yield a more representative model

like timestamps, CPU cycles, and system call arguments are skipped.

Authors of [19] have an anomaly model built using system call frequency distribution as well as clustering techniques to detect when processes deviate from their standard profiles. Similar to the anomaly frameworks of [13], [14], the model of [19] has no temporal modeling of the events and can only be used for post-mortem analysis. Furthermore, the authors of [20] used the statistical metric of entropy to implement an anomaly detection model for network logs, but this type of anomaly model is best suited for cases where the volume of logs determine if an anomaly has occurred as obtainable in denial of service attacks. Also, the model of [20]'s use of entropy as the discriminating factor makes the result non-specific as entropy values are not unique. In [21], a real-time systems anomaly detection model is designed using the principle of inter-arrival curves to detect anomalous traces in a log sequence. Again, this inter-arrival curve-based model ignores other properties of system calls and is suitable for offline analysis only. The authors of [22] used an optimization method of minimum debugging frontier sets to create a model for detection of errors/faults in software execution. Also, [23] used system call traces without arguments to create an anomalous profile detector using deterministic finite automaton (DFA). The authors assume that anomalous system call sequences have a local profile and that with the use of a locality frame, these anomalies can be detected. And this local profile assumption is the limitation of the work as sophistication in cyber attacks has shown that both local and non-local profiles are exploited in the design of anomalies. Authors of [23] however admitted that when the anomalous sequences are not concentrated in a burst; their algorithm cannot handle such scenarios. Reference [24] used techniques such as the counting of observed system calls, frequency distribution approaches, a rule-based technique called RIPPER and a Hidden Markov Model (HMM) to construct anomaly models to detect valid and irregular behavioral profiles. Furthermore, in [25], a sliding window approach is used to construct a tree of the possible routes of the system call sequences based on the observed behavior during the learning phase. Again, [24], [25] only consider the temporal ordering of the traces and all other parameters like the timing information and system call arguments are ignored, and their approaches cannot handle a previously unseen system call without reconstructing the whole model.

In [26], the authors implemented a generic GAN called MAD-GAN to do anomaly detection in multivariate data by using the linear combination of the residual errors of the generator and discriminator to create an anomaly threshold. This work is closely related to our work in principle but differs greatly in design and implementation. MAD-GAN uses the generic GAN architecture but we have a GAN architecture conditioned by an input sequence to create a classification model that evolves with the context of the application or process. MAD-GAN also differs from ours in terms of making anomaly decision. It uses both the *generator* and *discriminator* for both training and testing while we require only the *generator* during the testing phase. Also, instead of using residual errors of the GANs for anomaly decision as done in [26], we came up with a cluster-based anomaly detector. Finally, in [27], [28], the authors implemented an ensemble framework called based on the encoder-decoder model using attention layer to do anomaly detection in both the event and temporal stream of the system call properties. This work uses similar principle as our work but differs in architecture. Furthermore, [27], [28] algorithms' predictions could suffer from cumulative errors as a result of use of anomalous sub-sequence in the input samples.

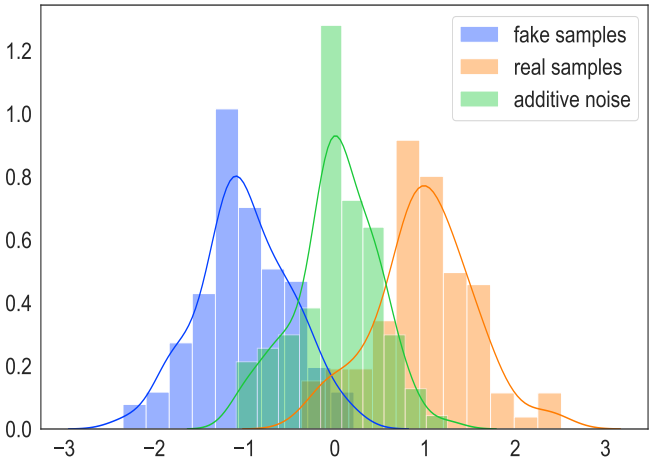

FIGURE 1: Visualizing the effect of Synthetic Noise Addition to the Discriminator Inputs

III. AD-CGAN: DESIGN AND IMPLEMENTATION

In this section, we discuss the design of AD-CGAN starting with the discussion of the qualities that a good anomaly detector should possess before we delve into the details of AD-CGAN design. We also include a list of symbols glossary for aid in deciphering the different concepts introduced in this section and beyond.

Imbalanced data creates overfitting and introduces poor generalization on the test data. Therefore, efficiency of the performance of an algorithm has to take into account the distribution of the different classes being *predicted* or *classified* to ensure that the performance being claimed actually performs better than a random guess. For example, a dataset that has *normal* class of $\mathbf{95}\%$ and *anomalous* class of $\mathbf{5}\%$ already has a $\mathbf{95}\%$ accuracy using human guess. Also, since the aim of anomaly detection frameworks is to detect that critical $\mathbf{5}\%$ of anomalous samples in this example, this $\mathbf{95}\%$ performance accuracy in this regard may not make sense without taking into account the *true negative, true positive* and other metrics that demonstrate how much of the critical samples were correctly detected. In some applications, the amount of human and capital resources required to investigate or deal with the effect of false positive classification is also huge, therefore, an effective anomaly framework should have the ability to

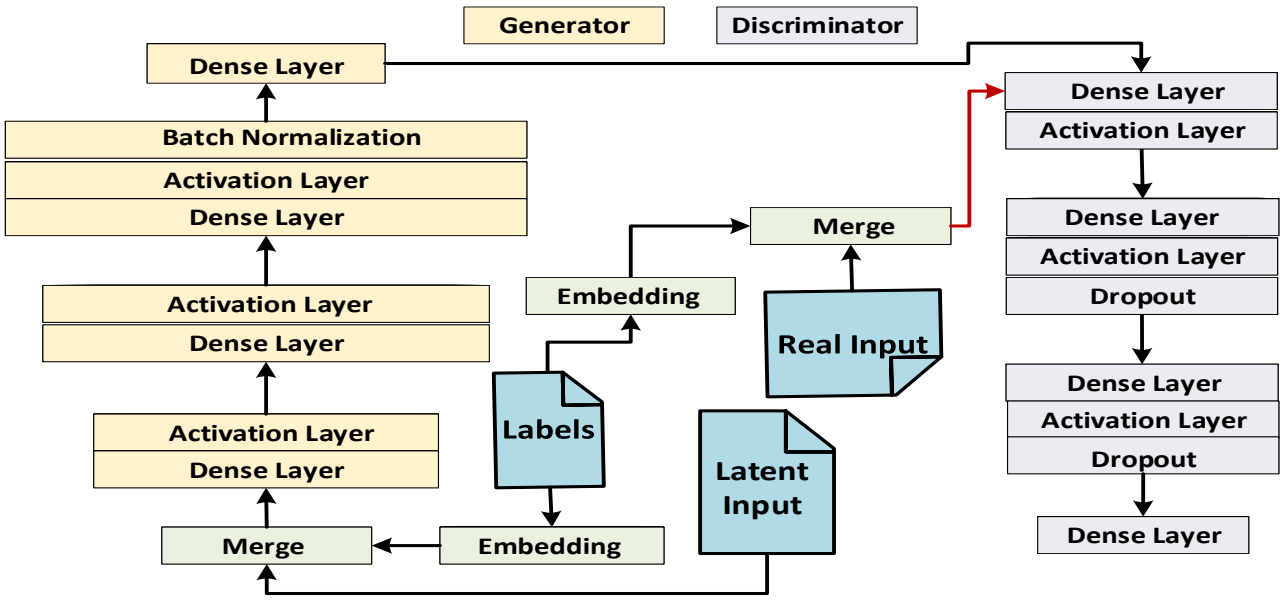

FIGURE 2: Architecture of the Conditional Generative Adversarial Networks (CGAN) of AD-CGAN

detect the anomalous samples while reducing the incidents of false positives. Having carefully considered the qualities that an anomaly detection framework should possess, we discuss the different modules of our *AD-CGAN*. First, we discuss the inner-workings of the *CGAN*s of Fig. 2 in Section III-A and in Section III-B, we discuss the complete architecture of the *AD-CGAN* as shown in Fig. 3.

List of Symbols

(m_j, n_j) embedding output tuple of sample j
C_i cluster i for $i \in \{0, 1\}$
D discriminator transformative function
G_b binary class GAN generator
G_s single class GAN generator
G generator transformative function
L_D discriminator loss
L_G generator loss
$P_{j|i}$ probability of point j choosing i as a neighbor
$(\bar{m}_i, \bar{n}_i)$ centroid cordinates of the detector for $i \in \{0, 1\}$
$\vec{x}_d$ real sample input to the discriminator
$\vec{y}_d$ discriminator output
$\vec{y}_g$ generator output
$\vec{y}$ discriminator input
$\vec{z}$ latent input sample to generator
$\widetilde{x}$ additive synthetic noise input
p_{2_i} euclidean norm to a cluster i
$\vec{c}$ conditional input sample

A. CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS

In this section, we discuss the modules of the CGAN with particular attention to the two main modules: *generator*, and *discriminator*.

1) Generator

Our generator is a stack of artificial neural networks in which the number of cells doubled for each succeeding layer. The dense layer are sandwiched with activation layers and the last layer before the output layer has a *batch normalization* layer as a *regularization* scheme that makes the network robust and improve the *generalization* properties of the network. Given a latent input, z, a conditional input, c, (1) is the generator equation where G is a non-linear function like an artificial neural networks. The output, $\vec{y}_g$ of (1) is a sequence of multivariate data representing the context being modeled.

$$G : (\vec{z}, \vec{c}) \longmapsto \vec{y}_g \tag{1}$$

The $\vec{y}_g$ can be discrete, continuous or a mixture of both. As an additional stabilization measure to avoid mode collapse and overfitting, the output, $\vec{y}_g$ is augmented with a white noise as described in Section III-A3 before it is fed to the discriminator.

2) Discriminator

In [29], it has been shown that some assumptions like finite log-likelihood-ratio between the generated data, $\vec{y}_g$ and the true value, $\vec{x}_d$, and the non-saturation of the Janson-Shannon divergence, $JS[\vec{y}_g|\vec{x}_d]$ to a maximum value do not hold in most cases. Therefore, an additive noise from a normal distribution with varying variance ensures that the discriminator does not overfit during training. We added the noise to each sample, $y_g \in \vec{y}_g$ and $x_d \in \vec{x}_d$ because adding it to the generated output only could also aid the discriminator in overfitting on the training data. In the discriminator model of Fig. 2, we use *dropout* in the second and third layers to reduce overfitting by reducing the *interdependent learning* amongst the neurons. In (2), we provide the equation which

captures the relationship between the input and output of the discriminator.

$$\boldsymbol{D} : (\vec{\boldsymbol{y}}, \widetilde{\boldsymbol{x}}) \longmapsto \vec{\boldsymbol{y}}_{\boldsymbol{d}} \text{ where } \vec{\boldsymbol{y}} \supseteq \vec{\boldsymbol{y}}_{\boldsymbol{g}}, \vec{\boldsymbol{x}}_{\boldsymbol{d}} \tag{2}$$

During training, AD-CGAN objective loss function that captures the relationship between the generator and discriminator is given in (3) where $\boldsymbol{L_G}$ and $\boldsymbol{L_D}$ represent the generator and discriminator loss respectively.

$$\begin{aligned} \boldsymbol{L_D} &= \boldsymbol{E}[\log(\boldsymbol{D}(\vec{\boldsymbol{x}}_{\boldsymbol{d}}, \widetilde{\boldsymbol{x}}))] \\ &\quad + \boldsymbol{E}[\log(\boldsymbol{1} - \boldsymbol{D}(\boldsymbol{G}(\vec{\boldsymbol{z}}, \vec{\boldsymbol{c}}), \widetilde{\boldsymbol{x}}))] \\ \boldsymbol{L_G} &= \boldsymbol{E}[\log(\boldsymbol{D}(\boldsymbol{G}(\vec{\boldsymbol{z}}, \vec{\boldsymbol{c}}), \widetilde{\boldsymbol{x}})))] \end{aligned} \tag{3}$$

3) Controlling Discriminator Overfitting

In Fig. 1, we show the effect of adding synthetic noise to both the generator output, $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ and the real samples input, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$. One of the underlying assumptions of GANs is that the log-likelihood ratio $\log \frac{\vec{y}_g(\vec{y_d})}{\vec{x}_d(\vec{y_d})}$ is finite. However, in some chaotic real world scenarios, given $\boldsymbol{G} : (\vec{\boldsymbol{z}}, \vec{\boldsymbol{c}}) \longmapsto \vec{\boldsymbol{y}}_{\boldsymbol{g}}$ where the support is $\{\boldsymbol{z} \in \vec{\boldsymbol{z}}, \boldsymbol{c} \in \vec{\boldsymbol{c}} : \boldsymbol{G}(\boldsymbol{z}, \boldsymbol{c}) \neq \boldsymbol{0}\}$, the intersection between the support of the generator and the support of the distribution that produced $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ in high dimensional space may be $\emptyset$ if the underlying distributions are *degenerate*. Hence, the addition of synthetic noise as shown in (2) aims to create an overlapping support for the two underlying distributions. This ensures that the log-likelihood remains finite and Jensen-Shannon divergence produces a continuous function which does not saturate to a constant value. Thereby reducing overfitting in the discriminator.

B. AD-CGAN

The main blocks of Fig. 3 are *single class CGAN, binary class CGAN, Embedding,* and the *Detector*. In the following sections, we highlight the role of each modules in the overall functionality of the *AD-CGAN*.

1) Single Class CGAN

Given a dataset for training that consists of *normal* and *anomalous* profiles, we determine the number of samples of each category. If the ratio of the majority class to the minority class is high and could introduce bias in the model, we learn the profile of the minority class using the *single class CGAN*. With the the single class CGAN trained on the minority class, we use it to generate more data samples of the minority class to augment the number of the minority class samples until the data samples of the majority and minority class samples are fairly even. The aim of the single class CGAN is to train the generator, $\boldsymbol{G_s}$ with the multivariate data, $\vec{\boldsymbol{z}}, \vec{\boldsymbol{c}}$ so that the generator output, $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ and the real samples of the class of interest, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ when clustered, show evidence that $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ and $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ are drawn from the same underlying distribution. We measure this similarity with the use of the *stochastic neighbor embedding* algorithm [30]. As will be shown in Section IV, when the single class CGAN is sufficiently trained, the generator output, $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ and the real samples, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ form a single indistinguishable cluster.

2) Binary Class CGAN

The *binary class CGAN* trains on the *normal* and *anomalous* samples from the original and augmented samples from the *single class CGAN*. Unlike the the single class CGAN, the aim of the binary class CGAN is to train the generator, $\boldsymbol{G_b}$ with the multivariate binary class data, $\vec{\boldsymbol{z}}, \vec{\boldsymbol{c}}$ such that the anomalous samples from the generator output, $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$, and the anomalous samples of the real samples, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ form a single distinct cluster while the corresponding normal samples from both the generator and the real samples form a different, distinct cluster. Therefore, we can say that the major difference between the *single class CGAN* and the *binary class CGAN* is that the former trains to *merge* two samples that belong to the same class while the latter trains to *fuse* the same class samples in one cluster and *diverge* different class samples to a different cluster. Therefore, the output of the single class CGAN is a single cluster while the binary class CGAN produces two clusters.

3) Embedding

When we need to test some given samples, the binary class CGAN generates a matching number of samples comprising of both *fake normal* and *fake malicious* profiles, and this output, $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ is a multivariate data of the same dimension as the real samples, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$. Therefore, the number of samples generated by the binary class CGAN, $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ is the same as the number of the real samples, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$. The significance of using the binary class CGAN output in the embedding are: **a)** when plotted using a visualization tool during training, it gives us an idea of how the CGANs are able to learn the profiles of the data being studied; **b)** during testing, the *fake labels* used in the CGANs to generate $\vec{\boldsymbol{y}}_{\boldsymbol{g}}$ guide us in constructing the different centroids of each class by giving us the ability to separate the *fake normal* samples from the *fake malicious* samples in the generated data. But while the dimension of the real sample, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ is fixed, AD-CGAN can handle data of arbitrary dimension by *preprocessing* the arbitrary dimensional data to the dimension of the real samples, $\vec{\boldsymbol{x}}_{\boldsymbol{d}}$ using the sparse principal component analysis algorithm [31]. Furthermore, we use the stochastic neighbor embedding algorithm to ensure that points in the high-dimensional space correspond to nearby embedded low-dimensional points, and distant points in high-dimensional space correspond to distant embedded low-dimensional points. Given $\{\vec{\boldsymbol{y}}_{\boldsymbol{g}}, \vec{\boldsymbol{x}}_{\boldsymbol{d}}\} \in \vec{\boldsymbol{y}}$, (4) shows the neighbor embedding algorithm which we employ to compute the conditional probabilities, $\boldsymbol{P_{j|i}}$ that highlights the probability of point $\boldsymbol{j}$ choosing point $\boldsymbol{i}$ as a neighbor.

$$\boldsymbol{P_{j|i}} = \frac{\boldsymbol{exp}\left(-|y_i - y_j|^2 / \boldsymbol{2\sigma_i^2}\right)}{\sum_{\boldsymbol{k \neq i}} \boldsymbol{exp}\left(-|y_i - y_k|^2 / \boldsymbol{2\sigma_i^2}\right)} \tag{4}$$

To ensure symmetry, we compute the *joint* probability, $\boldsymbol{P_{ij}}$ of the similarity between points $\boldsymbol{i}$ and $\boldsymbol{j}$ using (5). Since

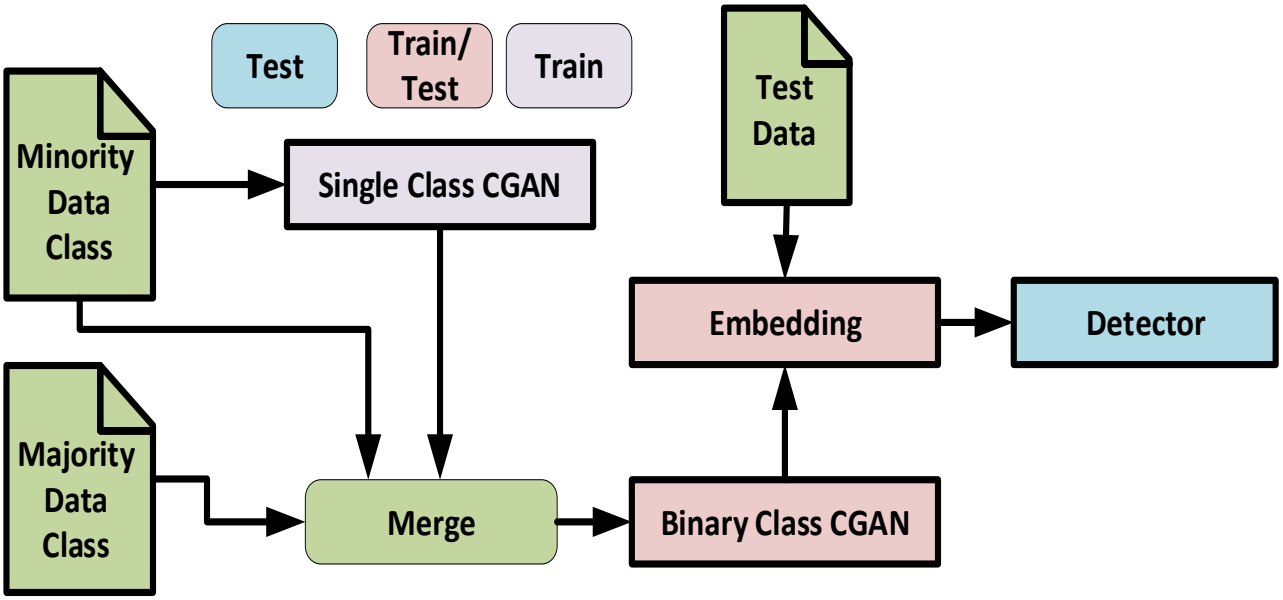

FIGURE 3: AD-CGAN Framework for Anomaly Detection in Balanced and Imbalanced Data

$\boldsymbol{P_{j|j} = 0}$, $\boldsymbol{P_{jj} = 0}$ in (5). In this work, our neighbor embedding algorithm of (4) produces embedding output in two dimensions so as to make it easy to use visualization tools to inspect the performance of the CGAN networks.

$$\boldsymbol{P_{ij}} = \frac{\boldsymbol{P_{j|i} + P_{i|j}}}{\boldsymbol{2N}} \text{ where } N = \text{number of rows of } \vec{\boldsymbol{y}} \quad (5)$$

4) Detector

From Fig. 3, (6) represents the embedding from higher to lower two-dimensional space. The resulting output is a set, $\boldsymbol{S = \{(m_1, n_1), (m_2, n_2), ..., (m_t, n_t)\}}$ of tuples of point coordinates of both the real and generated samples.

$$\boldsymbol{f : \mathbb{R}^{p \times q} \longmapsto \mathbb{R}^{p \times 2}} \text{ where } \boldsymbol{p} = \text{number of samples} \quad (6)$$

During evaluation, we use the labels used in the CGAN generator to separate the coordinates of the fake normal samples from the fake anomalous samples, and compute the centroid of each class using (7) on the fake samples only.

$$(\bar{\boldsymbol{m}}_i, \bar{\boldsymbol{n}}_i) = \left(\frac{1}{t} \sum_{j=0}^{t} \boldsymbol{m_{ji}}, \frac{1}{t} \sum_{j=0}^{t} \boldsymbol{n_{ji}} \right) \ \forall\, i \in \{0, 1\} \quad (7)$$

Using *t-statistics*, we derive the distance of new points being tested from the centroid using (8) where $\boldsymbol{\sigma_s}$ and $(\bar{\boldsymbol{m}}_s, \bar{\boldsymbol{n}}_s)$ are the evaluation sample standard deviation and mean respectively. Since the centroid is a vector, computing distance becomes simple vector subtraction standardized using the standard deviation as shown in (8). For simplification purposes, we use the norm of the output of (8) to measure the absolute distance from each of the two classes.

$$(\hat{\boldsymbol{m}}_i, \hat{\boldsymbol{n}}_i) = \left(\frac{\boldsymbol{m_i} - \bar{\boldsymbol{m}}_i}{\boldsymbol{\sigma_{s_{mi}}}}, \frac{\boldsymbol{n_i} - \bar{\boldsymbol{n}}_i}{\boldsymbol{\sigma_{s_{ni}}}} \right) \ \forall\, i \in \{0, 1\} \quad (8)$$

Therefore, anomaly detection decision is taken using (12) which selects the cluster with the highest probability score from the test point as the test point cluster. In (12), $\boldsymbol{C_k}$ is defined in (11).

$$\boldsymbol{p_{2_i}} = \sqrt{\hat{\boldsymbol{m}}_i^2 + \hat{\boldsymbol{n}}_i^2} \ \forall\, i \in \{0, 1\} \quad (9)$$

In cases where the distance is equal from the two clusters, we break the tie by using (11) to select a cluster for the test point. In (10), we compute the probability of the test point belonging to any of the clusters using the distances of (9).

$$\boldsymbol{P(i)} = \frac{\boldsymbol{p_{2_i}}}{\sum_i^n \boldsymbol{p_{2_i}}} \ \forall\, i \in \{0, 1\} \quad (10)$$

Then, we determine which cluster the test point belongs to using the probabilities of (10) and (11) in (12).

$$\boldsymbol{C_k} = \begin{cases} \boldsymbol{C_0 = 1 - C_1}, & \text{if } \boldsymbol{k = 0} \\ \boldsymbol{C_1}, & \text{if } \boldsymbol{k = 1} \end{cases} \quad (11)$$

Since the *single class CGAN* has been used to do data augmentation, the evaluation data samples used to generate the centroids are balanced, hence, the justification for the use of the Bernoulli probability distribution selection when a tie occurs. As the context changes, we update the centroids via *batch training* with the new data collected from the network.

$$\boldsymbol{C_i} = \begin{cases} \boldsymbol{C_0}, & \boldsymbol{P(1) < P(0)} \\ \boldsymbol{C_1}, & \boldsymbol{P(0) < P(1)} \\ \boldsymbol{C_k}, & \boldsymbol{P(1) == P(0)} \end{cases} \quad (12)$$

This parameter update can be done in parallel with the deployed algorithm. When a new set of model parameters are generated, we update the model by copying the new parameters to our deployed framework.

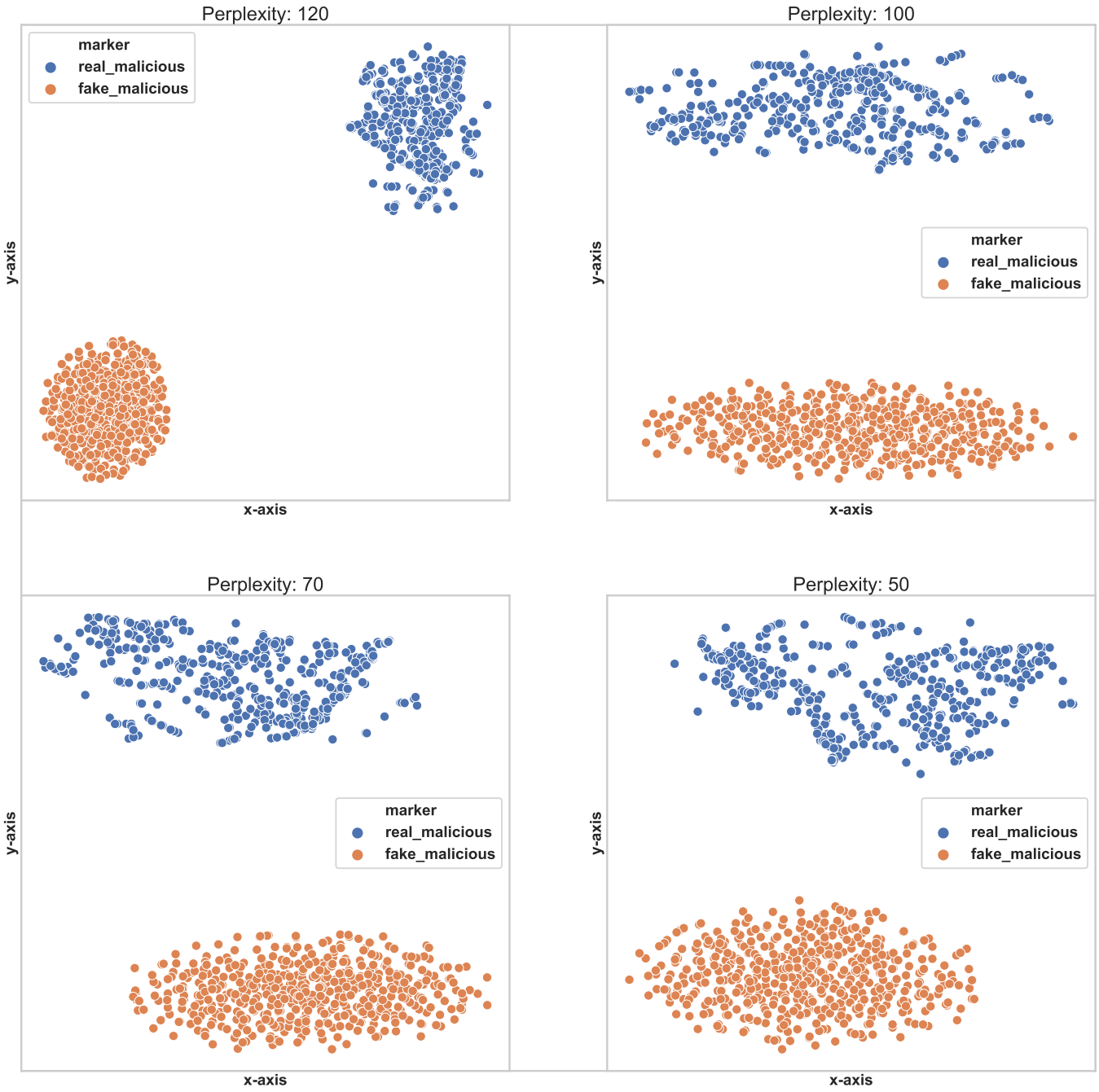

FIGURE 4: Before Training (ALOI): Real and Fake Samples of the Single Class CGAN form **2** Clusters

IV. EXPERIMENTS AND RESULTS

We test the *AD-CGAN* on two datasets: the *KDD99* network intrusion dataset and the *Amsterdam Library of Object Images (aloi)* [32] anomaly dataset. While the *KDD99* has the normal profile as the minority class, the *aloi* dataset has the anomalous profile as the minority class. Also, the two datasets selected for experiments on the framework comprises of image and network log data, thereby providing a diverse environment for measuring the effectiveness of the framework.

A. ALOI DATASET

In this dataset, there are a total of **50000** samples with **3.04**% of the total sample belonging to the anomalous class. This ALOI dataset has **27** multivariate features. With just **1508** anomalous samples out of the **50000** samples, it is difficult to train a model with this dataset without overfitting on the majority class. And since detection of the anomalous samples is more critical than detecting the normal profile, we will adopt the multi-stage process of data augmentation with the *single class CGAN* of Fig. 3.

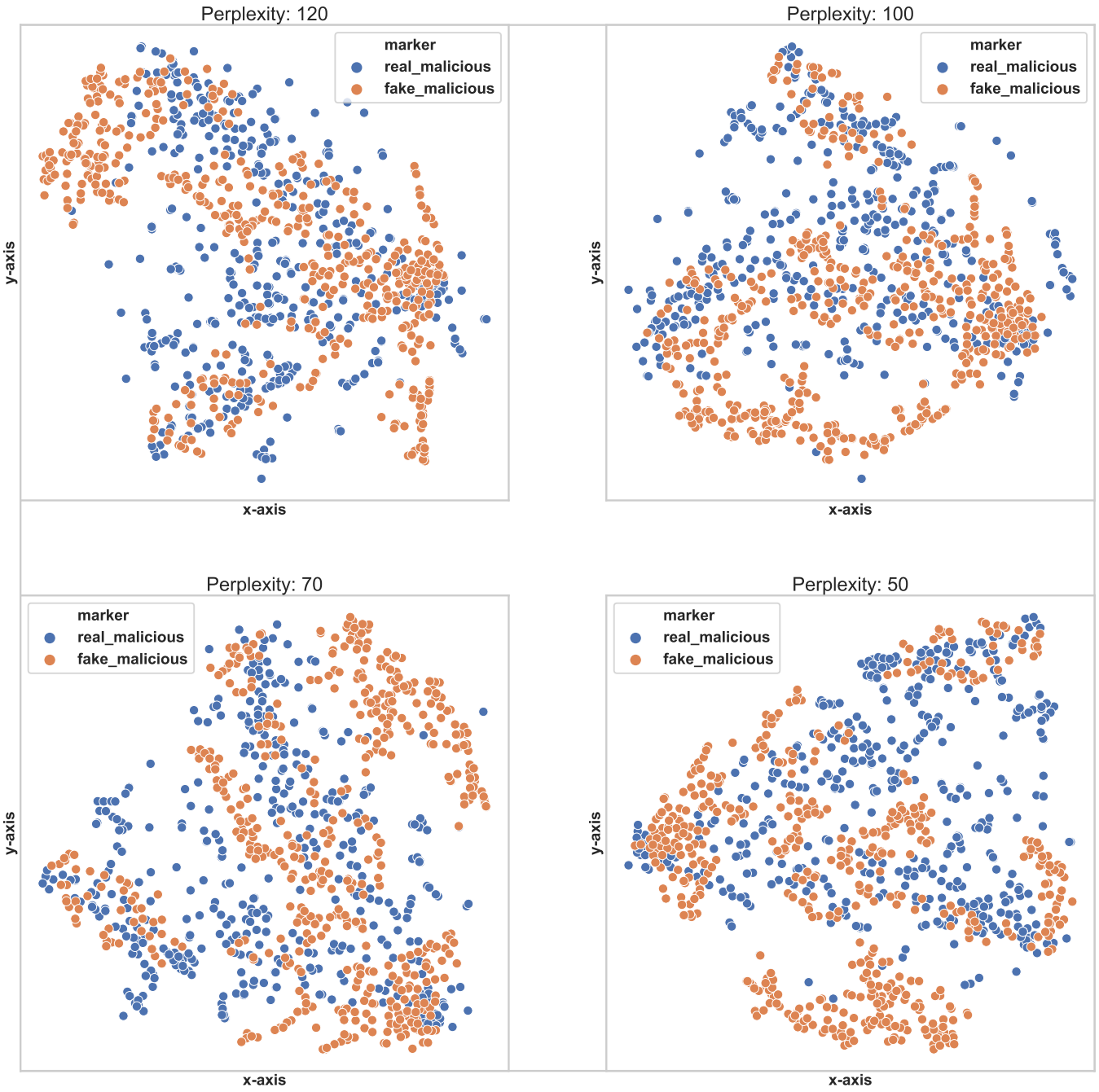

FIGURE 5: After Training (ALOI): Real and Fake Samples of the Single Class CGAN form **1** Cluster

1) ALOI Single Class CGAN Results

As highlighted in Section III-B1, the aim of the *single class CGAN* is to understand the profile of the data being trained such that when the *fake* and the *real* samples are subjected to the embedding algorithm, the result should be a single cluster which confirms how the single class CGAN has been able to understand the underlying distribution of the real samples. To measure the performance of the generator, we take snapshots of the model at intervals of **50** epochs during training. Thereafter, we compare the generated quality at each snapshot and the best model becomes our trained generator model. According to [33], the *tSNE* algorithm used for the embedding of Section III-B3 has a hyperparameter called *perplexity*[1], therefore, we generate the results under different perplexities to measure the stability of the model under varying perplexities.

In Fig. 4, the *fake* and *real* samples show two distinct clusters under different perplexities before we train the generator. After we are done training the generator, we can see that the

[1]Perplexity is a parameter used to determine the number of close neighbors each point has.

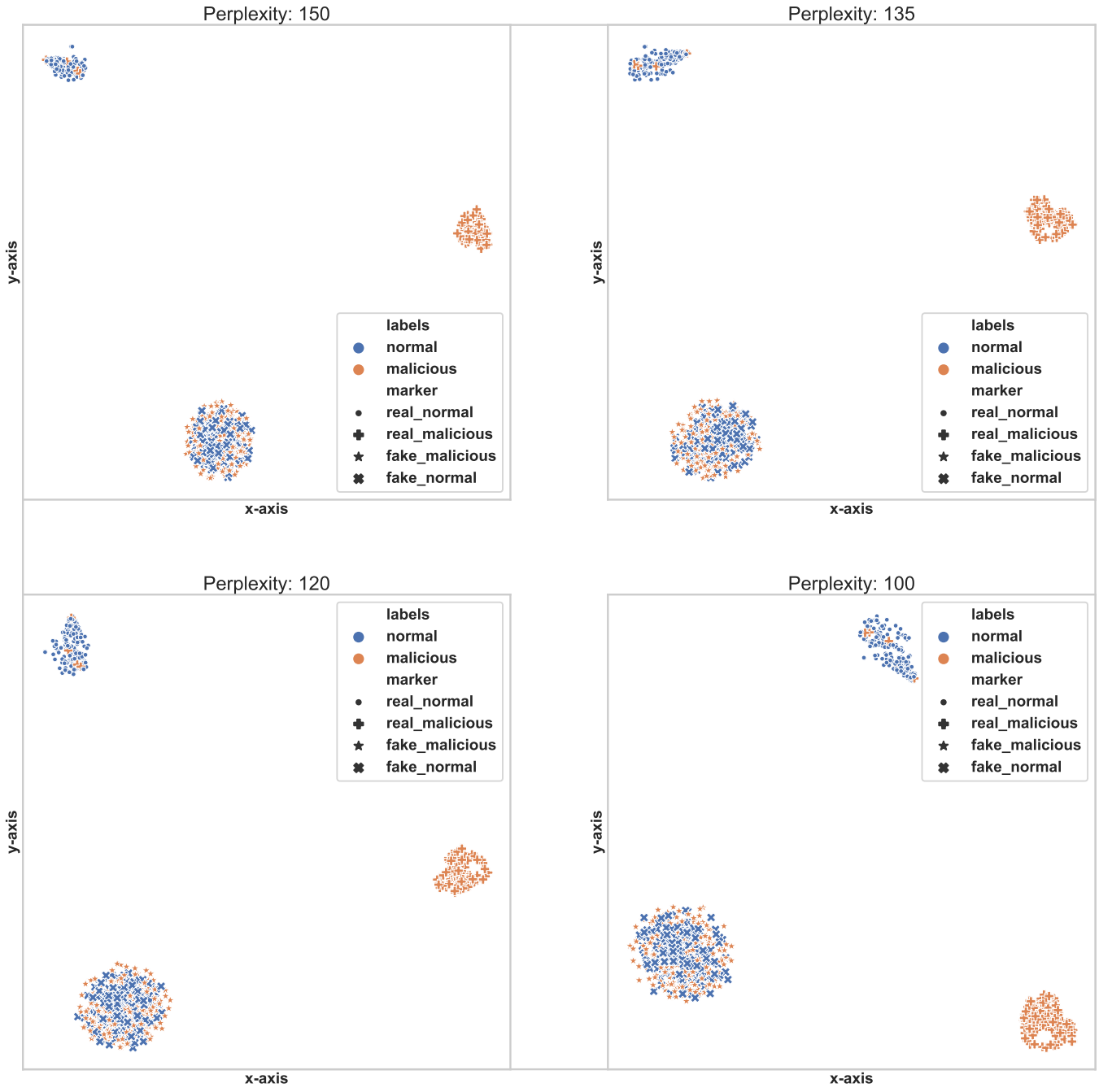

FIGURE 6: Before Training (ALOI): Binary Class CGAN generates Mixed Clusters for Malicious and Normal Profiles

generator is able to decipher the underlying distribution of the *real* samples as seen in Fig. 5 where the *real* and *fake* samples form one cluster. The ability of the single class CGAN to generate samples which are indistinguishable from the real samples confirms one of our hypothesis that we can use the generator to *augment the minority class data samples* and train the *binary class CGAN* with a balanced dataset.

2) ALOI Binary Class CGAN

The binary class CGAN takes input training data from both the original and augmented samples generated from the single class CGAN with the aim of producing the reverse behavioral expectation of the single class CGAN. As the name suggests, the binary class CGAN aims to understand the underlying distributions of both the malicious and normal data samples in such a way that the *fake* and *real* samples from each class belong to the same cluster when subjected to the embedding algorithm and the decision functions of Section III-B4. As seen in Fig. 6, the embedding output before training shows a mixture of the positive and negative class in the same cluster, thereby creating both false positive and false negative situation on both the *real* and *fake* samples.

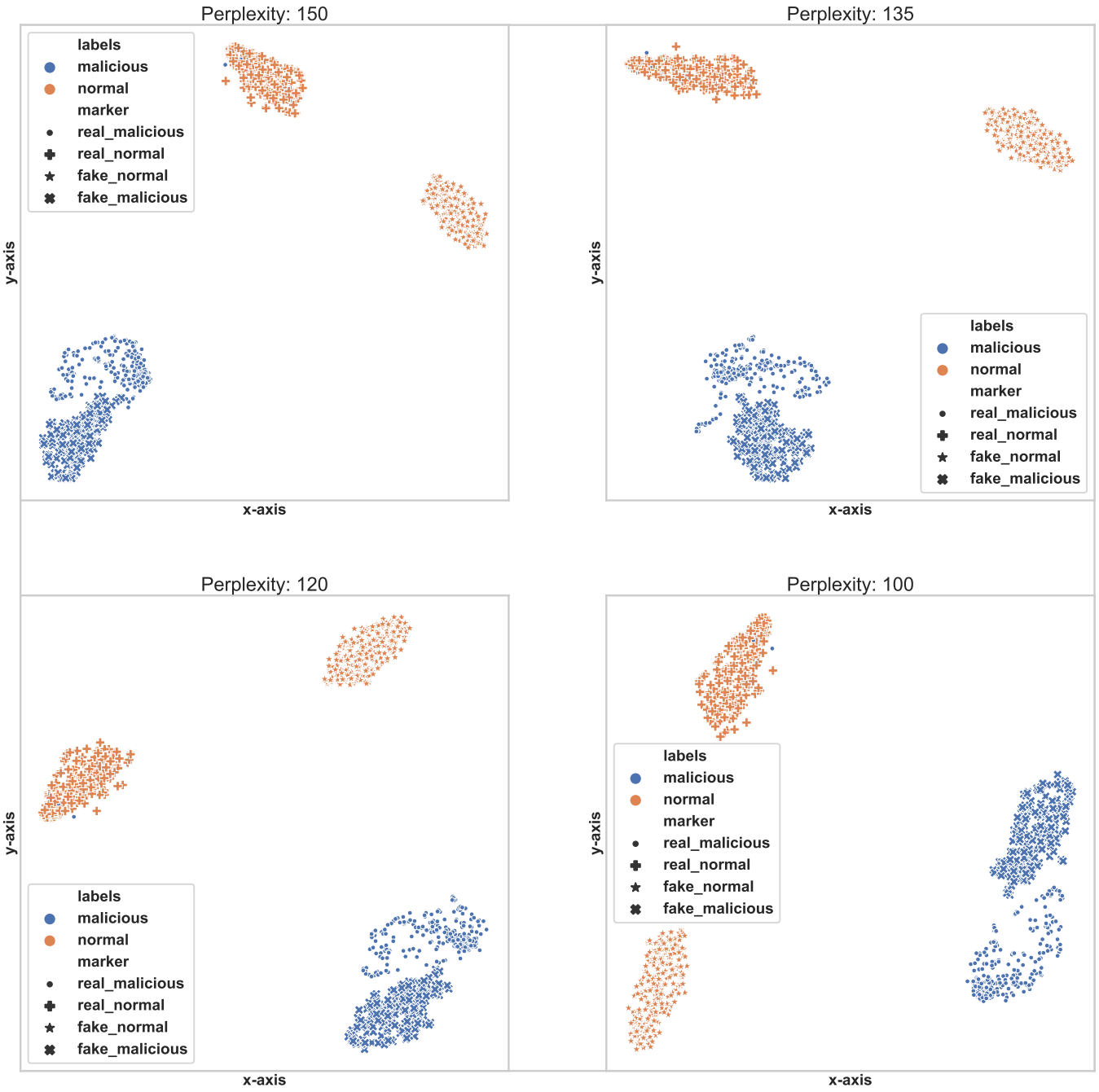

FIGURE 7: After Training (ALOI): Binary Class CGAN generates Distinct Clusters for the Malicious and Normal Profiles

In all our result analysis, we designate the malicious samples as the *positive class*. In Fig. 7, the results of the trained model show how the generator output and real test samples have successfully been separated into their respective classes. Although the snapshots of Fig. 7 does not show exactly two clusters as hypothesized, from the position of the *fake normal* and *fake malicious* data, we see that when we compute the centroids using (7), the closest samples are data points that belong to the same class from the test points. Thereby, confirming our hypothesis that the generator can be used to understand the different latent class distributions. And that an embedding of the *fake* and the *real* samples of each class should produce a single cluster that aims to classify samples from that particular class with a higher precision. Also, since the malicious class is our positive class, we can see from Fig. 7 that the malicious class forms a single cluster as hypothesized, and this result reduces the chance of false negative to almost zero as shown in Table 1 and Fig. 9. In Table 1, we show the different classification metric of the *ALOI* dataset alongside the total number of test samples for each class. The malicious class test sample consists of the *real* and *fake* samples from the single class CGAN. The

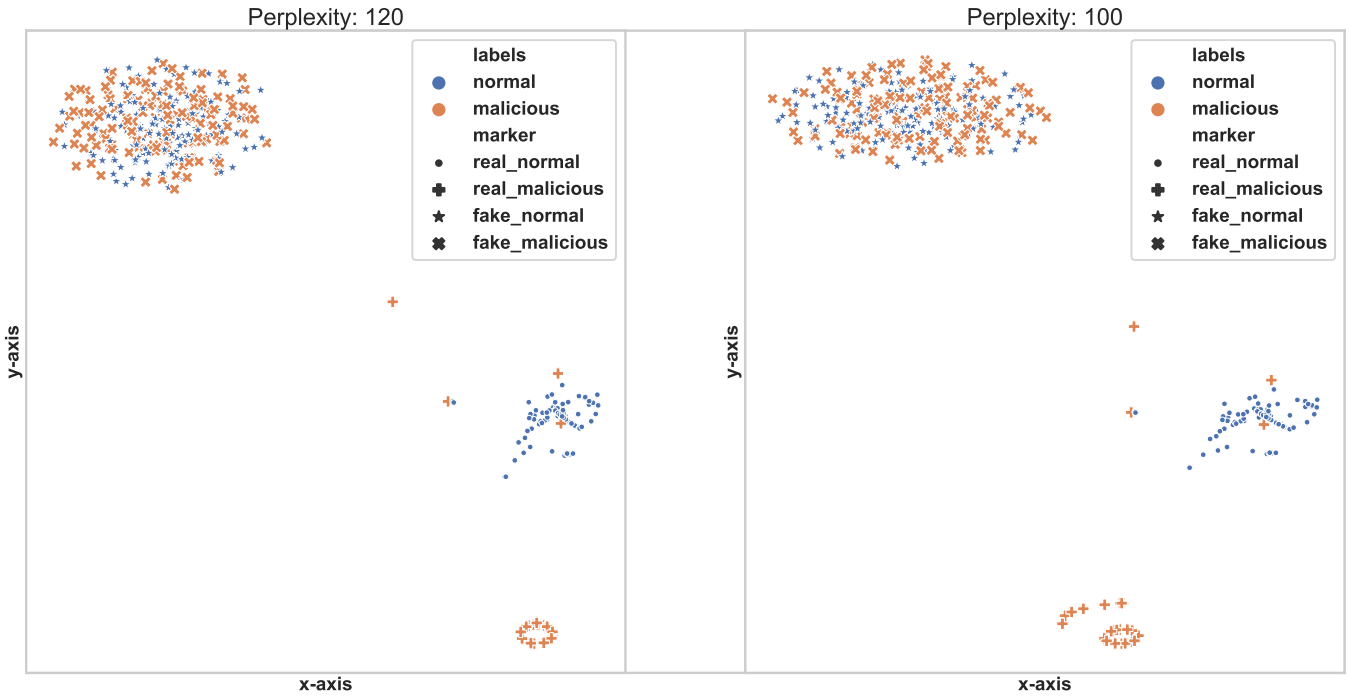

FIGURE 8: Before Training (KDD99): Binary Class CGAN creates Mixed Clusters for the Malicious and Normal Profiles

TABLE 1: Classification Report of AD-CGAN on the ALOI and KDD Dataset

Dataset	Class	Precision	Recall	F1 Score	No. Test Samples	Accuracy
ALOI	Normal	0.96945	0.98905	0.97915	10044	0.97885
	Malicious	0.98872	0.96856	0.97854	9956	
KDD	Normal	0.77749	0.99862	0.87429	74886	0.85663
	Malicious	0.99809	0.71507	0.83320	75114	

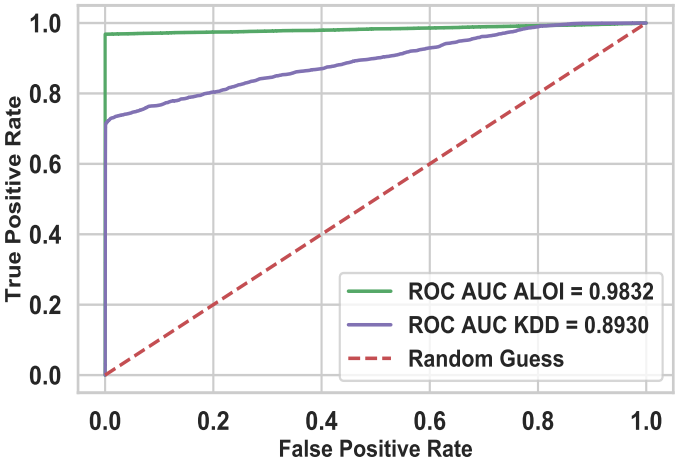

FIGURE 9: ROC Curve for the ALOI and KDD99 Dataset

perplexity hyperparameter that we use for this testing is **150**, and we sample **500** samples of the test data at a time. While the binary class CGAN generates samples during testing, the test report evaluates the performance of the model on the *real* input test samples only. The *fake* samples generated by the binary class CGAN during testing controls the position of the centroids. The authors of [34] have done a comprehensive evaluation of the performance of different algorithms on the most commonly used anomaly datasets. We compare the results of some algorithms used in [34] on the ALOI dataset to the results of *AD-CGAN* in Table 2. Since we are using the original dataset with **466** duplicates of the normal samples, we will be comparing with *clustering* algorithms results on the *normalized, duplicate* results of the ALOI dataset in [34]. Also, [34] did evaluation using different hyperparameters on the same algorithm but we report only the one with the highest receiver operating characteristics area under the curve (ROC AUC) value.

B. KDD99 DATASET

Unlike the ALOI dataset of Section IV-A, the *KDD99* dataset has more than two categories of data which are broadly classified into *malicious* and *normal* profiles. Since the test data has more categories of anomalies that are not available in the training data, we converted the whole data into two categories by collapsing any non-normal category to a new category called *malicious* profile. This way, AD-CGAN can be tested on any kind of normal or anomalous scenarios that were not available during the training phase. One of the strengths of *AD-CGAN* which we highlight in this test is the ability of the model to train on a small subset of the data and

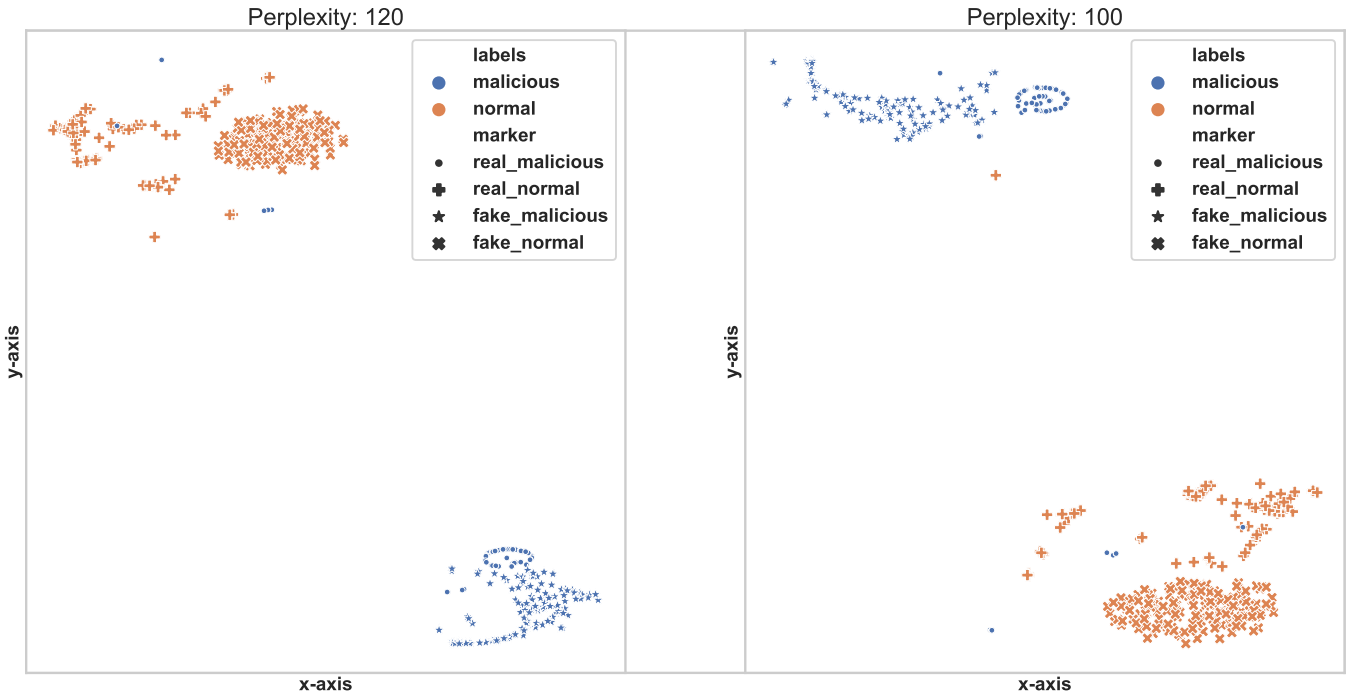

FIGURE 10: After Training (KDD99): Binary Class CGAN creates Distinct Clusters for the Malicious and Normal Profiles

TABLE 2: Comparison Report of Algorithms on the ALOI and KDD Dataset

Dataset	Algorithm	k (nearest neighbor)	Precision	F-1 Score	ROC AUC Score
ALOI	**KNN**	**1**	0.13019	0.17982	0.74141
	KNNW	**1**	0.14090	0.18164	0.74862
	LOF	**9**	0.11933	0.19791	0.78437
	SimplifiedLOF	**9**	0.13114	0.21236	0.79755
	LoOP	**12**	0.14852	0.23272	0.80243
	ODIN	**12**	0.15644	0.24430	0.80608
	KDEOS	**98**	0.09264	0.15412	0.77409
	LDF	**9**	0.08717	0.14766	0.74790
	INFLO	**9**	0.13931	0.22596	0.80020
	COF	**13**	0.14095	0.21083	0.80353
	AD-CGAN	N/A	**0.97909**	**0.97885**	**0.9832**
KDD	**KNN**	**85**	0.36786	0.48159	**0.98982**
	KNNW	**100**	0.11989	0.23577	0.98273
	LOF	**100**	0.01091	0.03139	0.82330
	SimplifiedLOF	**34**	0.00966	0.04012	0.62814
	LoOP	**34**	0.01445	0.06088	0.68726
	ODIN	**100**	0.01327	0.04944	0.78924
	KDEOS	**50**	0.01130	0.05336	0.61999
	LDF	**87**	0.02674	0.06834	0.88959
	INFLO	**25**	0.01095	0.05044	0.64848
	COF	**35**	0.01278	0.06275	0.61605
	AD-CGAN	N/A	**0.88796**	**0.85372**	0.89300

still generalize well on the rest of the test data. Traditional machine learning algorithms train on a larger portion of the data and usually reserve between $\mathbf{10}\% - \mathbf{20}\%$ of the data for evaluation and testing. However, in the *AD-CGAN* model, we show that we can relax this convention and still achieve great model performance. The model is able to generalize even when trained on a small subset of the data because CGANs have inherent ability to draw knowledge from the *latent* space distribution of the data used for training.

After we binarized the data, the *KDD99* dataset has **3925650** samples of the malicious profile and **972781** samples of the normal profile with **41** multivariate features. Instead of training on the whole dataset, we train on a balanced subset of the data comprising of **772781** samples from the normal profile and **772781** from the malicious profile. This training sample represents $\mathbf{31.55}\%$ of the whole dataset, and this percentage contrasts with the normal convention of training on larger subsets of the data. We sample the test data from the remaining subsets of the data.

Since our model depends on the perplexity parameter of the embedding layer, we used hyperparamter tuning to determine the optimum perplexity. For our evaluations on

KDD99 dataset, we use $\mathbf{120}$ as the perplexity parameter, and we sample a batch of $\mathbf{300}$ test samples at a time. Therefore, in Fig. 8, the *real* and *fake* samples of both the normal and malicious profiles form several indistinguishable clusters before we train the model. This figure acts as a baseline for the trained model. Furthermore, in Fig. 10, the *real* and *fake* samples of the normal profile class form a single, distinct cluster while the *real* and *fake* samples of the malicious profile class form a separate, distinct cluster after training the generator. This contrast in the behavior of the generator after training confirms our hypothesis that the CGANs can be used to learn the context of the data, and form the basis for anomaly detection.

V. DISCUSSION OF RESULTS

In Table 1, we show the metrics of the AD-CGAN when tested on the ALOI and the KDD99 dataset. In the ALOI dataset, we sample randomly from both the *augmented data* from the single class CGAN and the original anomalous data. The number of data samples tested is included in Table 1 and we can see that with $\mathbf{20000}$ data samples tested, the accuracy of the model and other metrics show effective detection of malicious samples with almost zero false positives. We attribute the ability of the AD-CGAN to achieve this level of performance on the false positive metric on the inherent nature AD-CGAN to use the latent knowledge of the distribution of the data to reduce the effect of noise during testing. Since the *positive class* is our malicious profile data, the data of Table 1 confirms the effectiveness of AD-CGAN on the ALOI dataset as it reports high precision and recall on both classes of data. Similarly, in Table 1, we report the class-based metrics for the KDD99 dataset as tested on $\mathbf{150000}$ samples of the test data. As seen from Table 1, for $\mathbf{150000}$ samples of the test data, the precision score of the malicious profile achieves one of our objectives of reducing false positives while ensuring that we detect the malicious profiles in the system. This high precision score ensures that while we detect the actual malicious profiles, we do not invest resources to investigate false alarms. And since the positive class is the malicious profile, this high precision at the cost of lower recall is preferable.

Also, in Table 2, we compare the performance of AD-CGAN on the ALOI dataset with other cluster-based algorithms that have been used on the data. We report only the *normalized, duplicate* and the *normalized, duplicate, 1-of-n encoding* versions of the ALOI and KDD99 dataset experiments respectively from [34] because we took similar preprocessing steps. We select the *average precision, maximum F-1 score*, and the *ROC AUC* score for comparison because those are the common metrics reported in [34]. From the results in Table 2 under the ALOI dataset category, AD-CGAN outperforms all the other algorithms in all the metrics reported. Although [34] trained and tested on a very small subset of the KDD99 dataset of about $\mathbf{60839}$ while we train and test on the full dataset of about $\mathbf{4,898,431}$ samples, we compare the performance of AD-CGAN because while some metrics like ROC AUC may vary with sample size and class imbalance, some others like *accuracy, precision, recall* is expected to be stable irrespective of the sample size for a well designed algorithm. Therefore, in Table 2, we compare the results of other clustering-based algorithms as reported in [34] to AD-CGAN. While the KNN algorithm has a better ROC AUC score, AD-CGAN outperforms all other algorithms in the average precision and F-1 scores. Since the ROC AUC are a function of sample size and class balance, we posit that the high ROC AUC score of KNN on KDD99 dataset is because of the small sample of data that it was trained and tested on.

In Fig. 9, we plot the ROC curves for the ALOI and KDD99 datasets with the malicious profiles designated as the positive class. The curves show that AD-CGAN has high precision for the malicious profiles of both data but the confidence in the KDD99 dataset is lower than in that of the ALOI dataset. This behavior is expected because the KDD99 test dataset has some anomalous types that are not present in the training sample. Therefore, even though AD-CGAN is able to detect these profiles with high precision, the true positive rate will not be the same when compared with dataset that has similar profile types in both the test and training samples.

VI. CONCLUSION AND FUTURE WORKS

In this research work, we introduce AD-CGAN, a CGANs-based anomaly detector for both balanced and imbalanced data. With the multi-stage transfer learning knowledge from the CGANs, we are able to eliminate the inherent bias introduced by class imbalance in data. We compare AD-CGAN performance with other cluster-based algorithms and the report show that AD-CGAN returns superior scores in almost all fronts. One of our objectives is to design an anomaly detector that reduces false positives while improving malicious profile detection accuracy, and from the metrics reported in this work, AD-CGAN achieves this with high precision that ensures that false positive is almost zero. Also, even though our sampling frequency is $\mathbf{500}$ and $\mathbf{300}$ samples respectively for the ALOI and KDD99 datasets respectively, AD-CGAN can operate in as little as $> \mathbf{20}$ samples during testing. This provides high flexibility for users to tune the algorithm according to their desired frequency of operation depending on the kind of application being monitored.

One limitation of the AD-CGAN algorithm we discovered is that the single class CGAN can not train on data in which the minority class data is only a handful of samples ($< \mathbf{900}$), therefore, in our future research endeavors, we will be looking to improve AD-CGAN to be able to handle this set of data.

REFERENCES

[1] H. Wang, M. J. Bah, and M. Hammad, "Progress in outlier detection techniques: A survey," IEEE Access, vol. 7, pp. 107 964–108 000, 2019.
[2] Y. luo, Y. Xiao, L. Cheng, G. Peng, and D. Yao, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," ACM Computing Survey, p. 29 pages, March 2020.

[3] D. M. Hawkins, Identification of outliers. Springer, 1980, vol. 11.
[4] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 6, no. 1, pp. 1–39, 2012.
[5] A. C. Bahnsen, Jun 2017. [Online]. Available: https://blog.easysol.net/building-ai-applications/
[6] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," arXiv preprint arXiv:1901.03407, 2019.
[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in neural information processing systems, 2014, pp. 2672–2680.
[8] Z. Wang, Q. She, and T. E. Ward, "Generative adversarial networks: A survey and taxonomy," arXiv preprint arXiv:1906.01529, 2019.
[9] M. Mirza and S. Osindero, "Conditional generative adversarial nets," arXiv preprint arXiv:1411.1784, 2014.
[10] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," Neurocomputing, vol. 50, pp. 159–175, 2003.
[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM computing surveys (CSUR), vol. 41, no. 3, p. 15, 2009.
[12] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, "Anomalous system call detection," ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 1, pp. 61–93, 2006.
[13] M. Ezeme, A. Azim, and Q. H. Mahmoud, "An imputation-based augmented anomaly detection from large traces of operating system events," in Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies. ACM, 2017, pp. 43–52.
[14] O. M. Ezeme, Q. H. Mahmoud, and A. Azim, "Peskea: Anomaly detection framework for profiling kernel event attributes in embedded systems," IEEE Transactions on Emerging Topics in Computing, p. accepted, 2020.
[15] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017, pp. 1285–1298.
[16] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," Proceedings of SOSP'09, 2009.
[17] M. O. Ezeme, Q. Mahmoud, and A. Azim, "Hierarchical attention-based anomaly detection model for embedded operating systems," in 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2018.
[18] O. M. Ezeme, Q. H. Mahmoud, and A. Azim, "Dream: deep recursive attentive model for anomaly detection in kernel events," IEEE Access, vol. 7, pp. 18 860–18 870, 2019.
[19] M.-K. Yoon, S. Mohan, J. Choi, M. Christodorescu, and L. Sha, "Learning execution contexts from system call distribution for anomaly detection in smart embedded system," in Proceedings of the Second International Conference on Internet-of-Things Design and Implementation. ACM, 2017, pp. 191–196.
[20] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement. USENIX Association, 2005, pp. 32–32.
[21] M. Salem, M. Crowley, and S. Fischmeister, "Anomaly detection using inter-arrival curves for real-time systems," in Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on. IEEE, 2016, pp. 97–106.
[22] F. Li, Z. Li, W. Huo, and X. Feng, "Locating software faults based on minimum debugging frontier set," IEEE Transactions on Software Engineering, vol. 43, no. 8, pp. 760–776, 2017.
[23] A. P. Kosoresow and S. Hofmeyer, "Intrusion detection via system call traces," IEEE software, vol. 14, no. 5, pp. 35–42, 1997.
[24] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344). IEEE, 1999, pp. 133–145.
[25] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," Journal of computer security, vol. 6, no. 3, pp. 151–180, 1998.
[26] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks," in International Conference on Artificial Neural Networks. Springer, 2019, pp. 703–716.
[27] O. M. Ezeme, M. Lescisin, Q. H. Mahmoud, and A. Azim, "Deepanom: An ensemble deep framework for anomaly detection in system processes," in Canadian Conference on Artificial Intelligence. Springer, 2019, pp. 549–555.
[28] O. M. Ezeme, Q. Mahmoud, and A. Azim, "A framework for anomaly detection in time-driven and event-driven processes using kernel traces," IEEE Transactions on Knowledge and Data Engineering, 2020.
[29] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in Advances in neural information processing systems, 2016, pp. 2172–2180.
[30] G. E. Hinton and S. T. Roweis, "Stochastic neighbor embedding," in Advances in neural information processing systems, 2003, pp. 857–864.
[31] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," Journal of computational and graphical statistics, vol. 15, no. 2, pp. 265–286, 2006.
[32] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," PloS one, vol. 11, no. 4, p. e0152173, 2016.
[33] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-sne effectively," Distill, 2016. [Online]. Available: http://distill.pub/2016/misread-tsne
[34] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study," Data Mining and Knowledge Discovery, vol. 30, no. 4, pp. 891–927, 2016.

OKWUDILI M. EZEME is a Ph.D. candidate in the Department of Electrical, Computer and Software Engineering, Ontario Tech University, Oshawa, Ontario, Canada. He received the M.Sc. degree from the Department of Electrical and Computer Engineering, University of Toronto (UofT), Ontario, Canada in 2015, and the B.Engr. degree in Electronics Engineering from the University of Nigeria, Nsukka (UNN), Nigeria in 2010. His current research interests include deep recurrent neural networks, time-series data processing, generative adversarial networks and distributed processing in Internet of Things (IoT) devices.

QUSAY H. MAHMOUD is a Professor of Software Engineering in the Department of Electrical, Computer, and Software Engineering at Ontario Tech University in Oshawa, Canada. He was the Founding Chair of the Department, and more recently he has served as Associate Dean of the Faculty of Engineering and Applied Science at the same university. His research interests include intelligent software systems and cybersecurity.

DR. AKRAMUL AZIM is an assistant professor in the Department of Electrical, Computer and Software Engineering and head of the real-time embedded software (RTEMSOFT) research group at Ontario Tech University, Oshawa, Ontario, Canada. His research interests include real-time systems, embedded software, software verification and validation, safety-critical software and intelligent transportation systems. He is a Professional Engineer in Ontario and a senior member of IEEE.