Received July 21, 2020, accepted September 14, 2020, date of publication September 21, 2020, date of current version October 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025530

Design and Development of AD-CGAN: Conditional Generative Adversarial Networks for Anomaly Detection

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

Corresponding author: Okwudili M. Ezeme (mellitus.ezeme@ontariotechu.net)

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

ABSTRACT Whether in the realm of software or hardware, datasets representing the state of systems are mostly imbalanced. This imbalance is because these systems' reliability requirements 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 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 GAN-derived and non-GAN-derived state of the art algorithms on benchmark datasets. AD-CGAN outperforms most of the algorithms in the standard metrics of *Precision, Recall*, and *F-1 Score*. Where AD-CGAN does not perform better in the parameters used, it has the advantage of being lightweight. Therefore, it can be deployed for both online and offline anomaly detection tasks since it does not use an input sample inversion strategy.

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 that suggests 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 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 trying 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

The associate editor coordinating the review of this manuscript and approving it for publication was Ilaria Boscolo Galazzo.

advantages over the traditional algorithms [5], [6]: **a**) they scale better when the dataset is significant, 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 do 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 widespread because the evolving nature of the boundary between normal and deviant behaviors in several domains makes it difficult to demarcate with certainty 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 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 asset and the company's financial outlay. Recently, GANs [7] has emerged as an essential area of deep learning with groundbreaking 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 can 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 contain normal operating conditions and 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 return 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) introducing 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 tests 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 identify a misuse behavior by storing signatures of discovered anomalies, anomaly detection models construct contexts using known standard features and tag any deviation from the created profile as anomalous. The apparent 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 the anomaly detection approach, we highlight mostly the practices 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, they do not capture the nature of the system process behavior that 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 authors augment the LSTM model with a workflow model that helps to identify context switches. Still, 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 use 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. Still, 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 that 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 and 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. Still, this type of anomaly model is best suited for cases where the volume of logs determines 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], the authors design a realtime systems anomaly detection to detect anomalous traces in a log sequence using the principle of inter-arrival curves. 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 detecting 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 a *locality frame*, we can detect these anomalies. And this *local profile* assumption is the limitation of the work as sophistication in cyberattacks has shown that we can exploit both local and non-local profiles in the design of anomalies. However, authors of [23] admitted that when the abnormal sequences are not concentrated in a burst, their algorithm cannot handle such scenarios. Reference [24] used techniques such as *counting observed system calls, frequency distribution approaches, a rule-based method 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 build a tree of the system call sequences' possible routes based on the observed behavior during the learning phase. Since [24], [25] only consider the temporal *ordering* of the traces, they ignore all other parameters like the timing information and system call arguments. Therefore, their methods cannot handle a previously unseen system call without reconstructing the whole model. In [26], [27], the authors implemented an ensemble framework 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 principles as our work but differs in architecture. Furthermore, [26], [27] algorithms' predictions could suffer from cumulative errors as a result of the use of anomalous sub-sequence in the input samples. The authors of [28] propose a deep transfer network (DTN) that uses transfer learning to leverage the knowledge gained from rich labeled data in the source domain to facilitate diagnosing a new but similar target task. The DTN method primarily tries to solve the challenge of data drift between the training samples and test samples. By extending the marginal distribution adaptation (MDA) to joint distribution adaptation (JDA), DTN can exploit the discrimination structures associated with the well-defined data in the source domain to adapt the conditional distribution of unlabeled target data, and thus guarantee a more accurate distribution matching.

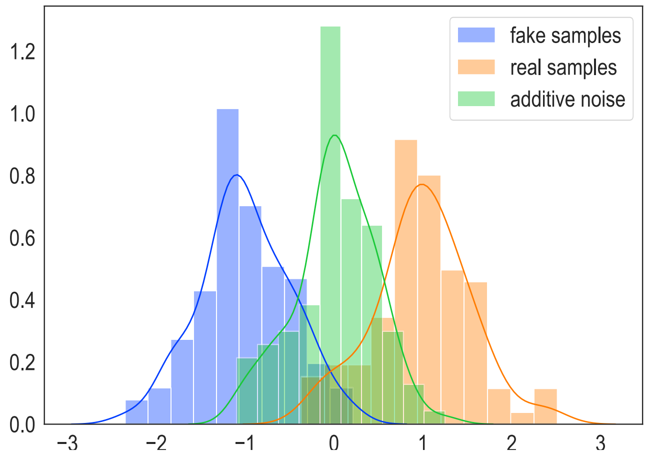


FIGURE 1. Visualizing the effect of synthetic noise addition to the discriminator inputs.

In [29], 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 significantly 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 the *discriminator* for both training and inference while we require only the *generator* during inference. Also, instead of using residual errors of the GANs for anomaly decision as done in [29], we came up with a cluster-based anomaly detector. Another key difference is the requirement to invert the input samples to the latent space during inference, making the computation expensive. The authors of [30] propose a deep convolutional GAN named *AnoGAN* that learns the normal anatomical variability based on mapping from image space to latent space and uses this knowledge to discriminate between benign and anomalous medical image samples. AnoGAN is designed for unsupervised learning, and as such, it trains only on the normal profile sample during the training phase. Therefore, even though it is designed for unsupervised learning, it still relies

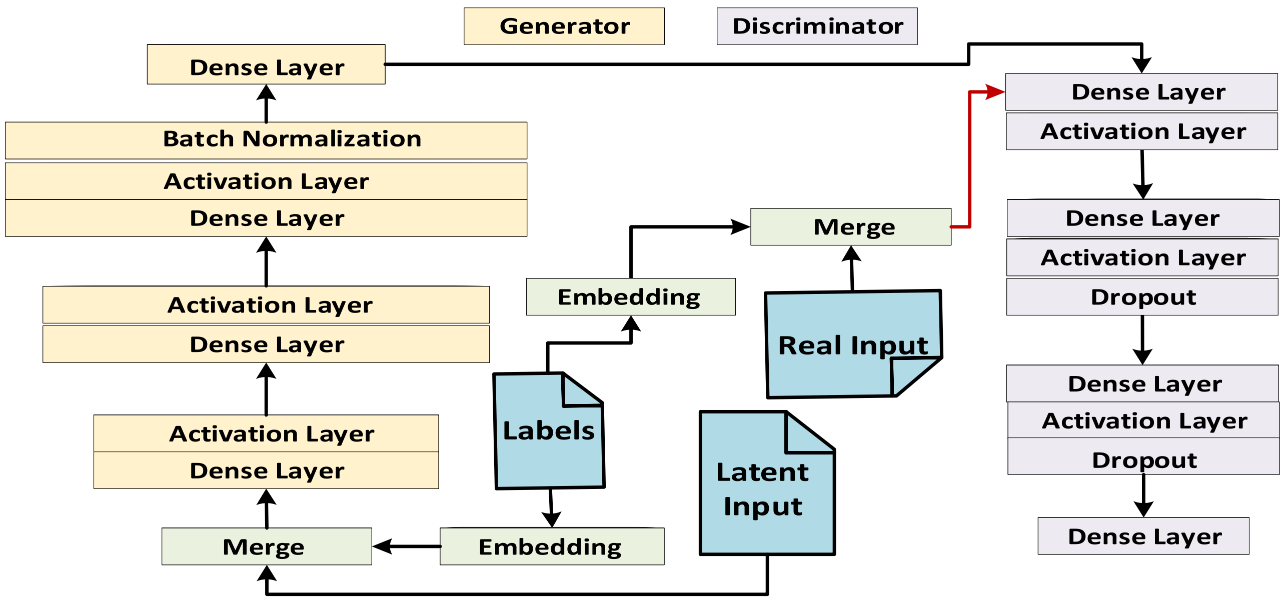


FIGURE 2. Architecture of the conditional generative adversarial networks (CGAN) of AD-CGAN.

on human intervention to accurately identify the standard samples used for training, which can be a source of error. Further on GAN-based approaches, [31] proposes an anomaly detection framework with GAN called *EGAN* which uses an extra module called an encoder to learn the input to latent space mapping during training to avoid the computationally intensive process of input sample inversion during inference. While the use of an encoder is smart, this framework is still complicated and makes it difficult for online application since all the modules are used during inference. Finally, the authors of [32] employ transfer learning to train a deep convolutional neural network called *DACNN* for diagnosing mechanical faults. The generative adversarial network learning is used as a regularization scheme during training to enable DACNN to overcome the limitation of insufficient labeled data that plague many supervised algorithms. Thereby resulting in improved generalization performance on the test data during inference.

III. AD-CGAN: DESIGN AND IMPLEMENTATION

In this section, we discuss the design of AD-CGAN, starting with examining the qualities that a good anomaly detector should possess before delving 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, the efficiency of an algorithm's performance has to take into account the distribution of the different classes we *predict* or *classify* to ensure that the algorithm's performance is better than a random guess. For example, a dataset with a *normal* class of **95**% and an *anomalous* class of **5**% already has a **95**% accuracy using a human guess. Since the aim of anomaly detection frameworks is to detect that critical **5**% of anomalous samples in this example, this **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 a false positive classification is also huge; therefore, an effective anomaly framework should have the ability to detect the unusual samples while reducing the incidents of false positives. Having 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

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

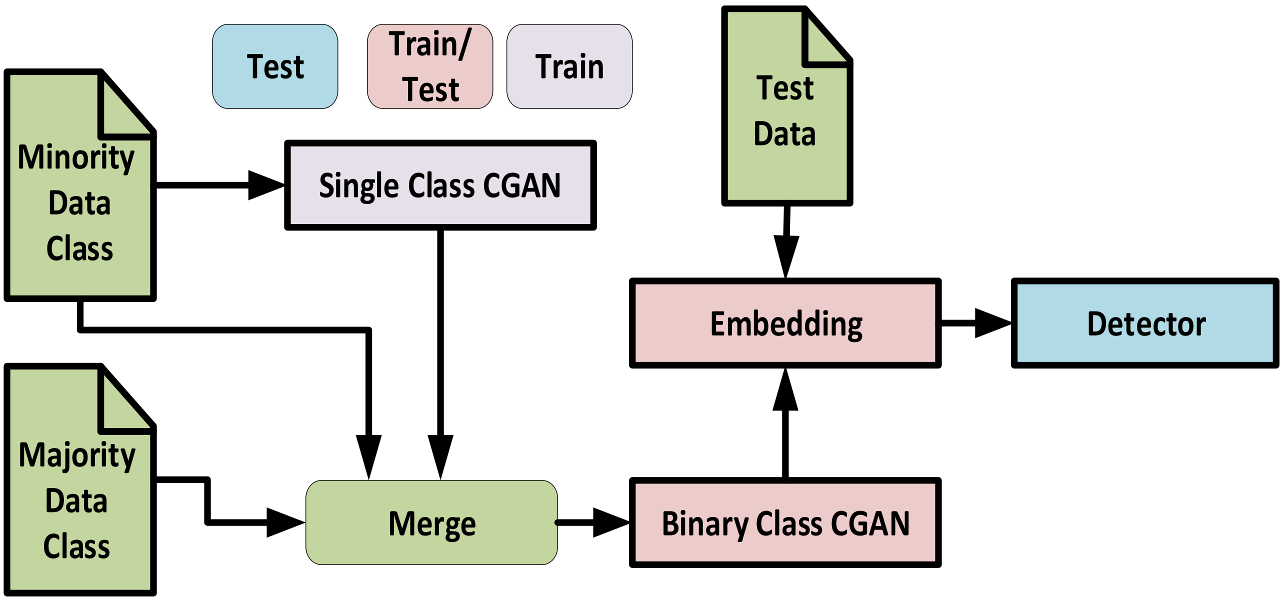


FIGURE 3. AD-CGAN framework for anomaly detection in balanced and imbalanced data.

A. CGAN

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. We sandwich the dense layers with activation layers. The last layer before the output layer has a *batch normalization* layer as a *regularization* scheme that makes the network robust and improves the network's *generalization* properties. Given a latent input, z, a conditional input, $\boldsymbol{c}$, (1) is the generator equation where $\boldsymbol{G}$ 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.

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

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 white noise as described in Section III-A.3 before it is fed to the discriminator.

2) DISCRIMINATOR

In [33], 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, $\boldsymbol{JS}[\vec{y}_g|\vec{x}_d]$ 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.

$$\boldsymbol{D} : (\vec{y}, \vec{x}) \longmapsto \vec{y}_d \quad \text{where } \vec{y} \supseteq \vec{y}_g, \vec{x}_d \tag{2}$$

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

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

3) CONTROLLING DISCRIMINATOR OVERFITTING

In Fig. 1, we show the effect of adding synthetic noise to both the generator output, $\vec{y}_g$ and the real samples input, $\vec{x}_d$. 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 $\boldsymbol{G} : (\vec{z}, \vec{c}) \longmapsto \vec{y}_g$ where the support is $\{z \in \vec{z}, \boldsymbol{c} \in \vec{c} : \boldsymbol{G}(z, \boldsymbol{c}) \neq \boldsymbol{0}\}$, the intersection between the support of the generator and the support of the distribution that produced $\vec{x}_d$ 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, $\vec{z}, \vec{c}$ so that the generator output, $\vec{y}_g$ and the real samples of the class of interest, $\vec{x}_d$ when clustered, show evidence that $\vec{x}_d$ and $\vec{y}_g$ are drawn from the same underlying distribution. We measure this similarity with the use of the *stochastic neighbor embedding* algorithm [34]. As will be shown in Section IV, when the single class CGAN is sufficiently trained, the generator output, $\vec{y}_g$ and the real samples, $\vec{x}_d$ form a single indistinguishable cluster.

2) BINARY CLASS CGAN

The *binary class CGAN* trains on the *standard* and *abnormal* samples from the original and augmented samples from the *single class CGAN*. Unlike the single class CGAN, the binary class CGAN aims to train the generator, $\boldsymbol{G_b}$ with the multivariate binary class data, $\vec{z}, \vec{c}$ such that the anomalous samples from the generator output, $\vec{y}_g$, and the anomalous samples of the real samples, $\vec{x}_d$ 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 *fake normal* and *fake malicious* profiles, and this output, $\vec{y}_g$ is a multivariate data of the same dimension as the real samples, $\vec{x}_d$. Therefore, the number of samples generated by the binary class CGAN, $\vec{y}_g$ is the same as the number of the real samples, $\vec{x}_d$. 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 can learn the profiles of the data being studied; **b**) during testing, the *fake labels* used in the CGANs to generate $\vec{y}_g$ 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, $\vec{x}_d$ is fixed, AD-CGAN can handle data of arbitrary dimension by *preprocessing* the arbitrary dimensional data to the dimension of the real samples, $\vec{x}_d$ using the sparse principal component analysis algorithm [35]. 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 $\{\vec{y}_g, \vec{x}_d\} \in \vec{y}$, (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.

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

To ensure symmetry, we compute the *joint* probability, $\boldsymbol{P_{ij}}$ of the similarity between points $\boldsymbol{i}$ and $\boldsymbol{j}$ using (5). Since $\boldsymbol{P_{j|j} = 0}$, $\boldsymbol{P_{jj} = 0}$ 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.

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

4) DETECTOR

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

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

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.

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

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.

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

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).

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

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).

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

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

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

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.

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

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.

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)* [36] anomaly dataset. While the *KDD99* has a 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 consists 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 not easy to train a model with this dataset without overfitting on the majority class. And since the 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.

1) ALOI SINGLE CLASS CGAN RESULTS

As highlighted in Section III-B.1, the *single class CGAN* aims 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 can understand the underlying distribution of the real samples. To measure the generator's performance, we take snapshots of the model at intervals of **50** epochs during training. After that, we compare the generated quality at each snapshot, and the best model becomes our trained generator model. According to [37], the *tSNE* algorithm used for the embedding of Section III-B.3 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 generator can 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 that are indistinguishable from the real samples confirms one of our hypotheses 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 to produce 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 so 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-B.4. As seen in Fig. 6, the embedding output before training shows a mixture of the positive and negative classes in the same cluster, thereby creating both false positive and false negative situations on both the *real* and *fake* samples. In all our result analysis, we designate the malicious samples as the *positive class*. In Fig. 7, the trained model results show how the generator output and real test samples have successfully been separated into their respective categories. Although the snapshots of Fig. 7 does not show precisely 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

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

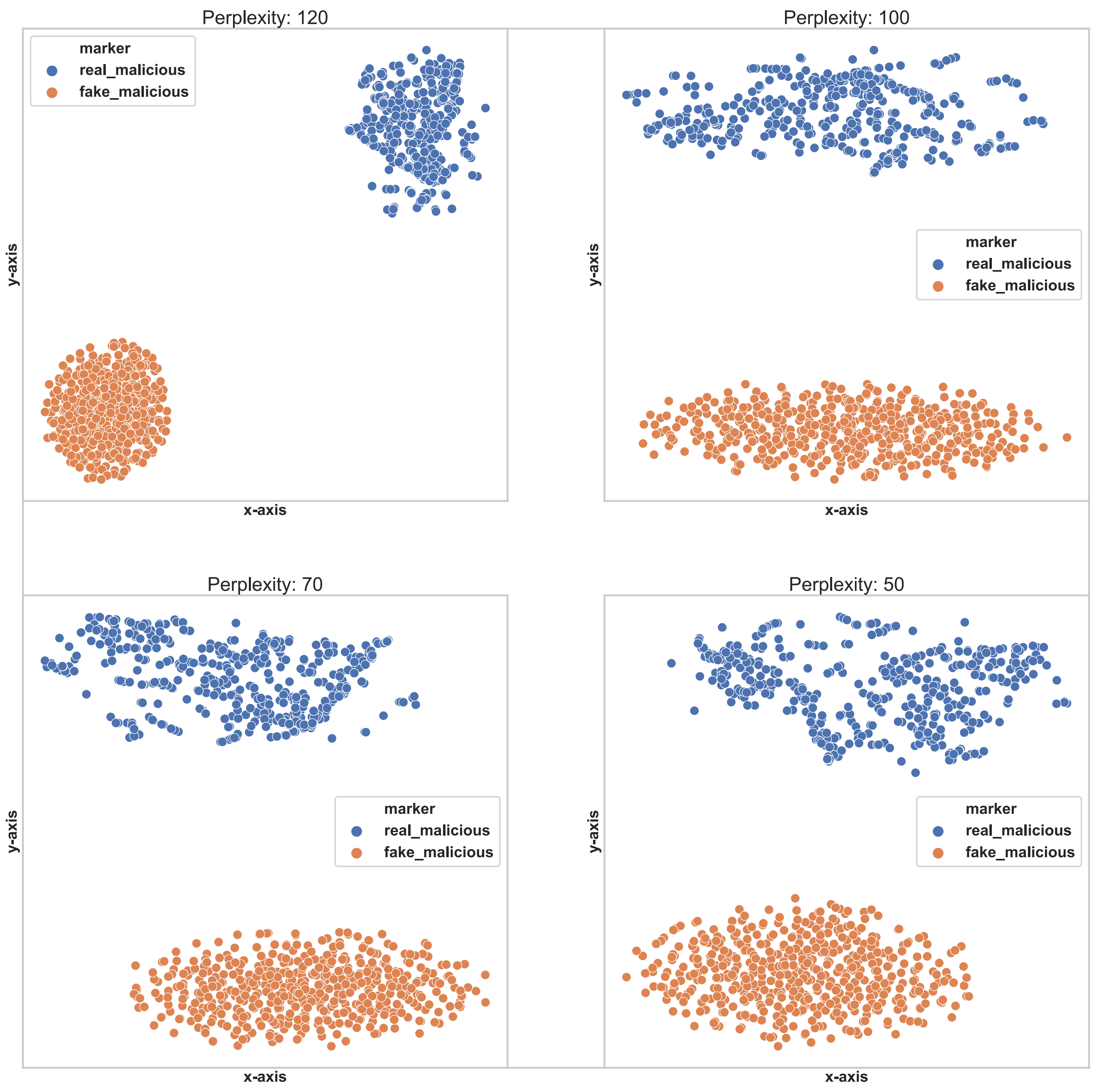


FIGURE 4. Before training (ALOI): real and fake samples of the single class CGAN form 2 clusters.

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	

that the generator can be used to understand the different latent class distributions. 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 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

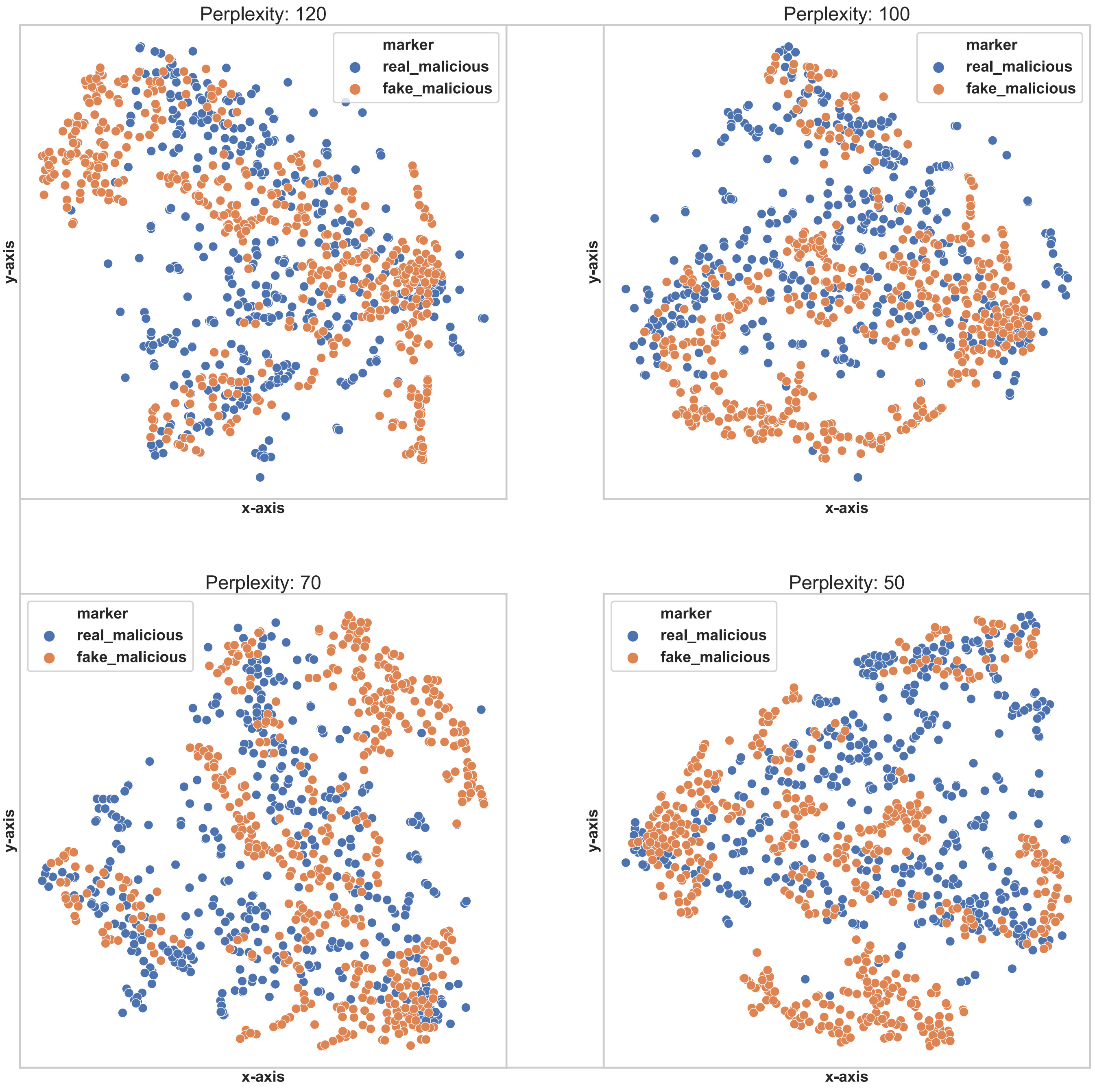


FIGURE 5. **After training (ALOI): real and fake samples of the single class CGAN form 1 cluster.**

for each class. The malicious class test sample consists of the *real* and *fake* samples from the single class CGAN. The 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 model's performance 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 [38] 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 [38] 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 [38]. Also, [38] evaluated 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, broadly classified into

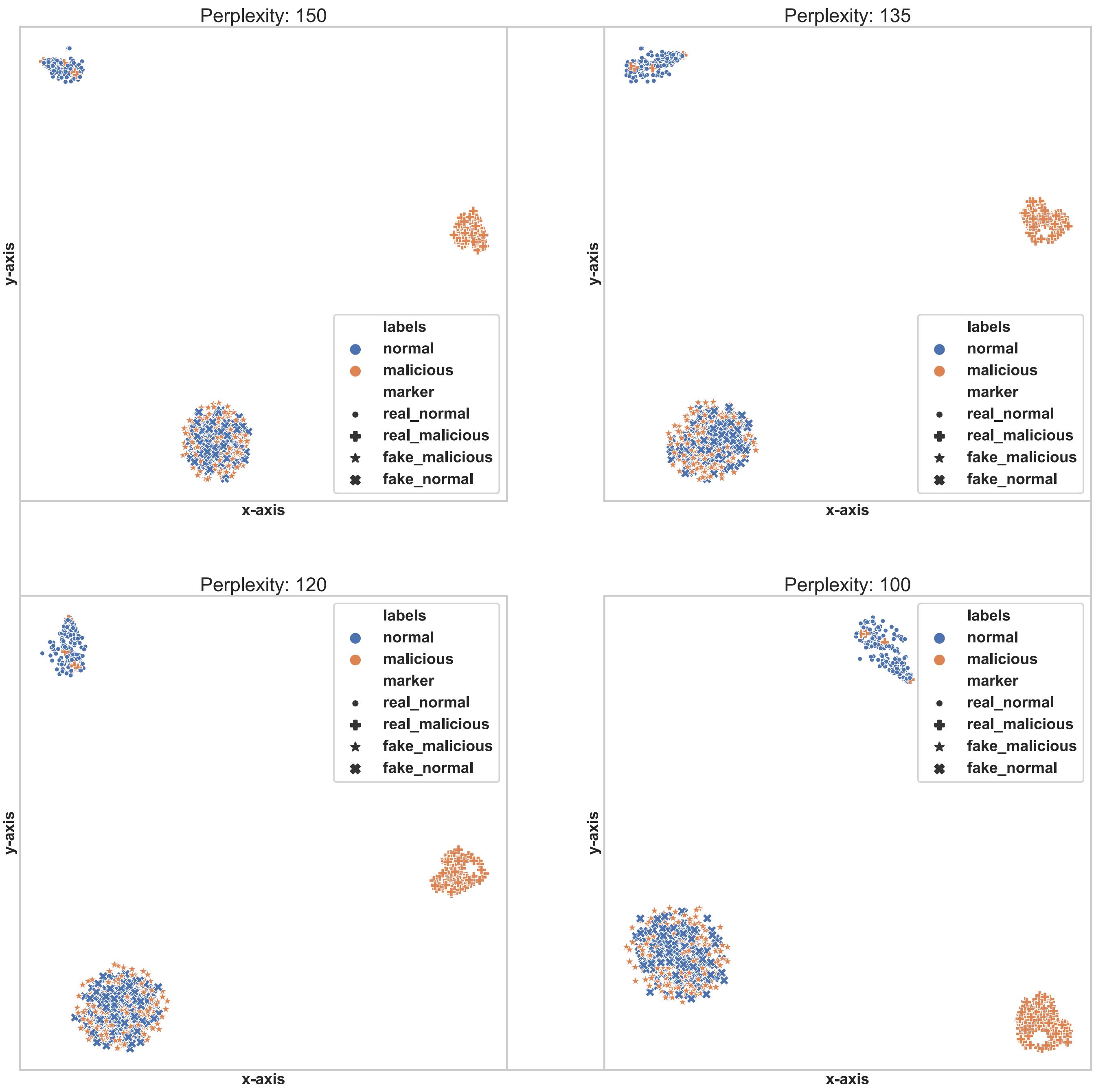


FIGURE 6. Before training (ALOI): binary class CGAN generates mixed clusters for malicious and normal profiles.

malicious and *benign* 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 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 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 **10%-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 excellent model performance. The model can generalize even when trained on a small subset of the data because CGANs have the 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

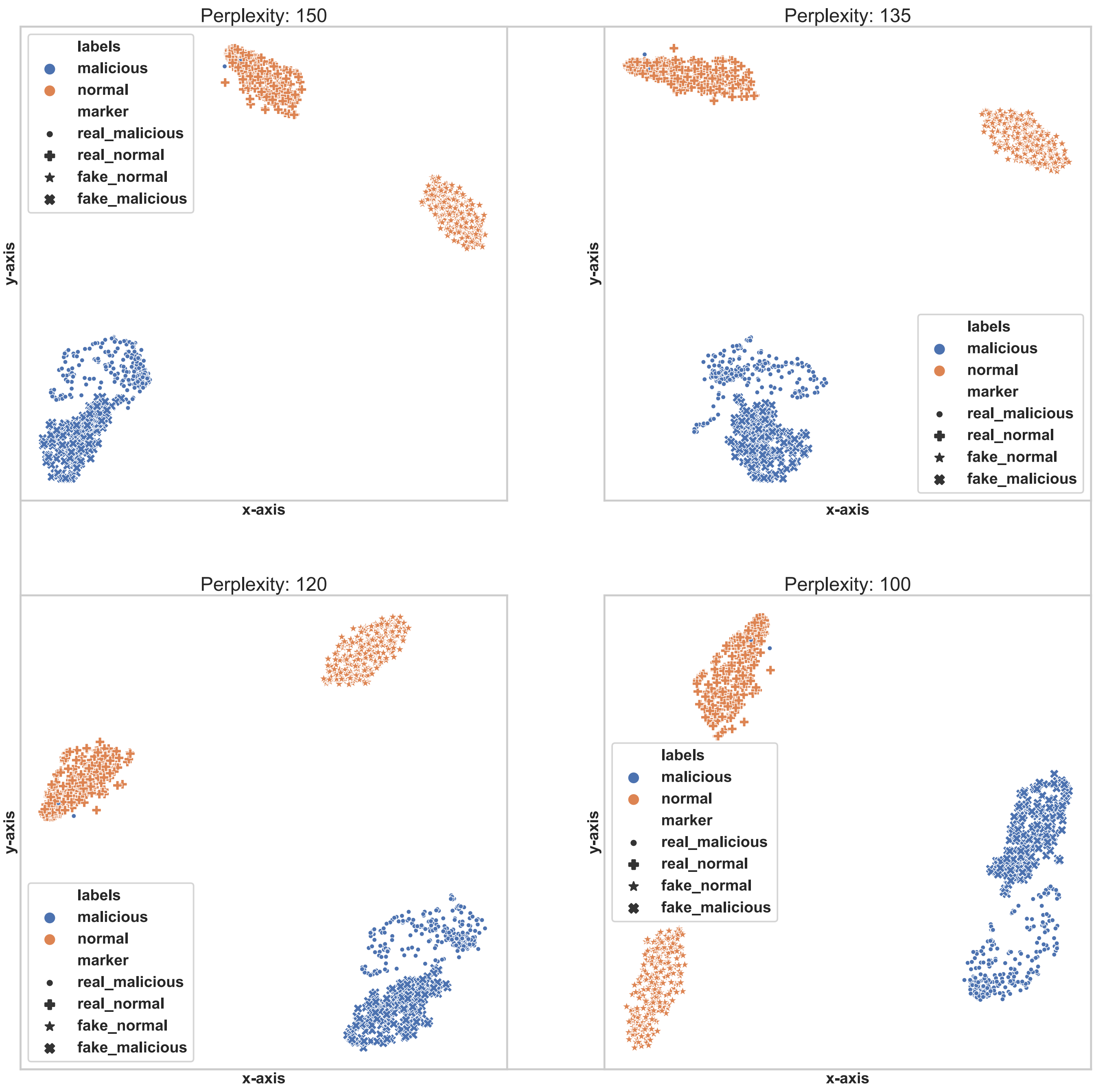


FIGURE 7. **After training (ALOI): binary class CGAN generates distinct clusters for the malicious and normal profiles.**

balanced subset of the data comprising **772781** samples from the normal profile and **772781** from the malicious profile. This training sample represents **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 the KDD99 dataset, we use **120** as the perplexity parameter, and we sample a batch of **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.

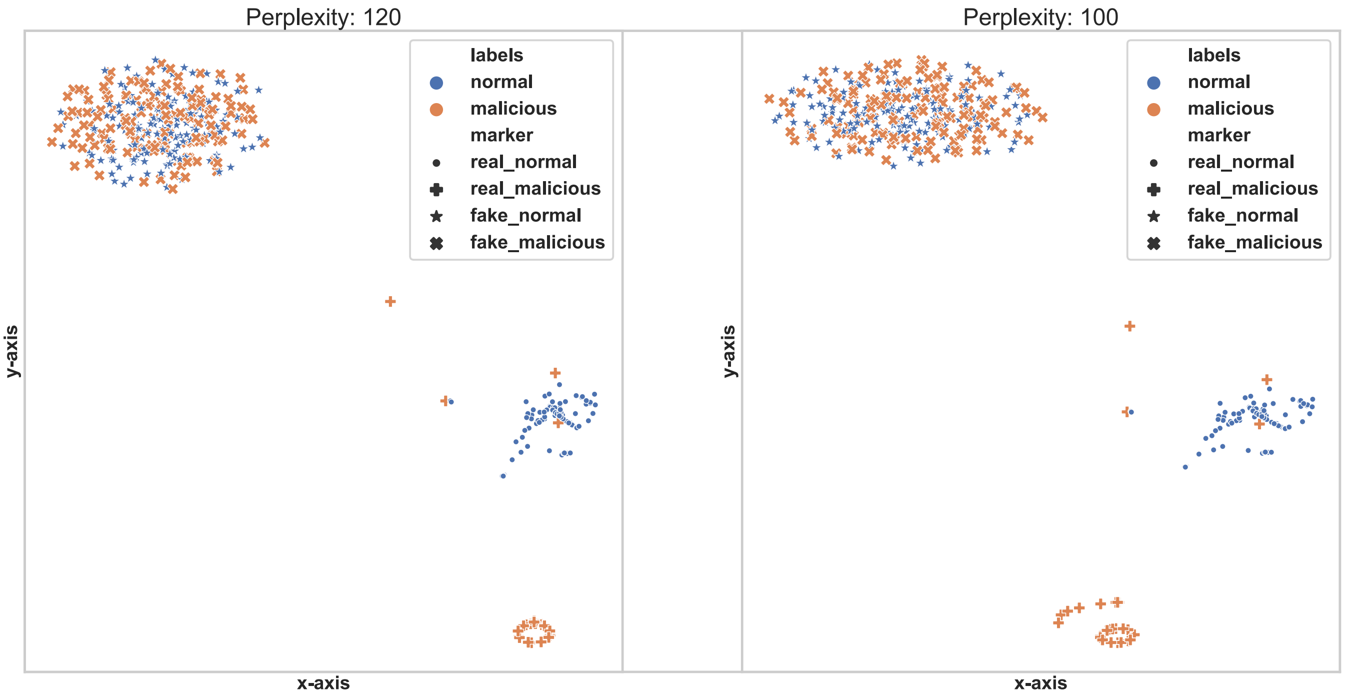


FIGURE 8. **Before training (KDD99): binary class CGAN creates mixed clusters for the malicious and normal profiles.**

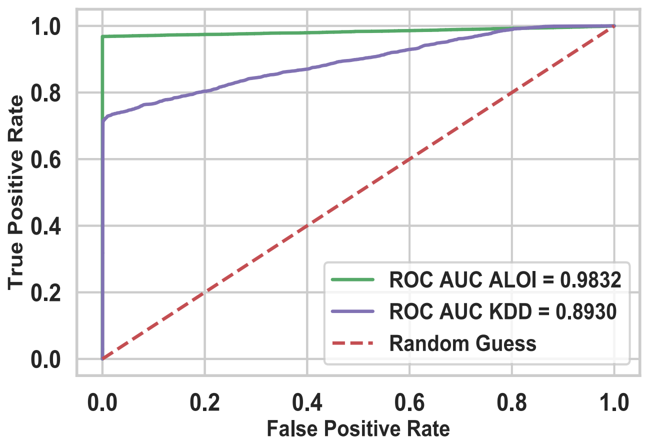


FIGURE 9. **ROC curve for the ALOI and KDD99 dataset.**

V. DISCUSSION OF RESULTS

A. AD-CGAN PERFORMANCE ON ALOI AND KDD DATA

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 randomly sample from the *augmented data* from the single class CGAN and the original anomalous data. The number of data samples tested is included in Table 1. We see that with **20000** data samples tested, the accuracy of the model and other metrics show the 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 **150000** samples of the test data. As seen from Table 1, for **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.

B. AD-CGAN VERSUS NON-GAN-DERIVED METHODS

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. From [38], we report only the *normalized, duplicate,* and the *normalized, duplicate, 1-of-n encoding* versions of the ALOI and KDD99 dataset experiments respectively 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 [38]. From the results in Table 2 under the ALOI dataset category, AD-CGAN outperforms all the other algorithms in all the metrics reported. Although [38] trained and tested on a very small subset of the KDD99 dataset of about **60839** while we train and test on the full dataset of about **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* are expected to

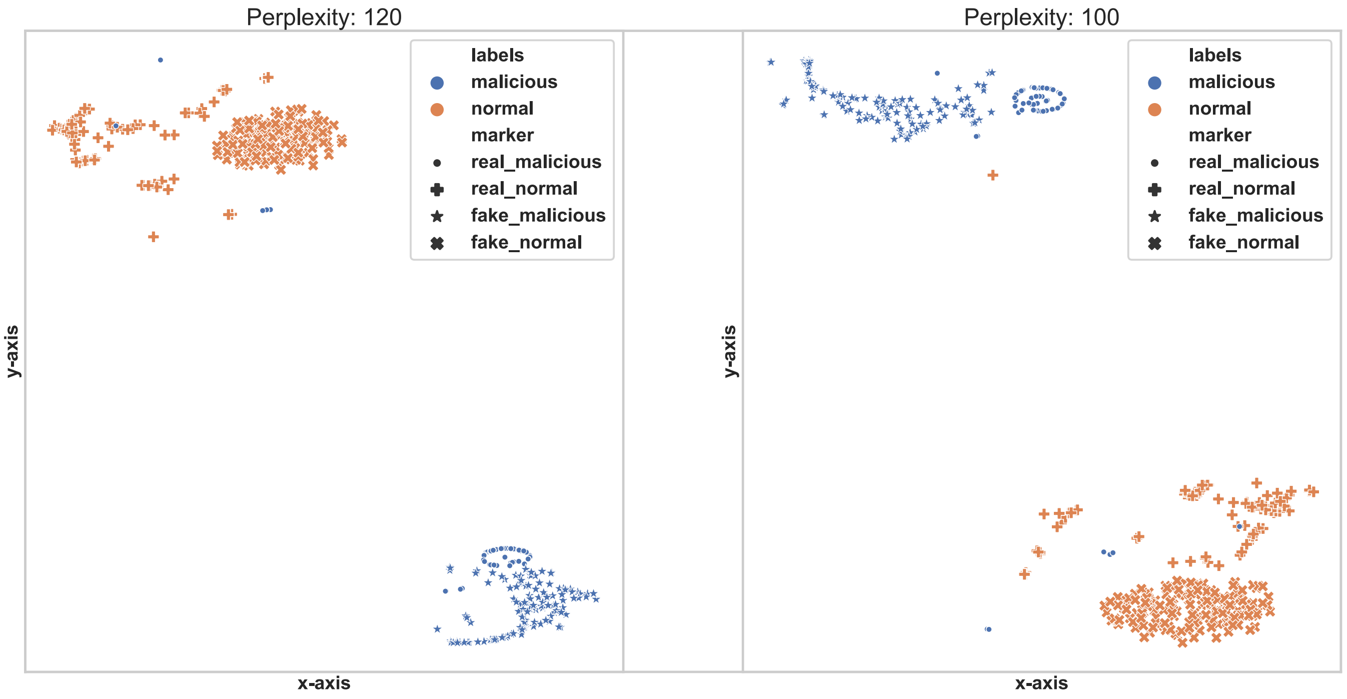


FIGURE 10. After training (KDD99): binary class CGAN creates distinct clusters for the malicious and normal profiles.

TABLE 2. Comparison report of non-GAN-based 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

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 [38] 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 a high precision for both data's malicious profiles, 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 can detect these profiles with high precision, the true positive rate

TABLE 3. Comparison report of AD-CGAN with recent GAN-based and non-GAN-based approaches on KDD99 dataset.

Dataset	Models	Precision	Recall	F-1 Score
KDD	**PCA**	60.66	37.69	0.47
	KNN	45.51	18.98	0.53
	FB	48.98	19.36	0.23
	AE	80.59	42.36	0.55
	DSEBM	86.19	64.46	0.74
	AnoGAN	87.86	82.97	0.85
	EGAN	92.00	95.82	**0.94**
	MAD-GAN*	**94.92**	19.14	0.32
	MAD-GAN**	81.58	**96.33**	0.88
	MAD-GAN***	86.91	94.79	0.90
	AD-CGAN	88.80	85.37	0.89

will not be the same when compared with the dataset with similar profile types in both the test and training samples.

C. AD-CGAN VERSUS GAN-DERIVED AND NON-GAN-DERIVED METHODS

In Table 3, we compare the performance of AD-CGAN to other methods, including those that implement variants of GAN-based anomaly detection frameworks. We report the PCA performance again in Table 3 since the report in Table 2 does not use the full dataset and is also older than that of the report in Table 3. *FB* in Table 3 refers to feature bagging, while *AE* refers to generic autoencoder where reconstruction error is used to generate anomaly scores. *DSEBM* of Table 3 refers to the Deep Structured Energy-Based Models for Anomaly Detection developed by the authors of [39]. These models serve as the baseline for the non-GAN-based approaches while using the *EGAN, AnoGAN*, and *MAD-GAN* for comparison in the GAN-based methods. The data for the algorithms and methods in Table 3 are taken from [29] and [31], which were published in **2019** and **2018** respectively.

In all the non-GAN-based models, AD-CGAN performs better than all of the models in all three key metrics of *Precision, Recall* and *F-1 Score*. For the GAN-derived models, AD-CGAN outperforms the AnoGAN in all the three metrics reported in Table 3. While EGAN performs better than AD-CGAN in this dataset, the use of an additional module called an *Encoder* in addition to the generator and discriminator during inference makes this model computationally complex when compared to AD-CGAN, which uses only the generator for inference and does not involve the computationally expensive process of input sample inversion to latent space during inference. We report results for three versions of MAD-GAN with each corresponding to the best result of the model in one of *Precision, Recall*, or *F-1 Score*. One obvious observation from the report is model inference instability as the results of the *Precision, Recall*, and *F-1 Score* vary in each run. We posit that this variability in inference output is caused by information loss during the computationally expensive process in input inversion during inference. Unlike these GAN-derived models that use either an *encoder* or perform input sample inversion to latent space during inference, AD-CGAN smartly avoids all these approaches by using the binary class GAN generator (G_b) to produce centroids in two-dimensional space for both the anomalous and benign samples during inference and quickly produces anomaly score for each sample projected into the same two-dimensional space as that of the outputs of G_b. Hence, while some metrics of MAD-GAN are better in some models, AD-CGAN generates inference outputs that are consistent and AD-CGAN does not rely on a heuristic.

VI. CONCLUSION AND FUTURE WORK

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 **500** and **300** samples for the ALOI and KDD99 datasets respectively, AD-CGAN can operate in as little as $>$**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 ($<$**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. 107964–108000, 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 Comput. Surv.*, p. 29, Mar. 2020.

[3] D. M. Hawkins, *Identification of Outliers*, vol. 11. Cham, Switzerland: Springer, 1980.

[4] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discovery Data*, vol. 6, no. 1, pp. 1–39, Mar. 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," 2019, *arXiv:1901.03407*. [Online]. Available: http://arxiv.org/abs/1901.03407

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[8] Z. Wang, Q. She, and T. E. Ward, "Generative adversarial networks in computer vision: A survey and taxonomy," 2019, *arXiv:1906.01529*. [Online]. Available: http://arxiv.org/abs/1906.01529

[9] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*. [Online]. Available: http://arxiv.org/abs/1411.1784

[10] G. P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, Jan. 2003.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[12] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, "Anomalous system call detection," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 1, pp. 61–93, Feb. 2006.

[13] M. Ezeme, A. Azim, and Q. H. Mahmoud, "An imputation-based augmented anomaly detection from large traces of operating system events," in *Proc. 4th IEEE/ACM Int. Conf. Big Data Comput., Appl. Technol. (BDCAT)*, 2017, pp. 43–52.

[14] O. M. Ezeme, A. Azim, and Q. Mahmoud, "PESKEA: Anomaly detection framework for profiling kernel event attributes in embedded systems," *IEEE Trans. Emerg. Topics Comput.*, early access, Feb. 3, 2020, doi: 10.1109/TETC.2020.2971251.

[15] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[16] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," in *Proc. SOSP*, 2009.

[17] M. O. Ezeme, Q. H. Mahmoud, and A. Azim, "Hierarchical attention-based anomaly detection model for embedded operating systems," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2018, pp. 225–231.

[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. 18860–18870, 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 *Proc. 2nd Int. Conf. Internet-of-Things Design Implement.*, Apr. 2017, pp. 191–196.

[20] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proc. 5th ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2005, p. 32.

[21] M. Salem, M. Crowley, and S. Fischmeister, "Anomaly detection using inter-arrival curves for real-time systems," in *Proc. 28th Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2016, pp. 97–106.

[22] F. Li, Z. Li, W. Huo, and X. Feng, "Locating software faults based on minimum debugging frontier set," *IEEE Trans. Softw. Eng.*, vol. 43, no. 8, pp. 760–776, Aug. 2017.

[23] A. P. Kosoresow and S. A. Hofmeyer, "Intrusion detection via system call traces," *IEEE Softw.*, vol. 14, no. 5, pp. 35–42, Sep. 1997.

[24] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Proc. IEEE Symp. Secur. Privacy*, May 1999, pp. 133–145.

[25] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Secur.*, vol. 6, no. 3, pp. 151–180, Jul. 1998.

[26] O. M. Ezeme, M. Lescisin, Q. H. Mahmoud, and A. Azim, "Deepanom: An ensemble deep framework for anomaly detection in system processes," in *Proc. Can. Conf. Artif. Intell.* Cham, Switzerland: Springer, 2019, pp. 549–555.

[27] O. M. Ezeme, Q. Mahmoud, and A. Azim, "A framework for anomaly detection in time-driven and event-driven processes using kernel traces," *IEEE Trans. Knowl. Data Eng.*, early access, Mar. 5, 2020, doi: 10.1109/TKDE.2020.2978469.

[28] T. Han, C. Liu, W. Yang, and D. Jiang, "Deep transfer network with joint distribution adaptation: A new intelligent fault diagnosis framework for industry application," *ISA Trans.*, vol. 97, pp. 269–281, Feb. 2020.

[29] 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 *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2019, pp. 703–716.

[30] T. Schlegl, P. Seeböck, S. M. Waldstein, S.-E. Ursula, and L. Georg, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Proc. Int. Conf. Inf. Process. Med. Imag.* Cham, Switzerland: Springer, 2017, pp. 146–157.

[31] H. Zenati, C. Sheng Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-based anomaly detection," 2018, *arXiv:1802.06222*. [Online]. Available: http://arxiv.org/abs/1802.06222

[32] T. Han, C. Liu, W. Yang, and D. Jiang, "A novel adversarial learning framework in deep convolutional neural network for intelligent diagnosis of mechanical faults," *Knowl.-Based Syst.*, vol. 165, pp. 474–487, Feb. 2019.

[33] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2172–2180.

[34] G. E. Hinton and S. T. Roweis, "Stochastic neighbor embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2003, pp. 857–864.

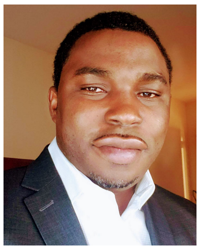
[35] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *J. Comput. Graph. Statist.*, vol. 15, no. 2, pp. 265–286, Jun. 2006.

[36] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLoS ONE*, vol. 11, no. 4, Apr. 2016, Art. no. e0152173.

[37] M. Wattenberg, F. Viégas, and I. Johnson, "How to use T-SNE effectively," *Distill*, vol. 1, no. 10, p. e2, 2016. [Online]. Available: http://distill.pub/2016/misread-tsne

[38] G. O. Campos, A. Zimek, J. Sander, R. J. G. B. 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 Knowl. Discovery*, vol. 30, no. 4, pp. 891–927, Jul. 2016.

[39] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," 2016, *arXiv:1605.07717*. [Online]. Available: http://arxiv.org/abs/1605.07717



OKWUDILI M. EZEME (Member, IEEE) received the Bachelor of Engineering degree in electronic engineering from the University of Nigeria, Nsukka, the M.A.Sc. degree from the University of Toronto, Canada, and the Ph.D. degree in electrical and computer engineering from Ontario Tech University. His research interests include the intersection of applied machine learning and cybersecurity.

QUSAY H. MAHMOUD (Senior Member, IEEE) is currently a Professor of software engineering with the Department of Electrical, Computer, and Software Engineering, Ontario Tech University, Oshawa, Canada. He was the Founding Chair of the Department, and more recently, he has served as an Associate Dean of the Faculty of Engineering and Applied Science with the Ontario Tech University. His research interests include intelligent software systems and cybersecurity.

AKRAMUL AZIM (Senior Member, IEEE) is currently an Assistant Professor with the Department of Electrical, Computer, and Software Engineering and the Head of the Real-Time Embedded Software (RTEMSOFT) Research Group, Ontario Tech University, Oshawa, 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.

• • •