IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 October 2022, accepted 8 November 2022, date of publication 24 November 2022,
date of current version 23 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3224583

RESEARCH ARTICLE

iKnight–Guarding IoT Infrastructure Using Generative Adversarial Networks

SANTOSH NUKAVARAPU, (Member, IEEE), AND TAMER NADEEM, (Senior Member, IEEE)
Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23220, USA

Corresponding author: Tamer Nadeem (tnadeem@vcu.edu)

This work was supported in part by the U.S. National Science Foundation under Grant CNS-1764185 and Grant OAC-2212424; and in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber research and development, innovation, and workforce development.

ABSTRACT With the phenomenal growth of IoT devices and their exponentially increasing applications at the edge of the network, it has become imminent to provide secure, flexible, and programmable network services to support user privacy, security, and quality of service. However, such services can only be enabled and effectively applied when the edge systems automatically identify these devices. Existing IoT classification systems are based on supervised training methods that require labeled data and manual feature extraction. Such approaches suffer from many challenges, such as privacy, labeling efforts, and adaptability to new devices. This paper develops a multistage multi-class classifier based on semi-supervised generative adversarial networks that perform automatic feature extraction with minimal labeled data. The classifier can identify devices with 96% accuracy when only 3% of the training data is labeled. Moreover, the classifier can correctly infer the device type (IoT, non-IoT, and anomaly) of any new device with 90% accuracy. We also show how our model can support various features such as noise robustness, continual learning ability, and novelty detection, such as zero-day malware attacks. Finally, we integrate our classifier with a software-defined network to enable smart IoT network services at the edge.

INDEX TERMS Catastrophic forgetting, convolutional neural networks, continual learning, edge computing, generative adversarial networks, Internet of Things, IoT privacy, IoT security, semi supervised GAN, software defined networking.

I. INTRODUCTION

The advent of edge computing has given rise to a plethora of extraordinary new use-cases and applications, the most important and ground-breaking of which is the IoT. Edge-based IoT applications are anticipated to grow, with an estimate of 41.6 billion devices producing 79.4 ZB of data by 2025 [1] and 300 million smart homes by 2023 [2]. The bulk of these devices are linked to distant servers and communicate on a regular basis or constantly, even without user input [3]. IoT devices can now receive automatic updates, customizable user control, and remote monitoring owing to this cloud-connected design.

The exponential growth of IoT devices, their use cases, and their autonomous communication behavior have created new difficulties for the edge infrastructure. Recent studies have demonstrated numerous potential network-based security and privacy attacks [4]. Numerous of these devices also offer sensitive functions and have stringent QoS specifications, such as baby monitoring webcams. Therefore, by enabling network-wide services, the edge-based IoT architecture should fulfill various security, privacy, and QoS needs. Additionally, the network services must be adaptable due to the dynamic network features of these devices, such as mobility and resource-constrained needs (sleep periods). Given these difficulties, the upcoming generation of edge-based systems must offer adaptable and customizable network services to aid in the secure and effective operation of IoT devices.

Recently, some studies have presented network-based solutions with programmable and adaptable network services [5]. Nevertheless, various network device types must be automatically detected in order to perform these services

The associate editor coordinating the review of this manuscript and approving it for publication was Giovanni Merlino.

effectively and efficiently. Similar to this, for a network-based security solution to successfully isolate any malware traffic, the malware must be rapidly discovered as soon as it enters the network. Given these specifications, edge-based systems need to efficiently and quickly find any new devices connecting to the network. Additionally, in order to effectively support various services like privacy, security, and quality of service, such device discovery should support various scenarios, such as identifying the known classes of devices and malware, any new classes of devices (unknown devices) that were not included in the classifier training data, and anomalies.

The majority of existing IoT classification methods are based on supervised machine learning models [6], [7].However, there are many problems with these methods, including labeling attempts, privacy issues with data collection, and trouble identifying new devices that weren't covered in the training. Furthermore, the accuracy of supervised learning methods is mostly dependent on the training dataset without taking into account any variance, leaving these algorithms vulnerable to changes in network traffic brought on by concept drift or abnormal network conditions. Furthermore, these classifiers must be adaptable enough to be updated with new classes of devices that join the network through retraining, which often requires access to the previous data, in order to be used in real-world deployments.Retraining the current IoT classifiers is difficult because of the limited storage capacity of the edge devices and the privacy concerns resulting from retaining the network data for extended periods of time.

Generative adversarial networks (GANs) [8] have been shown to be good at modeling unknown complex distributions and making new data that is close to the true distribution of the training data. Given this ability, numerous GAN extensions for generating and classifying tasks have recently been presented. Semi-supervised GAN (SGAN) [9], [10], an extension to the GAN model, has lately demonstrated good results in image classification [9], online spam reviews [11], and medical data [12] where there is a very limited availability of labeled data. The SGAN model consists of a multi-class classifier that can automatically extract features from a small amount of labeled data. Further, SGAN's training process involves exposing the classifier to more synthetic data points containing noise, which improves the classifier's generalization capacity.

The SGAN model has several features that are aligned with the criteria and objectives of constructing a real-world IoT classifier. These features include multi-class classification, automatic feature extraction, the capacity to be robust to noise, and capturing the hidden distribution of distinct classes [13]. Furthermore, the IoT classifier will be able to learn incrementally without the need for any data storage because the SGAN generator can produce synthetic data after training without requiring access to training data.

Motivated by the above observations, in this paper, we develop IoT classifiers for real-world scenarios using SGAN. We summarize the contributions of this paper as follows:

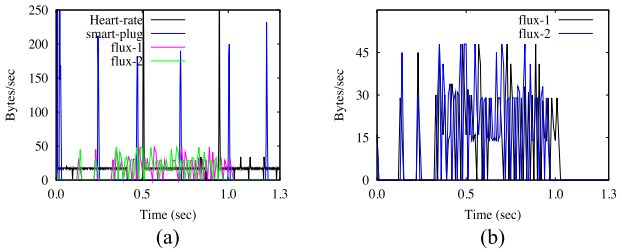

FIGURE 1. a) Traffic patterns of four different IoT devices, b) Zooming into the traffic of two Flux-lightbulb devices shows high similarity.

- We design and develop *iKnight*, a semi-supervised GAN-based multistage multi-class classifier for classifying IoT devices for different scenarios, such as known devices, unknown devices, malware, and anomalies, with very few labeled data points.
- As a use case example of *iKnight*, we design an edge-based privacy preserving framework (*iPrivacy*) for IoT traffic.
- We implement different real-world IoT classification features of *iKnight* such as continual learning and lightweight model deployment, using generative replay and knowledge distillation techniques, respectively.
- We evaluate the features of *iKnight* using real IoT devices and malware network traffic datasets.
- We deploy *iKnight* on real-world edge hardware, the NVIDIA Jetson Nano.

The rest of the paper is organized as follows: In section II, we highlight the significance of IoT classification and describe some of the characteristics of convolutional neural networks, GANs, and semi-supervised GANs that are the building blocks of our IoT network classifier. Next, we describe the different features of real-world classifiers for edge-based IoT devices in section IV and discuss the implementation of our proposed edge-based IoT classifier, *iKnight*, in section V. Other features of *iKnight* such as continual learning and lightweight deployment on the edge, are described further in sections VI and VII, respectively. Furthermore, in section VIII we discuss a privacy preserving service to protect IoT devices from side channel attacks, which is enabled through *iKnight*. Finally, in sections IX and X, we evaluate our *iKnight* classifier on edge-based real-world scenarios and discuss our future work.

II. BACKGROUND

A. THE SIGNIFICANCE OF IoT CLASSIFICATION

Many IoT devices are generally known to have two network subsystems: a keep-alive subsystem and an activity-driven subsystem [14], [15]. The keep-alive subsystem keeps the device connected to the cloud server and enables the device to receive remote communications [14] while the activity-driven subsystem sends the periodic application data such as sensor or user activity events. This type of autonomous network architecture significantly differs from non-IoT devices such

as laptops and smartphones that generally communicate with remote servers only for specific user-based application activities. A recent study showed that IoT devices communicate to cloud servers over 22 days without any user interactions [3].

The online network of IoT model devices makes them vulnerable to many network attacks. For example, an attacker can use a publicly available tool such as Shodan [16] to identify the different vulnerable IoT devices, such as cameras connected to the internet, and create bots for their distributed denial of service attacks (DDoS). Furthermore, this online network architecture may result in side-channel attacks for activity inference at the ISP and local WiFi network [4]. The traffic traces of different activities on a device can have different signatures. For example, based on our prior work, [17], Figure 1.a shows the different signatures for network traffic of different IoT devices. While different IoT devices have unique traffic signatures, the traffic signatures of similar devices are similar despite their different activities, as shown in Figure 1.b. Recently, researchers have shown different attack models for activity inference [4], [18].

Given that IoT devices have low resource constraints, many security solutions, such as anti-malware solutions deployed on non-IoT devices such as PCs and phones, are ineffective for IoT devices. Moreover, unlike non-IoT devices, IoT devices are integrated with different third-party apps for automation purposes and have a larger attack surface due to more autonomous network connections. To address these issues, special network policies for securing IoT devices need to be enforced. Hence, having a classifier that can distinguish between IoT and non-IoT is critical.

The network community is looking to build security and privacy solutions at the edge of the infrastructure to address the above issues. For example, authors in [6] deploy software-defined networking models for security purposes by isolating the network traffic through the CSS vulnerability database and the SDN-based on detecting anomalies. However, to implement and automate any such SDN-based solutions, it is essential to have an effective classification system that is not dependent on labeled data and manual feature extraction.

B. NETWORK TRAFFIC CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

Recent research has proposed using convolutional neural networks (CNN) [19] for network traffic classification [7]. CNNs are a type of deep learning model that has traditionally been used for image classification [20] and object detection [21]. A CNN based deep learning architecture uses convolutional operations to extract features from spatial data such as images. Recently, researchers have shown that network traffic has spatial relationships similar to images, which can be efficiently processed using CNNs [7]. However, like any other supervised deep learning model, CNN is impacted by labeled data availability. In our proposed approach, we use CNN as a classifier in a semi-supervised method based on Generative Adversarial Networks (GANs) [8], which supports training the model with less labeled data but high accuracy.

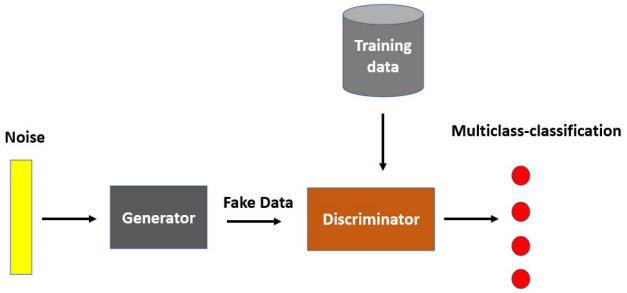

FIGURE 2. Semi supervised GAN (SGAN) architecture.

C. GENERATIVE ADVERSARIAL NETWORKS

For many complex generative tasks, such as image generation, estimating the probability distribution of the features of the images is difficult. Generative Adversarial Networks (GANs) [8] are known to model unknown complex distributions, such as image generation tasks. The GAN model consists of two neural networks, a discriminator and a generator, that form a two-player min-max game where the generator tries to generate fake samples. The discriminator tries to identify if the samples are from the training data (real) or generated by the generator (fake). As shown in Figure 2, the generator model accepts as an input a random noise and then transforms this noise into a fake sample as an output. The min-max game forces the generator to approximate the true distribution. Moreover, the training process reaches an optimum solution when the min-max game reaches Nash equilibrium and the generator's fake samples are predicted as fake by the discriminator with a 50% probability. The GAN training consists of simultaneously training the generator and discriminator models on small batches of data known as mini-batches with a fixed size. Each iteration will train the GAN model with this fixed mini-batch of data points. For a training iteration, the generator is given a batch of random points from the latent space to output a batch of fake samples. This batch of fake samples is given to the discriminator to predict if they are real or fake. The discriminator's loss for predicting fake samples is given to the generator, which updates its weights accordingly to generate better realistic samples. The generator training is called ''unsupervised training'', as the generator model is never exposed to real data during training. After training, the Generator can be detached from the GAN model and independently used for generative tasks. Similarly, the discriminator model can be used for many image classification tasks [9], [10]. Very recently, image-based anomaly detection using GAN discriminator has been proposed [22].

D. SEMI-SUPERVISED GAN

The amount of labeled data has a significant impact on the accuracy of classifiers in machine learning models. However,

many real-world problems have less labeled data and would require significant labeling efforts. Some researchers have proposed utilizing the unlabeled data using semi-supervised learning [23], which uses a few labeled data points to identify the class of the majority of the unlabeled data points. With only a few labeled points, this approach approximately predicts the label of unlabeled data.

Recently, some researchers have proposed semi-supervised learning using GAN, generally known as semi-supervised GAN (SGAN) [9], [10]. The SGAN has shown promising results for image classification problems that have sparsely labeled data [9]. Similar to GAN, the semi-supervised GAN model consists of a generator and discriminator but has an additional classifier model that shares all the layers with the discriminator but has a different output layer [9]. The objective of the classifier is to classify the K classes of the training data, unlike the discriminator, which acts as a binary classifier and only predicts if the data is fake or real.

In SGAN, in addition to training the discriminator on unlabeled data, the classifier is also trained on supervised labeled data. The generator is given the combined loss of both the classifier and discriminator. The rationale behind this is that the feature representations learned from the discriminator to identify the real data distribution improve classifier efficiency. Furthermore, the min-max game-based adversarial training will force the generator to create synthetic samples that create additional data points for the classifier, thus improving its efficiency [10]. In SGAN training, both the discriminator and classifier are exposed to real labeled data, real unlabeled data, and fake data from the generator. This approach generalizes better than other semi-supervised methods that do not have access to additional synthetic data points during training. Moreover, generator-based fake data points include noise that makes the classifier robust to noise in real-world classification tasks.

III. RELATED WORK

In recent years, there has been a great deal of interest in IoT device fingerprinting research [6], [24], [25], and activity inference analysis [4], [18]. While some works proposed supervised classifiers based on features such as device-dependent features for device identification [6], [26], others used time series based features [18], and flow level metadata [4], to detect IoT device activities. Recently, authors in [7] used CNN and RNN based deep learning classifiers for network classification, but this technique is also based on supervised learning that is highly dependent on labeled data availability. Recently, researchers have proposed an unsupervised clustering approach for IoT device classification [27]. In addition, authors in [28] use IoT-based raw network packets as features for classifying IoT devices using autoencoders and Bayesian modeling. We believe that multi-class classifiers are much more helpful for real-world scenarios given their ease of deployment, maintenance, and scalability.

Some researchers used GANs to generate synthetic packets [29], [30], [31] and perform adversarial attacks [32]. However, in this work, we use semi-supervised GANs for classification tasks. GANs have emerged as a popular tool to perform anamoly detection. In [33] authors have used GANs as a tool to generate and detect malware samples. Very recent work has used a semi-supervised based approach towards classifying network applications belonging to QUIC, VPN, and non-VPN datasets [34]. However, it is well-known that IoT-based network traffic is very different from traditional network traffic. In addition, although this recent work is aimed at building classifiers for a limited set of applications with very unique characteristics (e.g., browsing, streaming), building a classifier for IoT network classification with many devices with similar network characteristics is far more challenging. Furthermore, IoT environments have several unique challenges, such as concept drift, dynamic addition of devices, malware infections, and anomaly detection, which our work addresses.

In this paper, we integrate our very recent work [35] which, unlike the traditional approach of using flow-based metadata features, uses a *2D flow-based encoding scheme* that is more relevant for IoT based classifiers since metadata features such as inter-arrival times can have adverse effects on the change in network conditions (e.g., limited bandwidth). In addition, we implement real world IoT classification features such as a continual learning feature using SGAN, which can continuously learn new device classes and deploy the lightweight model at the edge using knowledge distillation. Furthermore, we design a software-defined network (SDN) enabled privacy-preserving service at the edge supported by our IoT classifier. Finally, we deploy and evaluate the performance of our SGAN based IoT classifier on NVIDIA Jetson hardware.

IV. iKnight FEATURES

Our goal in designing and developing *iKnight* with a multistage and multiclass SGAN based implementation is to allow different exciting features that are essential to a real-world edge-based IoT classifier. There are multiple challenges for building an IoT classifier, such as less labeled data, different devices with different network traffic features, and performing multiclass classification for many devices. Moreover, given that these devices can change the firmware, the tool should continuously train on-device data without forgetting the old device class classification. Furthermore, such a tool should be lightly deployed on edge devices.

In this section, we list and discuss the various features of *iKnight*.

A. AUTOMATIC FEATURE EXTRACTION AND LIMITED LABELED TRAINING DATA

Many IoT classifiers are based on supervised learning and require manual feature engineering [6], [36]. However, these techniques rely on the availability of labeled data, which is challenging given the privacy concerns associated with

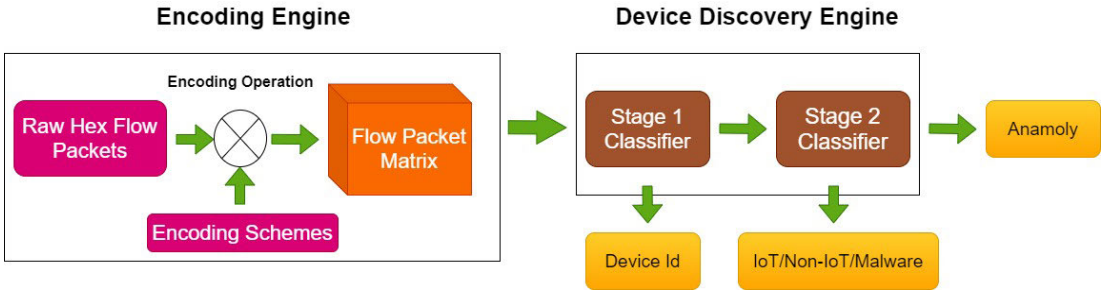

FIGURE 3. ***iKnight*** **architecture.**

the IoT data collection process and the needed labeling efforts. Moreover, the network flows of different devices can have different network characteristics [4]. For example, the network flow of complex IoT devices such as *Withings* baby video monitor differs from the flow characteristics of simple IoT devices such as *Lifx* light bulb. Thus, the manual extraction of universal features that is efficient across all IoT devices is complex. In *iKnight*, the SGAN CNN classifier will automatically extract the hidden features during training. Moreover, the training of the SGAN model for each stage requires only 3% of labeled data. Therefore, *iKnight* supports both automated feature extraction and limited labeled training data.

B. IDENTIFYING UNKNOWN DEVICES AND ZERO-DAY ATTACKS

In real edge networks, devices may dynamically join a network and create challenges for the network administrator in identifying the type of unknown devices (i.e., IoT, non-IoT, malware, or anomaly). In existing multiclass IoT classifier systems, unknown devices are typically classified as known classes that cause serious vulnerabilities. For example, a zero-day attack on network traffic classified as a known device may allow the device to carry out attacks. Detecting the class of an unknown device is called novelty detection, which is supported by the recent implementation of semi-supervised GAN [37]. Given the significance of this feature, *iKnight* utilizes the multistage architecture to realize this feature. More specifically, as described later, *iKnight* uses the multiclass classifier in stage #1 to detect any unknown device, and the multiclass classifier in stage #2 is then used to identify the type (i.e., IoT, non-IoT, malware) of the device. If stage #2 still identifies the device type as unknown, *iKnight* will mark the device flow as an anomaly.

C. GENERALIZED MODEL WITH ROBUSTNESS TO NOISE

IoT network traffic suffers from concept drift, where the network traffic signatures of the device may change over time due to firmware updates [27]. Moreover, bad network conditions such as high packet drops can create noise in the network flows and have an adverse effect on classification tasks. Furthermore, an overfit model to training data can miss-classify a device-specific flow that has a slight deviation from the training data. Therefore, a robust classifier should be well generalized to the training data and effectively robust to noises, such as changes in payload and packet sizes or missing packets. In *iKnight*, the semi-supervised GAN model classifier is exposed to adversarial noise in the fake samples from the generator, allowing the SGAN classifier to adjust its weights for any small deviation from the training data. Therefore, *iKnight* will be resistant to noises in both stages.

D. FLEXIBLE AND CONTINUAL LEARNING MODEL

With the addition of new devices to the network, the model should update its knowledge through incremental training on new network data. However, this is challenging given the limited data storage space of edge systems. In addition, the preservation of IoT data for a long period of time would pose challenges to privacy. In *iKnight*, the semi-supervised GAN Generator learns the distribution of the training data during training, enabling *iKnight* to use the generator to produce synthetic data similar to the training data without requiring any access to previous data. Moreover, GAN-based models can also learn incrementally without fully retraining by using the synthetic samples generated by the generator while learning new class data. Therefore, *iKnight* is flexible enough to continue to learn new data.

In view of these features, iKnight can be used effectively and efficiently for device and device type identification. This device type discovery capability of iKnight can enable different network services on the edge, which we will discuss next.

V. iKnight FRAMEWORK

In this paper, we design and implement *iKnight*, an IoT network classifier for edge-based systems. We discuss below the architecture and implementation of *iKnight*.

A. ARCHITECTURE

Figure 3 shows an overview of the *iKnight* framework, which consists of an Encoding Engine and a Device Discovery Engine. In *iKnight*, the encoding engine encodes the device's flow and passes the encoded flow to the device discovery engine. The device discovery engine uses

multi-stage multi-class SGAN-classifiers trained using two different SGAN-models for identifying the device ID (device name) and its type (IoT vs. non-IoT); respectively. In this paper, we use SGAN-model to refer to the combined models of SGAN-generator, SGAN-discriminator, and SGAN-classifier. We discuss below the design of the encoding engine and device discovery engine.

1) ENCODING ENGINE

The *iKnight* encoding engine encodes the raw network flow packets of new devices using an encoding scheme. An encoding scheme is a process of mapping the raw network packets to a feature matrix that we define as a flow encoded matrix, which is sent to the device discovery engine for the device's name or type identification. The multi-stage multi-class SGAN-classifier in the device discovery engine uses these raw encoded features to extract the hidden features and identify the device ID or its type automatically. An efficient encoding scheme is a significant factor in the efficiency of device type identification. In *iKnight*, we use two encoding schemes; *packet-only* and *packet-with-inter-arrival-time* schemes that we discuss later in the implementation subsection.

2) DEVICE DISCOVERY ENGINE

The *iKnight* Device Discovery Engine is a multi-stage classifier that can identify the device under different scenarios, such as identifying the device ID for a known device and the device type for an unknown device. We define a device as known if it is available to the SGAN-classifier during training and unknown otherwise. More specifically, two different classifiers with different sub-objectives help with device discovery. The stage #1 classifier's objective is to identify the known device, and the stage #2 classifier's objective is to identify the device type (IoT or non-IoT) for any unknown device. It is to be noted that unlike most of the traditional multi-class classifiers that classify any unknown samples during inference as one of the known classes from the training data, the stage #2 classifier can identify any unknown device class as unknown. However, we refer to the unknown class in stage #2 as an anomaly, given that it does not fall into any known network traffic classes from the training data. This type of inference at stage #2 will help the network administrators identify the types of unknown devices and, consequently, adapt network policies. Some of these policies can give higher network priority to specific device types (IoT vs. non-IoT), enforce any known anti-malware patches for malware, and issue alerts in case of an anomaly for immediate action. It is to be noted that we use two different SGAN-models to produce two different classifier models for stage #1 and stage #2. We only activate the classifier at stage #2 if the confidence of the stage #1 classifier is low in identifying the device uniquely, and only then the input is processed by the stage #2 classifier to identify network flows as IoT/non-IoT devices or anomalies. This conditional activation of the classifier avoids additional overhead on edge-based resources such as power and latency. In this paper, we refer to this sequence of multi-classifiers as a ''multi-stage classifier'', given the adoption of this term in the machine learning and systems community for objectives similar to ours [38].

TABLE 1. The list of IoT and non-IoT devices used in experiments.

Device Name	Device Type
Smart Things	Hub
Amazon Echo	Speaker
iHome	Speaker
Triby Speaker	Speaker
Netatmo Welcome	Camera
TP-Link Day Night Cloud Camera	Camera
Samsung SmartCam	Camera
Dropcam	Camera
Insteon Camera	Camera
Withings Smart Baby Monitor	Camera
Nest Dropcam	Camera
Belkin Wemo Switch	Acutator
TP-Link Smart Plug	Acutator
Light Bulbs LiFX Smart Bulb	Acutator
NEST Protect Smoke Alarm	Sensor
Netatmo Weather Station	Sensor
Withings Smart Scale	Sensor
Blipcare Blood Pressure Meter	Sensor
Withings Aura smart Sleep	Sensor
PIX-Star Photo-frame	Digital Frame
Laptop	non-IoT
Android Phone	non-IoT
iPhone	non-IoT
Samsumg Galaxy Tab	non-IoT

B. IMPLEMENTATION

We discuss below the datasets, training procedure, and SGAN-model architecture for both stages in detail.

1) DATA SETS COLLECTION FOR iKnight

We use the publicly available UNSW [26] and IoT-23 datasets [39] to validate and evaluate our proposed device and device type identification approach. The UNSW network dataset consists of network traffic collected from multiple IoT and non-IoT devices over a period of 4 weeks. The researchers deploy the real-world IoT devices and collect the network traffic in real-time, which is generated as a result of the different activities of the IoT and non-IoT devices. The devices are connected to a LAN network and consist of both wireless Wi-Fi enabled devices and wired devices connected to an access point. The testbed architecture consists of energy management devices, controllers/hubs, cameras, appliances, health monitors, and non-IoT devices. More information about the collection of network pcap files, connectivity scenarios, and other processing details can be found in their paper [26]. We use the network traffic traces from different classes of IoT devices and non-IoT devices from the UNSW dataset, as shown in Table 1. The IoT-23 consists of labeled IoT malware network traffic flows from twenty different malware binary files (each class) executed on a Raspberry PI device in a controlled environment, with flows

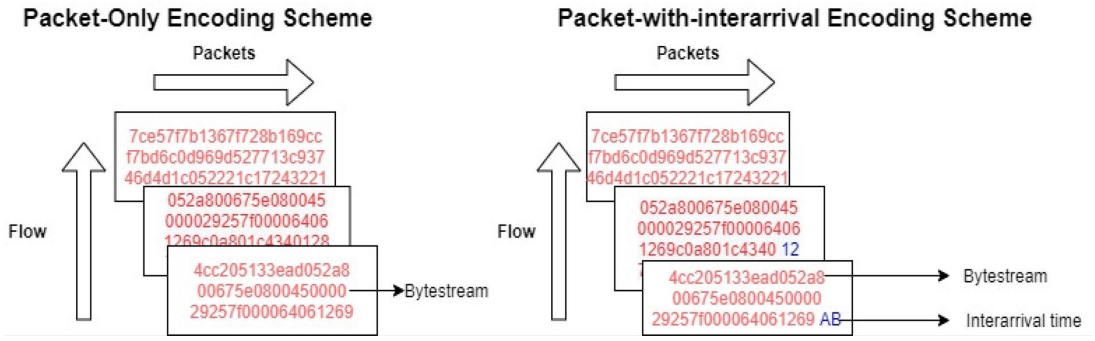

FIGURE 4. *iKnight* encoding schemes.

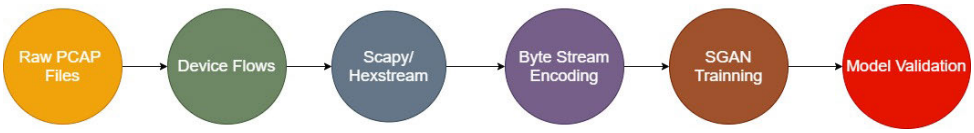

FIGURE 5. The model training pipeline of *iKnight* stage 1 and stage 2.

labeled benign or malicious. For our experiments, we selected 60,000 flows from the UNSW dataset and 10000 malicious flows from four different malware classes, Mirai, Trojan, Hakai, and Torii. For the stage #1 model, we train the model with twenty IoT devices. For the stage #2 model, we use the combined network flows of twenty IoT devices, four non-IoT devices, and four malware classes.

2) DATA-PREPOSSESSING FOR iKnight STAGE #1 AND STAGE #2

For both stages, stage #1 and stage #2, we train two different SGAN-models and output two different SGAN-classifiers to identify device ID and device type identification. Figure 5 shows the SGAN-model training pipeline that we repeat for each stage. We first extract flows from the raw pcap files using the pkt2flow tool [40], which splits the pcap files into individual flows. We then convert all the packets of a flow into raw byte streams using *scapy* tool [41]. For both models, we use raw TCP, and UDP flows for our training data. We remove the MAC layer header and device-dependent fields (source address, destination address, destination port, source port, and checksum) from IP and transport layer headers to make our training data network independent. We then train the SGAN-model at each stage with two types of training data: payloads plus network independent headers, and payloads only. We discuss the effect of the selection of each training data set in the evaluation section. The training data at each stage consists of randomly sampled flows for all classes with a balanced representation.

3) BYTE STREAM ENCODING FOR iKnight STAGE #1 AND STAGE #2 SGAN-MODELS

For image classification tasks, transforming the training image data by rotating or transforming can show performance improvements for the classification tasks. Similarly, in *iKnight*, the design of arranging the bytes of network packets in a flow can effectively impact the features extracted by the discriminator and have an overall impact on the SGAN training. In this paper, we take raw bytes as our data and then arrange them in a sequence of three dimensions: width, height, and length, so that the 3D kernels of the CNN layers can better understand the relationships between the bytes of different packets in a flow and their relationships. For example, for an RGB image, the channel filters will extract the relationship between pixels of different colors in a non-gray image. Similarly, in *iKnight* the intuition behind this byte encoding scheme is that the channel's filters will understand the spatial and temporal relationships between the bytes of the headers and the payloads of different packets. Moreover, we believe that we can encode other information along with the bytes, such as inter-arrival times, that can help increase the efficiency of the model. We do note that coming up with the most efficient encoding scheme is challenging research, a work in progress, and outside the scope of this paper.

From Figure 4, in the *packet-only* encoding scheme, we use the raw network byte-stream representation of the packets as features. We first convert each byte-stream representation of the packet of the flow into its equivalent hexadecimal integer value. This hexadecimal flow array is then encoded as a three-dimensional image matrix, a flow encoded NumPy matrix with height, width, and depth. Each row of the flow encoded matrix consists of a packet hex stream arranged as a 2-dimensional array with a size of 56*56 (accommodating the maximum 1500 bytes of a MAC packet). The packet rows are ordered in the flow matrix as per their arrival sequence. This encoding scheme is similar to the pixel intensity arrangement in a non-gray image matrix, where each row is a 2D matrix representing the different channels of the image. The flow

encoded matrix's depth is the number of packets in the flow, a configurable parameter, which we find to be the first 5 packets based on experiments. If the flow has fewer than 5 packets, then the rest of the packet rows of the flow encoded matrix are appended with zeros, an approach similar to [28].

Similar to the *packet-only* encoding scheme, in *packet-with-interarrival* scheme, we use both the raw network byte stream and the packets' inter-arrival time. We arrange the 2D packets in the flow encoded matrix as rows, similar to the *packet-only* scheme. However, we append an additional row at the end containing the inter-arrival times of the packets in their arrival sequence. Moreover, we convert the inter-arrival values to a hexadecimal value similar to the packet data. We discuss the impact of these encoding schemes on device type inference in the evaluation section.

4) STAGE #1 AND STAGE #2 GAN ARCHITECTURE

For both stage #1 and stage #2, we implement the SGAN-model architecture described in [9] that consists of a SGAN-generator, SGAN-discriminator, and SGAN-classifier [9] for *iKnight*. The classifier shares the weights with the discriminator [9] and has a Softmax layer [42] used to classify the flows into the names or types of corresponding devices. For our SGAN-generator and SGAN-discriminator models, we adopt an architecture very similar to that of the DCGAN guidelines [43]. Note that the approach of using DCGAN for SGAN was also very recently used by [34]. However, we design our model with layers and hyper-parameters optimized specifically for byte stream-based flow encoding scheme training data, which consists of inputs of a 56*56*n Numpy array, where n is the number of packets in the flow. For SGAN-generator, SGAN-discriminator, and SGAN-classifier, we use Adam optimizer [44]. Below, we discuss architectures for the SGAN-generator, SGAN-discriminator, and SGAN-classifier, as well as hyper-parameters. The SGAN-generator model consists of 5 layers. The first two layers are fully connected layers that accept an input noise vector of size 100 and give an output of shape 256*7*7. The output is reshaped into a vector of size 12544. The next hidden layers perform convolution transpose operations with a shape of 256*256. A convolution layer follows this with the tanh activation function. Each of the convolution transpose layers is followed by the Leaky ReLu activation function and batch normalization layers. The output of the generator is a matrix with dimensions 56*56*n.

The SGAN-classifier architecture consists of six layers. The first layer is a fully connected layer that accepts an input of 56*56*n from the generator. The next four hidden layers are convolution layers that perform the convolution operations, having 2*2 filters of stride 3*3. The first and last hidden convolution layers have a shape of 128*128, while the rest of the layers have a shape of 256*256. Each of the three hidden layers is followed by the Leaky ReLu activation function and batch normalization layers. The output of the last hidden layer is reshaped and given to the fully connected dense layer. Finally, the SGAN-classifier output is a Softmax layer that provides the probability of each training class's predictions. For the SGAN-discriminator, we share all the layers with the SGAN-classifier except the output layer. The input to the SGAN-discriminator is the output from the SGAN-classifier's last layer activation before Softmax. This SGAN-discriminator implementation is known to increase the accuracy of the classifier [9]. The SGAN-discriminator's output layer is a Sigmoid activation function, classifying whether the input flow is real or fake.

5) STAGE #1 AND STAGE #2 MODEL TRAINING

In stage #1 SGAN-model training, we split the dataset with 80% for training and 20% for testing. As discussed earlier in Section II, SGAN-model training consists of training the SGAN-generator, SGAN-discriminator, and SGAN-classifier with mini-batches of data in each epoch. Therefore, we select labeled flow samples for training the SGAN-classifier and unlabeled flows for training the SGAN-discriminator. Then, we train the SGAN-model for multiple epochs, where each epoch consists of multiple batches of training data. We then validate the stage #1 SGAN-classifier model with the testing data. The SGAN-model training process for stage #2 is similar to stage #1, except that we filter the flows of a specific device and use the filtered flows to infer the device type class during validation. For example, we filter out all the flows of Dropcam during training and then evaluate the trained SGAN-classifier model with these filtered flows to validate whether the trained model can infer the correct type of device as IoT. We repeat this process for all the device classes (IoT, non-IoT, and malware) and report the average accuracy. This process effectively evaluates the ability of the SGAN-classifier to infer unknown devices, which is discussed in the evaluation.

During the training of the SGAN, the SGAN CNN-based discriminator does the feature extraction, whereas the SGAN generator has convolutional transpose layers that learn the features to transform a random input noise vector into synthetic samples similar to training data. Specifically, in the case of *iKnight*, the GAN CNN discriminator is extracting the features from the raw bytes and their sequence in a network flow to identify the difference between real and fake network flows, while the generator is trying to learn the features to generate a fake network flow similar to the training data. The input to the generator is a random vector of 1*100 dimension and the output is a fake network flow of dimension 56*56*5. The 56*56*5 output from the generator is then given to the classifier/discriminator. While the classifier's output is a softmax function that gives the probabilities of all the device classes, the discriminator's output is 0 or 1 to distinguish between real and fake flow.

Both stages of *iKnight* framework utilize the SGAN-classifier to achieve their objectives: known-device identification and unknown device type identification. The SGAN-generator model can also be utilized by *iKnight* for learning to classify new device classes or identify new device

Algorithm 1 Implementation of *iKnight* Continual Learning Using SGAN

```
Step 1: generator, discriminator, classifier =
perform_sgan_training();
Step 2: i = 1;
Step 3: while i<=N do
    Step 3.1: Current_task_data =
    get_ current_task_data(i);
    Step 3.2: Total_number_device_classes_prior_tasks
    = get_number_classes(i);
    Step 3.3: Total_number_samples =
    get_total_number_samples (Total_number);
    Step 3.4: while Number_samples_generated_per_
    class <= Total_number_samples_each_to_generate
    do
        Step 3.4.1: generated_samples =
        generator(1000);
        Step 3.4.2: generated_samples, class_labels =
        classifier(generated_samples) ;
    end while
    Step 3.5: final_training_data =
    current_task_data.append(generated_samples,
    class_labels);
    Step 3.6: generator, discriminator, classifier =
    perform_sgan_training();
    Step 3.7: i + = 1 ;
end while
```

types that were not seen during the initial training process, which we discuss next.

VI. iKnight ADAPTABILITY AND CONTINUAL LEARNING

IoT devices' network traffic can change over time due to firmware updates by the vendor or server API changes. This change of network traffic over time is called concept drift and can cause the IoT classifier's accuracy to drop over time as they are dependent on network features such as flow level features (number of bytes sent or received, etc.) or packet-level features (payload signature, etc.) derived from original network traffic. Moreover, to keep supporting the device or device type identification, the classifier must be retrained with the updated traffic features. Furthermore, the IoT classifier would have to learn to classify any new device classes and their device types that are not possibly seen during training but added to the smart environment during the inference stage in an ad-hoc fashion. Therefore, the classifier should support class-incremental learning [45], a machine learning technique where the classifier can be trained on batches of classes incrementally. However, existing IoT classifiers [6], [7] do not support class-incremental training due to catastrophic forgetting [46], a phenomenon where the classifier accuracy will significantly drop after being incrementally trained on new class data. The accuracy decreases because the model forgets the parameters (neural network weights) learned during prior training when retrained on the new class data.

To demonstrate the effect of catastrophic forgetting in an IoT classification scenario, we pick a subset of training data described in Section V consisting of 8 devices and split this subset into two datasets: Task #1 and Task #2, composed of 4 devices each. We then train a fully supervised CNN classifier with the Task #1 data and further retrain it on the Task #2 data. In this paper, we will refer to this approach of training and then retraining on different tasks successively as class-incremental learning. Table 4 shows the accuracy of the CNN classifier and SGAN classifier for class-incremental learning. The accuracy of the CNN classifier reaches 100% after training it with the Task #1 data, but the accuracy drops to 0% after retraining the classifier on the Task #2 data. The rationale behind this significant drop is that during the Task #2 training, the model updates its parameters learned during the Task #1 training to classify the new classes of Task #2, thus forgetting the knowledge to classify the Task #1 classes. A naive solution to address this challenge is to retrain the classifiers with Task #1 and Task #2 combined data from scratch. However, this approach would require storing prior task data, which is highly inefficient for storage reasons. Moreover, the IoT network data contains very sensitive information [4] and keeping it on edge or remote servers for a long time could have severe security and privacy risks. Therefore, addressing catastrophic forgetting, which is still an open and challenging area in the machine learning community, needs to be explored by the network community to support IoT-based class-incremental learning.

To address the catastrophic forgetting problem, we utilize the SGAN-generator's ability to mimic training data to implement continual learning in *iKnight*. We use the prior task generator to generate prior task synthetic samples and then augment them with the new task data to train the classifier incrementally. Note that this approach is similar to [47] where the old generator (from the prior training task) of ACGAN [48] is used to address catastrophic forgetting in continuous image generation tasks. However, in this paper, we implement the generative replay approach using the SGAN-generator for IoT-based network classification tasks. With this approach, edge systems do not have to store any old training data, thus not requiring additional storage. Moreover, this approach would require less computation time as it is not necessary to train the model from scratch.

In Algorithm 1, we show the continual learning algorithm for *iKnight* adaptability. In step #1, SGAN training for the current task or initial set of training data is performed using the semi-supervised GAN training as described in Section II. This step produces three models: a generator that can generate trained task data from synthetic data, a discriminator that can discriminate between trained task data and synthetic data, and a classifier that can classify the devices of the trained task. Next, step #3 is the core of the algorithm's implementation and is responsible for incrementally training the prior task

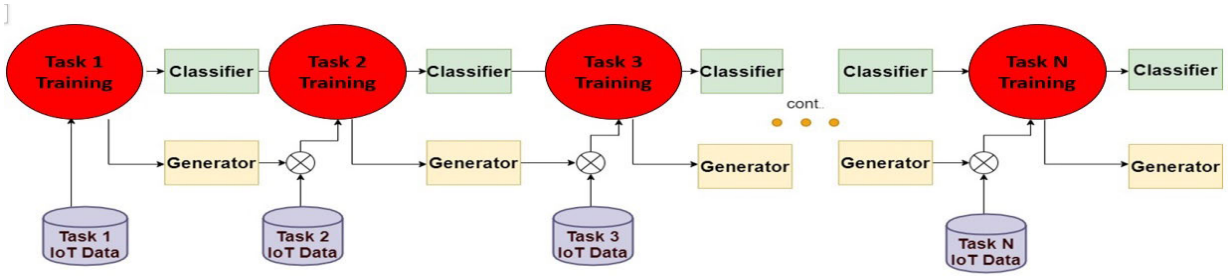

FIGURE 6. ***iKnight*'s continual learning across n number of tasks.**

classifier on the new task data. Steps #3.1 to #3.4 generate the synthetic data representing the prior tasks and append it to the current task training data, which is then used as the training data for the SGAN model. We will discuss each of these steps in detail next. In step #3.1, i^{th} task training data is loaded, followed by steps #3.2 and #3.3, which calculate the number of synthetic samples per class that need to be generated to represent all the prior i-1 tasks data. Next, in step #3.4.1, the generator from the i-1 Task generates synthetic samples, and in step #3.4.2 the class labels are assigned using the i-1 task classifier since the generator randomly generates the samples without any class labels. Steps #3.4.1 to #3.4.2 are repeated until a balanced dataset representing all the old prior task classes is collected, as calculated in step #3.3 earlier. Finally, in step #3.5, this balanced synthetic data representing prior tasks is appended with the new i^{th} task data to train the i-1 SGAN-model in step #3.6. At the end of step #3.6, the updated generator, discriminator, and classifier models can generate, discriminate, and classify all the tasks from Task 1 to Task i respectively. These updated models, along with the following task data, become the input for the next iteration. All the sub-steps under step #3 are repeated for N tasks incrementally, as shown in Figure 6 using the i-1 SGAN-classifier, i-1 SGAN-discriminator, and i-1 SGAN-generator. This approach achieves good accuracy even without access to old training data (ground truth), as shown later in our evaluation section. It is to be noted that for each new task, the algorithm will update the existing SGAN classifier that is pre-trained with all the prior classifications to include classifying a new class/task, and therefore, during inference on the edge device, only the latest updated SGAN classifier will be active. The training required for SGAN-assisted continual learning does not have to be done in real-time and could be either performed on edge or cloud servers since IoT devices are always connected to them and can download the updated model on a periodic basis to support new classification tasks. This approach achieves good accuracy even without access to old training data (ground truth), as shown later in our evaluation section.

Overall, *iKnight* can support many features required for edge-based IoT classifiers, as discussed in Sections V and VI. Given these abilities, we discuss in the next section our strategy for optimizing *iKnight* for edge-based deployment on resource-constrained devices such as routers, switches, and edge servers, which have strict application requirements such as high inference speed, a small memory footprint, and low power utilization.

VII. iKnight AT THE EDGE

In recent years, there has been an emergence in the deployment of deep learning-based applications on edge-based devices [17], [49], [50]. The main characteristics of these edge-based applications are low latency, low memory, and low energy consumption. Given *iKnight* will be deployed on the edge networks, in this paper, we optimize the *iKnight* classifiers into very lightweight models that can achieve similar accuracy but with significantly less memory footprint and increased inference speed. This section discusses our approach to implementing the lightweight version of the *iKnight*.

Many well-known deep learning models, such as RESNET [51] and VGG [52] have an intricate model design with many layers and parameters to learn. For example, RESNET, which is trained on the CIFAR-10 image dataset, [53] has a complicated model design with more than 100 layers and a total of 1.7M parameters. However, deploying such models on edge devices is challenging due to resource constraints. To address this challenge, recent studies [54], [55] have proposed a knowledge distillation technique where a lightweight model is distilled with the knowledge of an already trained and complex model. In this approach, a teacher model is first trained on the training data, which learns the parameters (network weights) required for classification tasks. During the training, the Softmax layer learns each class's probability distribution, which can be smoothed to form Softargets or training labels for the student model. Smoothing the probabilities can help discover other information about the training data, such as the proximity of different class features to the predicted class, and can help the student model to converge quicker. Therefore, unlike the traditional method of training the model with training samples and corresponding labels, the student model is trained with training samples and the teacher model's predicted soft probabilities, as shown in Figure 7. Note that softening the probabilities is done through a hyper-parameter

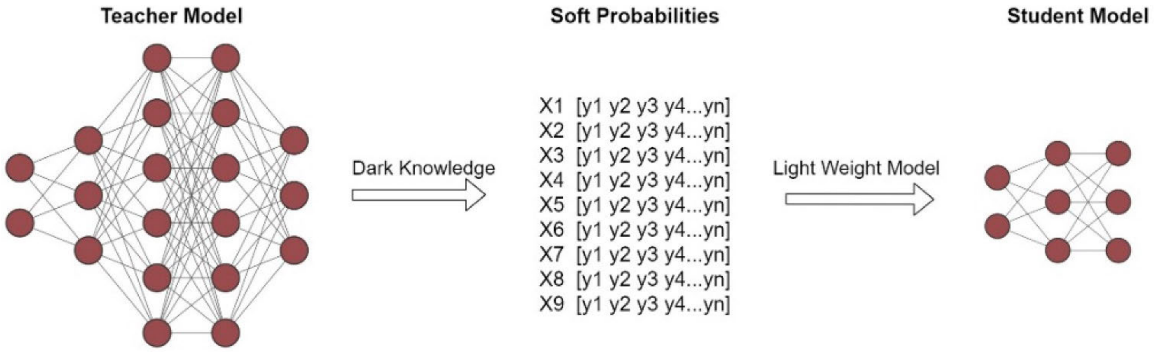

FIGURE 7. ***iKnight*** **lightweight transformation using knowledge distillation.**

called Temperature; the higher this value is, the smoother the probabilities.

In this paper, we implement the knowledge distillation approach using the *iKnight* Stage #1 SGAN-classifier as a teacher and a new lightweight CNN model as a student. We take the trained stage 1 SGAN-classifier as a teacher model with four convolutional layers as discussed in Section 5, using SGAN to create soft probabilities or soft targets for the student model. We then train a very small CNN model with only one convolutional layer, which has fewer layers than the teacher model and uses soft probabilities-based training data. The student model achieved an accuracy of 91% very close to our Stage #1 teacher model but with fewer layers, as shown in our evaluation section later. In our experiments, we have evaluated different student model architectures and found that the CNN model with one convolution layer gives the best accuracy. With this approach, we optimized the *iKnight* Stage #1 classifier for edge-based scenarios with less inference time, a smaller memory footprint, and lower power consumption, which we discuss in our evaluation section. One of the other advantages of this approach is that we can train our Stage #1 and Stage #2 SGAN-classifiers anywhere on cloud or edge servers and then deploy lightweight models much more easily with the help of knowledge distillation on the edge devices.

From sections 5 and 6, we can see that *iKnight* is equipped with different features that can discover device types, both known and unknown, and can continuously learn new device types. Moreover, the knowledge distillation approach, *iKnight* can be transformed into a much lightweight model and can effectively support many new network services at the edge. In the next section, we discuss the potential of *iKnight* to provide such services.

VIII. iKnight EXAMPLE—A FLEXIBLE PRIVACY SERVICE FOR IoT APPLICATIONS

Many recent works have proposed fine-grained and programmable network security policies targeting devices [17], [49]. However, to enable such services, we must automatically identify the device type of the device in the edge system. The automatic device type discovery abilities of *iKnight* can enable such applications at the edge and support dynamic and flexible IoT-based network services such as privacy, security, and quality of service.

We define a network service as a set of programmable APIs that define and configure different network operations (schemes) with configurable parameters dynamically. For example, a privacy-based service can have an API method to dynamically enable privacy-preserving schemes such as packet-delay with a 1-second delay. These service API methods are mapped to specific device type policies, where a policy is a device type identification for a particular context (e.g., user location, time, battery level, network load, etc.). The *iKnight* framework can enable different types of network service at the edge that can programmatically implement such policies. In this paper, as a case study, we propose an edge-based privacy-preserving service, *iPrivacy* to secure the IoT device network traffic from side-channel attacks. *iPrivacy* utilizes *iKnight* for device or device type identification to enable certain privacy-preserving policies. We discuss below the objectives of *iPrivacy* and their significance for IoT privacy.

A. iPrivacy OBJECTIVES

1) FLEXIBLE PER-DEVICE-TYPE SCHEMES

IoT devices have different network requirements. For example, a video camera needs low latency and high throughput requirements. Simultaneously, a light bulb with only two actions (ON or OFF) can accommodate a more tolerable network condition. In addition to this, users can categorize their IoT devices into multiple sensitivities based on usage behaviors. For example, a bedroom camera could be more sensitive than an outdoor camera. Therefore, different IoT devices have different privacy and operational requirements. Given this, a privacy-based service should be flexible in applying different schemes to different IoT devices' requirements. For example, a flexible privacy service should apply a high obfuscation scheme to a baby monitor and a less obfuscated scheme to a light bulb.

2) PROGRAMMABLE POLICIES

Different types of privacy schemes can have different configurations, which can have different performances. For example, a privacy scheme with traffic shaping and a high packet delay can have more latency than one with a low packet delay. Similarly, a traffic sharing scheme with maximum packet padding and high delay can have more obfuscation than one with only packet delay. Therefore, an edge-based privacy service should have APIs to support multiple schemes and dynamically configure them. Moreover, the service needs to define rules (policies) to map individual devices to their schemes.

3) CONTEXT AWARE POLICIES

Different privacy schemes can have different impacts on the performance of an IoT device. For example, a high obfuscation scheme on a bedroom camera can greatly impact the video's streaming, while low obfuscation can have good performance but less privacy. Moreover, a user could choose a better performance scheme with low obfuscation while he is away and remotely monitoring the house and high obfuscation when he is inside the bedroom. Therefore, based on the user context (location here), there could be different privacy policies on the same device. Moreover, a user could also choose to have a low obfuscation scheme during low bandwidth and a high obfuscation scheme during high bandwidth, thus setting a policy based on network conditions (context). Given these requirements, *iPrivacy* would develop different policies based on different contexts, such as application requirements, user objectives, and network conditions.

4) POLICIES ARE TRANSPARENT TO DEVICES

In *iPrivacy* we address the side-channel attacks on IoT devices without any modification of the end-user device network model or physically changing the firmware of the device. Therefore, privacy can seamlessly integrate with smart environments that have IoT devices without any modifications.

Given these objectives, we discuss next, the architecture of *iPrivacy* where we integrate our prior work *PrivacyGuard* [17], an extreme SDN framework, with *iKnight*, which is inspired by our vision of pushing the Software-Defined Network (SDN)-like paradigm all the way to the wireless network edge [50], [56].

B. iPrivacy ARCHITECTURE

We design *iPrivacy* by extending and deploying SDN components (e.g., Open vSwitch [57]) on WiFi access points (in-home or campus environments) or proxy servers (in open and public WiFi hotspots). In *iPrivacy*, the extreme SDN works independently without requiring any collaboration from the SDN at the network core. To apply the network service policies per device based on device requirements, user objectives, device characteristics, and network conditions in real-time, we create one or more virtual network slices between the edge IoT devices and the APs/proxy servers. Figure 8 shows a typical use case of *iPrivacy*, which consists of edge devices and an *iPrivacy* service that runs on the Wi-Fi access point. The *iPrivacy* service only applies traffic shaping policies per-device per-flow on the selected set of sensitive IoT devices provided by the user and utilizes Open vSwitch (OVS) to set and enforce traffic shaping policies correctly on the network flows from/to IoT devices, as we describe in the subsections below.

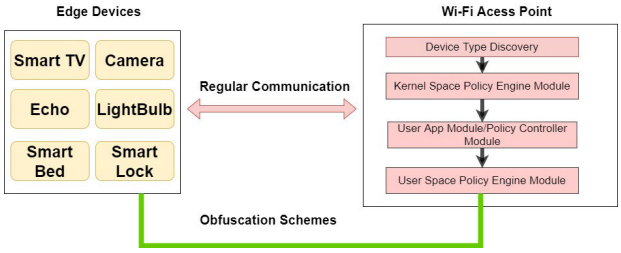

FIGURE 8. *iPrivacy* System design.

1) USER-APP MODULE

This module provides a user interface in which users could categorize the installed devices into different sensitive or priority classes. In addition, users utilize this interface to define the network service schemes for individual devices or devices of similar sensitivity under different contexts.

2) DEVICE TYPE DISCOVERY

This module's main function is to track any new device and map it to its corresponding device type. On seeing a new MAC address, the module first collects a flow with the first five packets from the device and sends it to *iKnight* running on the infrastructure agent. The *iKnight* which is a network independent device inference model, can infer the device type without any network information such as MAC address, IP address, ports, etc., identifying the device type. The module then builds IP-to-device-type information for the discovered device type and passes it to the policy controller module.

3) POLICY CONTROLLER MODULE

This module is responsible for creating and maintaining the device-type-policy table entries containing the network policies by utilizing the user-defined schemes and the device mapping information obtained from the user-app module, and is deployed on the infrastructure agents by the software-defined network local controller. Moreover, this module is also responsible for periodically estimating the current context information of the user (e.g., location) and network (e.g., load) by utilizing the network interface information (e.g., RSSI, throughput). Figure 9 shows examples of table policy entries where each device-type-policy is a single policy describing the rules for each specific device type,

```
Policy #1
  ID: srcIP='A' or dstIP='A'
  CONTEXT: Location='Home' AND Time=[12AM-2AM]
  ACTION: Padding='Normal:μ=1200,σ=100, p=1.0'
          Delay='Uniform:min=0,max=20ms'
Policy #2
  ID: srcIP='A' or dstIP='A'
  CONTEXT: Location='Home'
  ACTION: Padding='Normal:μ=400,σ=100, p=0.6'
Policy #3
  ID: srcIP='B' or dstIP='B'
  CONTEXT: Battery=Low OR WiFi Load=High
  ACTION: Padding='Normal:μ=1200,σ=100, p=0.6'
          Delay='Uniform:min=0,max=20ms'
Policy #4
  ID: srcIP='B' or dstIP='B'
  CONTEXT: Battery=High OR WiFi Load=Low
  ACTION: Padding='Normal:μ=1200,σ=100, p=1.0'
          Delay='Uniform:min=0,max=20ms'
Policy #5
  ID: srcIP='B' or dstIP='B'
  CONTEXT: Battery=High AND Location='Home'
  ACTION: Padding='Normal:μ=1200,σ=100, p=1.0'
          Delay='Uniform:min=0,max=20ms', IPSec
```

FIGURE 9. Device-type policies for *traffic shaping* schemes.

their contexts, and the corresponding network service privacy scheme with their configuration parameters. The policies are created and modified in the flow-policy table inside the Open vSwitch (OVS) of the device-type-policy Engine module using the OpenFlow protocol [58]. We support the addition of new devices and the runtime creation/modification of the network services' policies by extending the OpenFlow protocol, the OVS user-space module (discussed next), and the OVS kernel module (i.e., the OVS datapath).

4) USER-SPACE POLICY ENGINE MODULE

This module is responsible for identifying packets belonging to a new device and extracting the corresponding policy information mapping to the Policy Controller Module device. After searching for the best policy entry, this information is sent to the kernel-space policy engine module, which applies the corresponding network actions to the device's packets. To implement this module, we first extend the Open Flow APIs in OVS and enable the policy controller module to set up the policies. Next, the module runs a user-space thread that, on receiving any ''upcall'' action for a new device packet, will extract its IP header, find the matching policy from the Policy Controller module, and send the corresponding network scheme and configuration parameters to the kernel-space policy engine module. For example, on receiving a device with srcIP='A' or dstIP='A' (device identifier information), the module will query the policy table for the corresponding IP identifier in the policy table, as shown in Figure 9 and will find Policy #1. Next, it sends this corresponding information to the kernel-space policy engine module. In this case, the information is sent when the context is home and the time is between 12 a.m. and 2 a.m. by performing the following actions: a) apply padding by selecting the padding bytes from the Gaussian distribution with μ, σ, and p values; b) utilize

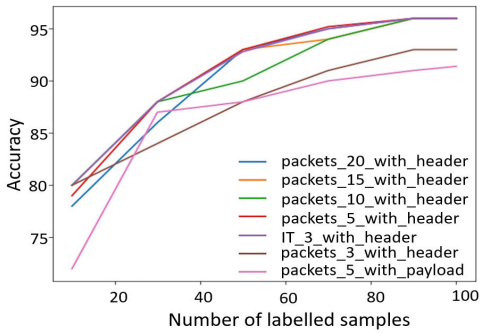

FIGURE 10. The accuracy of stage #1 classifier in *iKnight*'s device discovery engine.

delay for IPTS by choosing random delays from a Gaussian distribution with *min* and *max* values; and finally c) the use of IPSec.

5) KERNEL-SPACE POLICY ENGINE MODULE

This module is responsible for detecting a new device packet and passing it to the user-space policy engine. The user-space policy engine then identifies the new device and related policies from the policy controller module. It feeds this information back to the kernel-space policy engine, configuring the OVS datapath to note the action to be applied on the device packets. Finally, this module is responsible for applying the network actions corresponding to the schemes and configuration parameters.

In this section, we discussed *iPrivacy* which is enabled by the device type discovery abilities of *iKnight*. In the next section, we evaluate the different features of *iKnight* in different real-world IoT classification scenarios.

IX. EVALUATION PERFORMANCE

We implement *iKnight* Device Discovery Engine using Keras [59] and Python, and train it on a GPU enabled machine with 32 GB of installed memory. In the following sections, we evaluate *iKnight* for the different features of a real-world classifier discussed earlier in Section IV.

A. IMPACT OF TRAINING DATA AND ENCODING SCHEMES

Figure 10 shows the performance of the stage #1 classifier under different encoding schemes, including packets_<X>_with_header, packets_<X>_with_payload, and IT_<X>_with_header. The packets_<X>_with_header encoding scheme is a variant of the *Packet-only* encoding scheme in which the bytes of the network independent header fields are encoded with the payload of X packets in the flow, where X is a configuration parameter. Similarly, packet_<X>_with_payload encodes only the payload bytes of X packets without any header fields. The IT_<X>_with_header is a variation of the *Packet-with-interarrival* encoding scheme that encodes the inter-arrival time between packets as well as the bytes of the network independent header fields and the payload of X packets in

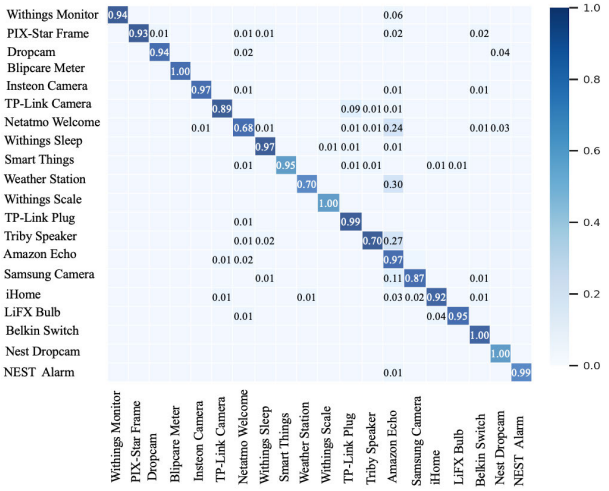

FIGURE 11. The confusion matrix for *iKnight* stage #1 classifier with packets_5_with_payload.

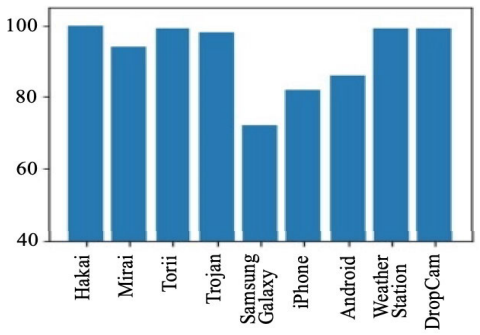

FIGURE 12. The accuracy of *iKnight* stage #2 classifier of unknown devices using packets_5_with_header for different type classes (i.e., IoT, Non-IoT, Malware).

the flow. The figure also depicts how the accuracy of the stage #1 classifier improves as the number of labeled samples increases.

As a result, as previously explained, the device type classification depends on the number of labeled samples. In our experiments, for both stage #1 and stage #2 classifiers, we found that we can achieve a high accuracy with just 3% labeled flows for each device, validating the effectiveness of *iKnight* in achieving high accuracy with just a few labeled data. We show the confusion matrix for the stage 1 classifier using packets_5_with_payload encoding scheme in Figure 11.

In addition, Figure 10 demonstrates that the accuracy of the stage #1 classifier increased from 92% to 96% when with the training data was changed from packets_5_with_payload to packets_5_with_header. Therefore, the network independent header fields of IoT devices significantly improve the IoT device classification since they enable the classifier to uniquely identify each of the device classes rather than just the payload. This is consistent with the recent finding that IoT devices employ specific network configurations by setting the corresponding fields in packet headers, such as the PUSH FLAG bit, for faster network data processing on IoT devices [14].

Furthermore, we can observe in Figure 10 that increasing the number of encoded packets from three to five boosts the system's accuracy by 3%. However, a further increase in the number of packets has no effect on the classifier's accuracy. Additionally, the encoding scheme IT_3_with_header does not improve the accuracy when compared to packets_3_with_header. The rationale behind this could be that the weights of the SGAN classifier have learned the best possible parameters from the raw encoded packet data. The additional meta-data information (inter-arrival time) has no impact on increasing the classifier's overall feature space representation. However, we remark that a better encoding design, such as the placement and representation of inter-arrival times inside the flow encoding matrix, can increase system accuracy. Yet, determining the best system encoding strategy for *iKnight* is beyond the scope of this paper and will be studied in the future.

B. UNKNOWN DEVICE TYPE AND ANOMALY DETECTION PERFORMANCE

In this section, we evaluate the *iKnight* performance in identifying unknown devices as either IoT, non-IoT, or malware, as well as detecting anomalies.

In our experiments to evaluate the *iKnight* stage #2 classifier, we randomly selected five IoT devices, three non-IoT devices, and four malware types as unknown devices. For each of the unknown devices, we first filter out the corresponding flows from the training data that is used in training the stage #2 classifier (and the stage #1 classifier too), and then we run the filtered out flows of the unknown device into the trained stage #2 classifier after it fails to be recognized by the stage #1 classifier, as discussed in Section IV. We repeat this process for all the selected devices.

Figure 12 shows the stage #2 classification accuracy for various unknown devices using packets_5_with_header encoding scheme. Note that we only show the results of two IoT devices, while the other three devices show very similar performance. The accuracy in identifying IoT and malware unknown devices is higher than that of non-IoT unknown devices. We believe this could be due to the small number of non-IoT devices in the training data, where a better representation of each device type class in the training data can further improve the classification accuracy of the stage #2 classifier. The average accuracy for each of the device type classes for the unknown scenario is IoT device type at 94%, non-IoT at 80%, and malware at 97%. The overall unknown accuracy for different classes of device types is 90%.

Figure 13 shows the accuracy of the stage #2 classifier in identifying one of the unknown devices (i.e., the DropCam IoT device) as an IoT device under both packets_5_with_header and packets_5_with_payload encoding schemes. As shown, switching the encoding scheme

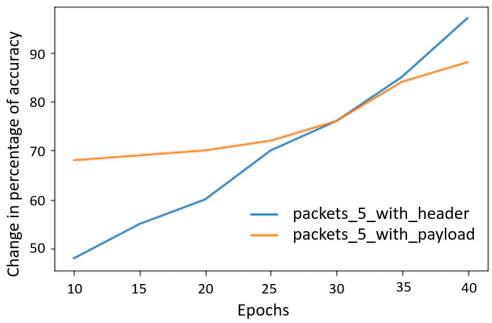

FIGURE 13. The accuracy of *iKnight* stage #2 classifier in classifying the unknown device DropCam as IoT device.

from packets_5_with_payload to packets_5_with_header improved classification accuracy by 9 percent, reaching 97 percent. This shows that *iKnight* can effectively capture the hidden feature space distribution for each of the IoT, non-IoT, and malware known device types. Moreover, this enables *iKnight* to have a better-generalized feature space representation of each device type class, which improves its ability to identify the type of unknown devices. Therefore, *iKnight* is capable of identifying the class types of unknown devices and known devices.

Semi-supervised GANs have recently shown good performance in identifying unknown classes as anomaly [37], also known as novelty detection. This SGAN capability helps the Stage #1 classifier in identifying any device that is not part of the training data as unknown, and also benefits the Stage #2 classifier in identifying the type of unknown device as IoT, non-IoT, malware, or an anomaly (i.e., novelty detection) as shown in the Device Discovery Engine in Figure 3. As future work, we are planning to run more experiments to evaluate SGAN's capability in anomaly detection.

C. PERFORMANCE COMPARISON TO OTHER CLASSIFIERS

One of the advantages of SGAN, which we utilize in building the *iKnight* classifiers, is its ability to achieve high accuracy compared to supervised machine learning algorithms with few labeled data. Therefore, we compare *iKnight* with the most popular classification algorithms used for IoT networks. We compare the SGAN classifier of *iKnight* stage #1 classifier using the packets_5_with_payload encoding scheme with both supervised and unsupervised classifiers, as shown in Table 2. Each supervised classifier was trained with 3% labeled data, while each unsupervised classifier was trained with all the unlabeled data. Finally, the SGAN classifier was trained with both the 3% labeled and the unlabeled data. We use the same packets_5_with_payload encoding scheme for all the classifiers. Table 2 shows that SGAN outperforms all other supervised classification methods, despite being trained with the same number of labeled flows.

Moreover, SGAN also exceeds the unsupervised K-means method, even when both were trained using the same number of unlabeled training samples. While SGAN extensively outperforms all other methods, we have around a 2% increase in accuracy over CNN. We believe that a better selection of encoding techniques can help SGAN extract the distribution of the training data even more effectively. A combination of flow-level metadata and flow-byte stream encoding, for example, can provide SGAN models with a lot of information during training and, consequently, build a better latent space representing the training data. In Table 3 we present multiple SGAN classification metrics such as precision, recall, and F1 score. Furthermore, accuracy is not the only metric to assess SGAN's potential for IoT classification, as SGAN outperforms CNN in many other scenarios, including noise and continual learning tasks, which we will discuss next.

TABLE 2. The accuracy of *iKnight* stage #1 classifier using packets_5_with_payload encoding scheme across twenty IoT devices shown in Table 1.

Classification Method	Accuracy
K-means	0.01
K-means + PCA	0.03
KNN	0.85
Decision Tree	0.86
Random Forest	0.88
CNN	0.90
SGAN	0.92

TABLE 3. Performance metrics of *iKnight* stage #1 classifier using packets_5_with_payload encoding scheme across twenty IoT devices shown in Table 3.

Metric	Performance
Precision	0.92
Recall	0.90
F1 Score	0.90

D. ROBUSTNESS TO NOISE IN IoT DEVICE TRAFFIC

This section evaluates the performance of *iKnight* under two noise-based scenarios: changes in payload and changes in packet sizes. For the packet payload modification scenario, we choose random positions for each packet's payload in a flow and replace them with random hex values. For the change in packet size scenarios, we append random hex values at the end of the packets. For both scenarios, we introduce the noise only in the testing data.

Figure 14 shows the performance of the stage #1 classifier for the changes in payload scenario. As shown, SGAN_packets_5_with_payload is more robust to change in payload-based noise than CNN_packets_5_with_payload. Therefore SGAN is a better choice for building robust classification models for IoT classification tasks with less labeled data. Moreover, Figure 14 SGAN_packets_5_with_header has only a small change in accuracy (i.e., 10%) for a noise change of 40% of the payload. During training of stage 1, the classifier has additional noise training data from the SGAN generator, and therefore *iKnight* is robust to reasonable changes in payloads of the network traffic. However, for

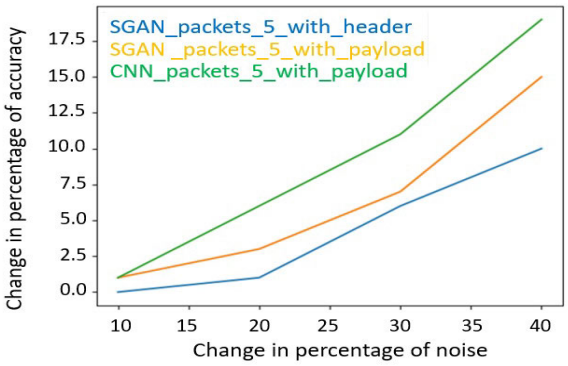

FIGURE 14. The change in *iKnight*'s accuracy of stage 1 classifier with different noise levels.

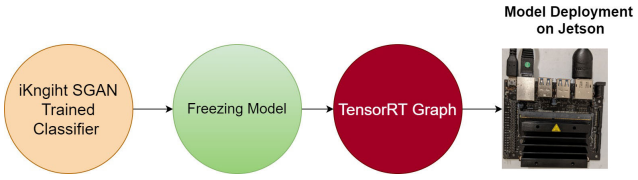

FIGURE 15. *iKnight* deployment on Edge Hardware, NVIDIA Jetson Nano.

noise up to 40% with SGAN_packets_5_with_payload, there is a decrease of 16% in accuracy. Therefore, the use of network headers in the encoding scheme is a more reliable approach in building robust IoT classifiers. In our experiments, we observe an increase in the payload's size by 40% for the SGAN_packets_5_with_header, the accuracy decreases by 20%, and for SGAN_packets_5_with_payload the accuracy decreases by 24%. In real-world scenarios, such large changes to the payload are rare since the already deployed IoT devices' operational requirements generally have small changes in the API methods that introduce small changes to the payload.

E. CONTINUAL LEARNING FOR NEW CLASSIFICATION TASKS

In this section, we evaluate the continual learning abilities of *iKnight*. As described in section VI, we establish different training scenarios in which the classifier is incrementally trained on the Task #1 and Task #2 data in each scenario. The data in Task #1 represents four IoT device classes, whereas Task #2 represents four more IoT devices. We then perform continual learning on the SGAN model for three different scenarios. In the first scenario, *without-old-tasks-data*, the classifier is first trained on Task #1 and then incrementally trained on Task #2 without using any of the Task #1 data. In scenario 2, *with-old-task-data*, the classifier is incrementally trained on Task #2 using data from both Task #1 and Task #2. Finally, in scenario 3, *with-generator-replay*, the classifier is incrementally trained on Task #2 using the Task #2 data and synthetically generated data from the Task #1 SGAN generator, as described in Algorithm 1.

From Table 4, we can observe that for *without-old-tasks-data* scenario, both CNN and SGAN models experience a significant accuracy degradation from 100% to 0% when incrementally trained on the Task #2 data. The SGAN, on the other hand, performs admirably when the scenario is set to *with-old-task-data* because the model learns to generalize the features for all device classes across Tasks #1 and #2 by being trained on ground-truth data for both tasks. However, when the scenario is changed to *with-generator-replay*, the accuracy reduces to 91% since the generated samples do not exactly match the actual Task #1 data. Nevertheless, the Task #1 generator synthesized samples are a good approximation of the Task #1 ground truth data, allowing the classifier to attain extremely high accuracy in comparison to *without-old-tasks data* scenario. As a result, a continual learning technique based on a semi-supervised GAN generator may be well suited to IoT-based continual learning network challenges.

F. DEPLOYMENT ON EDGE HARDWARE

As described previously in Section VII, we apply a knowledge distillation strategy to optimize *iKnight* for deployments at the edge. In our studies, the accuracy of the student model with a single convolution layer was 91%, representing a slight decrease in accuracy compared to the accuracy of the teacher model (stage #1 model with packets_<X>_with_payload encoding scheme), which was 92%. The ability to achieve higher accuracy with incredibly small models can have a significant impact on the performance of edge technology, which we will address next.

Recently, different hardware platforms such as the Raspberry PI [60], Google TPU board [61], Intel neural compute stick [62] and NVIDIA Jetson Nano [63] are being manufactured to support edge-based deep learning-based applications. We used the NVIDIA Jetson Nano [63] to test the deployment of *iKnight* on edge hardware given it has exhibited exceptional performance in deep-learning applications [64]. We then install Tensor Flow and other Python dependencies [65] on Nano, and convert the student and teacher models into a Tensor Graph using the Tensor RT library [66], as illustrated in Figure 15. The Tensor RT graph version of the student and teacher models is the model version that has been optimized for the Nano hardware architecture. We then use the Jetson Stats tool [67] to run both models individually to predict 2000 samples each and calculate the average performance parameter.

Table 5 demonstrates that the Nano student model outperforms the Nano teacher model across all performance parameters. The student model's memory footprint increases by 41% compared to when the hardware was in an idle state, which is much less than the teacher model's memory footprint rise of 81%. Similarly, whereas the student model only increased power usage by 28%, the teacher model increased it by 147% when compared to the idle state. Finally, the average inference time for the student model is 46% lower than the teacher model. As a result, a student model can

TABLE 4. Accuracy across continual learning tasks for IoT based classifier using SGAN and CNN models.

Model	Method	Training Task	# Classes in Training data	Task 1	Task 2
CNN	without-old-tasks-data	T1	4 (T1)	100	-
SGAN	without-old-tasks-data	T1	4 (T1)	100	-
CNN	without-old-tasks-data	T2	4 (T2)	0	94
SGAN	without-old-tasks-data	T2	4 (T2)	0	97
SGAN	with-old-task-data	T2	4 (T2) + 4 (T1)	100	97
SGAN	with generator	T2	4 (T1) + 4 (T2 -Generated)	90	97

TABLE 5. Average performance of *iKnight* stage 1 student and teacher models deployed on NVIDIA Jetson nano hardware.

Model	Accuracy	Change in % Memory (MB)	Change in % Power consumption (mw)	Inference Time (ms)
Teacher Tensor RT	0.92	79.93	146.97	0.0116
Student Tensor RT	0.91	41.4	28.46	0.0062

achieve high-performance goals for edge-based devices with reduced accuracy loss.

Based on these findings, we can conclude that employing a semi-supervised GAN to train the models and then using SGAN models to transfer the knowledge to the lightweight version for edge deployments complement each other well.

X. CONCLUSION AND DISCUSSION

In this paper, we evaluated and implemented the features of an SGAN-based IoT classification system. Based on the findings, we can conclude that a semi-supervised GAN technique has various advantages for designing an IoT classifier. Moreover, with our multi-class multi-stage classification approach, *iKnight* has more potential to identify both known and unknown devicess. Finally, we demonstrate how to offer smart network services at the edge using *iKnight*, such as *iPrivacy*. We were also able to demonstrate *iKnight*'s capacity to learn IoT classification tasks on a continuous basis as new devices are added or firmware is upgraded. However, one of the difficulties we confront in continual learning tasks is maintaining the quality of the generated samples over time. As the number of tasks grows, the SGAN must improve its ability to generate samples that represent the old ground truth classes data. This, however, is an ongoing research challenge in and of itself. We intend to strengthen *iKnight*'s continual learning skills by boosting the generator [68]. In addition, we intend to apply various combinations of continual learning, such as elastic weight consolidation [46] and generative methods [47], on GAN-based classifiers to aid *iKnight* in classification and novelty detection tasks.

Moreover, as discussed in our evaluation section, training *iKnight* with datasets encompassing a broader range of device type classes can improve the reliability and precision of *iKnight's* support for diverse network services. Increasing the number of samples from various device categories across all device classes, for example, will reinforce the classifier's decision boundary since the neural network will have more features extracted for classification and will be able to generalize the properties of each class type better. Furthermore, many non-IoT devices may have large packet sizes, causing devices with smaller packet sizes to be automatically classified as IoT devices. As a result, supplying data points from a variety of non-IoT devices with varying packet sizes can aid in reducing misclassification. In certain cases, the classifier's accuracy can also be improved separately for each class without influencing the performance of other categories by incorporating highly distinguishing properties of each class type, such as the short burst flow of IoT devices, which are notably different from non-IoT devices. Our approach also includes characteristics such as continuous learning, which requires the classifier to learn new data classes on a regular basis. Using samples from a variety of class categories will reveal the classifier's previous performance and help to forget less generalized attributes for specific IoT and non-IoT classes. The classifier, for example, notices that IoT devices have bursts of short packet size and a variety of payload types. However, the classifier may have forgotten some of this information with incremental task training. A larger number of samples, on the other hand, will force the classifier to minimize information loss.

Given these observations, in our future work, we plan to extend *iKnight* to larger device types and evaluate the efficiency of *iKnight* to support different kinds of network services, such as anomaly detection services, in real-time. Finally, we would like to evaluate some of the new compression techniques for the efficient deployment of GAN-based models for edge devices [69].

REFERENCES

[1] *The Growth in Connected IoT Devices is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast*. Accessed: Jun. 25, 2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS45213219

[2] *Smart Home-Statistics Facts*. Accessed: Jun. 25, 2020. [Online]. Available: https://www.statista.com/topics/2430/smart-homes/

[3] Y. Amar, H. Haddadi, R. Mortier, A. Brown, J. Colley, and A. Crabtree, "An analysis of home IoT network traffic and behaviour," 2018, *arXiv:1803.05368*.

[4] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted IoT traffic," 2017, *arXiv:1708.05044*.

[5] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards SDN-defined programmable BYOD (bring your own device) security," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–15.

[6] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated device-type identification for security enforcement in IoT," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 2177–2184.

[7] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for Internet of Things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[9] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," 2016, *arXiv:1606.03498*.

[10] A. Odena, "Semi-supervised learning with generative adversarial networks," 2016, *arXiv:1606.01583*.

[11] G. Stanton and A. A. Irissappane, "GANs for semi-supervised opinion spam detection," 2019, *arXiv:1903.08289*.

[12] B. Lecouat, K. Chang, C.-S. Foo, B. Unnikrishnan, J. M. Brown, H. Zenati, A. Beers, V. Chandrasekhar, J. Kalpathy-Cramer, and P. Krishnaswamy, "Semi-supervised deep learning for abnormality classification in retinal images," 2018, *arXiv:1812.07832*.

[13] F. H. K. D. S. Tanaka and C. Aranha, "Data augmentation using GANs," 2019, *arXiv:1904.09135*.

[14] T. O. Connor, W. Enck, and B. Reaves, "Blinded and confused: Uncovering systemic flaws in device telemetry for smart-home Internet of Things," in *Proc. 12th Conf. Secur. Privacy Wireless Mobile Netw.*, May 2019, pp. 140–150.

[15] M. Montoya, S. Bacles-Min, A. Molnos, and J. J. A. Fournier, "SWARD: A secure WAke-up RaDio against denial-of-service on IoT devices," in *Proc. 11th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jun. 2018, pp. 190–195.

[16] Shodan. Accessed: Jun. 25, 2020. [Online]. Available: https://www.shodan.io/

[17] M. Uddin, T. Nadeem, and S. Nukavarapu, "Extreme SDN framework for IoT and mobile applications flexible privacy at the edge," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom*, Mar. 2019, pp. 1–11.

[18] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2020, pp. 207–218.

[19] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2017, pp. 0588–0592.

[20] N. Jmour, S. Zayen, and A. Abdelkrim, "Convolutional neural networks for image classification," in *Proc. Int. Conf. Adv. Syst. Electric Technol.*, 2018, pp. 397–402.

[21] R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and J. M. Z. Maningo, "Object detection using convolutional neural networks," in *Proc. TENCON IEEE Region Conf.*, Oct. 2018, pp. 2023–2027.

[22] F. Di Mattia, P. Galeone, M. De Simoni, and E. Ghelfi, "A survey on GANs for anomaly detection," 2019, *arXiv:1906.11632*.

[23] N. N. Pise and P. Kulkarni, "A survey of semi-supervised learning methods," in *Proc. Int. Conf. Comput. Intell. Secur.*, Dec. 2008, pp. 30–34.

[24] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "IoTSense: Behavioral fingerprinting of IoT devices," 2018, *arXiv:1804.03852*.

[25] S. Dong, Z. Li, D. Tang, J. Chen, M. Sun, and K. Zhang, "Your smart home can't keep a secret: Towards automated fingerprinting of IoT traffic," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 47–59.

[26] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019.

[27] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Inferring IoT device types from network behavior using unsupervised clustering," in *Proc. IEEE 44th Conf. Local Comput. Netw. (LCN)*, Oct. 2019, pp. 230–233.

[28] J. Ortiz, C. Crawford, and F. Le, "DeviceMien: Network device behavior modeling for identifying unknown IoT devices," in *Proc. Int. Conf. Internet Things Design Implement.*, Apr. 2019, pp. 106–117.

[29] A. Cheng, "PAC-GAN: Packet generation of network traffic using generative adversarial networks," in *Proc. IEEE 10th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Oct. 2019, pp. 0728–0734.

[30] B. Dowoo, Y. Jung, and C. Choi, "PcapGAN: Packet capture file generator by style-based generative adversarial networks," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 1149–1154.

[31] S. Nukavarapu, M. Ayyat, and T. Nadeem, "MirageNet-towards a GAN-based framework for synthetic network traffic generation," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Rio de Janeiro, Brazil, Dec. 2022.

[32] M. Ayyat, S. Nukavarapu, and T. Nadeem, "Dynamic deep neural network adversarial attacks for edge-based IoT devices," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Rio de Janeiro, Brazil, Dec. 2022.

[33] I. Ullah and Q. H. Mahmoud, "A framework for anomaly detection in IoT networks using conditional generative adversarial networks," *IEEE Access*, vol. 9, pp. 165907–165931, 2021.

[34] A. S. Iliyasu and H. Deng, "Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks," *IEEE Access*, vol. 8, pp. 118–126, 2020.

[35] S. K. Nukavarapu and T. Nadeem, "Securing edge-based IoT networks with semi-supervised GANs," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Affiliated Events (PerCom Workshops)*, Mar. 2021, pp. 579–584.

[36] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 267–279.

[37] M. Kliger and S. Fleishman, "Novelty detection with GAN," 2018, *arXiv:1802.10560*.

[38] K. Trapeznikov, V. Saligrama, and D. Castanon, "Multi-stage classifier design," in *Proc. Asian Conf. Mach. Learn.*, 2012, pp. 459–474.

[39] A. Parmisano, S. Garcia, and M. J. Erquiaga, "Aposemat IoT-23: A labeled dataset with malicious and benign IoT network traffic," Stratosphere Lab.-Data set, Jan. 2020. Accessed: Oct. 2022. [Online]. Available: https://www.stratosphereips.org/datasets-iot23

[40] *Pkt2flow*. Accessed: Jun. 25, 2020. [Online]. Available: https://github.com/caesar0301/pkt2flow

[41] *Scapy*. Accessed: Jun. 25, 2020. [Online]. Available: https://scapy.net/

[42] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*.

[43] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015, *arXiv:1511.06434*.

[44] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, Dec. 2014, pp. 1–15.

[45] G. M. Van De Ven and A. S. Tolias, "Three scenarios for continual learning," 2019, *arXiv:1904.07734*.

[46] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, and K. Milan, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, 2017.

[47] C. Wu, L. Herranz, X. Liu, J. Van De Weijer, and B. Raducanu, "Memory replay GANs: Learning to generate new categories without forgetting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5962–5972.

[48] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2642–2651.

[49] I. B. Mustafa, T. Nadeem, and E. Halepovic, "FlexStream: Towards flexible adaptive video streaming on end devices using extreme SDN," in *Proc. 26th ACM Int. Conf. Multimedia*, Oct. 2018, pp. 555–563.

[50] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem, "MESDN: Mobile extension of SDN," in *Proc. 5th Int. Workshop Mobile Cloud Comput. Services*, 2014, pp. 7–14.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[53] *The cifar-10 dataset*. Accessed: Jun. 25, 2020. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html/

[54] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[55] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," 2020, *arXiv:2006.05525*.

[56] M. Uddin and T. Nadeem, "TrafficVision: A case for pushing software defined networks to wireless edges," in *Proc. IEEE 13th Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Oct. 2016, pp. 37–46.
[57] *Open vs Witch*. Accessed: Jun. 25, 2020. [Online]. Available: https://www.openvswitch.org/
[58] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.
[59] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras
[60] *Raspberry PI*. Accessed: Feb. 16, 2021. [Online]. Available: https://www.raspberrypi.org/
[61] *Edge TPU*. Accessed: Feb. 16, 2021. [Online]. Available: https://cloud.google.com/edge-tpu
[62] *Intel Neural Compute Stick 2*. Accessed: Feb. 16, 2021. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html
[63] *Jetson Nano Developer Kit*. Accessed: Dec. 29, 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[64] *Jetson Nano: Deep Learning Inference Benchmarks*. Accessed: Dec. 29, 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks
[65] *Installing Tensorflow for Jetson Platform*. Accessed: Dec. 29, 2020. [Online]. Available https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html
[66] *Nvidia Tensorrt*. Accessed: Dec. 29, 2020. [Online]. Available: https://developer.nvidia.com/tensorrt
[67] *Jetson Stats*. Accessed: Dec. 29, 2020. [Online]. Available: https://pypi.org/project/jetson-stats/
[68] C. Li, K. Xu, J. Zhu, and B. Zhang, "Triple generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 4088–4098.
[69] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "GAN compression: Efficient architectures for interactive conditional GANs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 5284–5294.

SANTOSH NUKAVARAPU received the master's degree in computer science from Old Dominion University and the doctoral degree from Virginia Commonwealth University. His current research interests include adversarial machine learning, generative adversarial networks, and network security.

TAMER NADEEM received the M.Sc. and Ph.D. degrees in computer science from the University of Maryland, College Park. He is an Associate Professor at the Department of Computer Science, Virginia Commonwealth University (VCU). He is also the Assistant Director of the VCU Cybersecurity Center and the Founder of the Mobile Systems and Intelligent Communication (MuSIC) Laboratory, Department of Computer Science, VCU. Prior to VCU, he was a Senior Research Scientist at Siemens Corporate Research (SCR), Princeton, NJ, USA. He holds six U.S. patents and has more than 100 publications in peer-reviewed top scholarly journals and conference proceedings. His research interests cover several aspects of wireless networking and mobile computing systems, including smart wireless systems, mobile and edge computing, software-defined networks, machine learning for network systems, network security and privacy, the Internet of Things, smart city systems, vehicular networks, and intelligent transportation systems. He serves as a member for the technical and organizing committees of various ACM and IEEE conferences. He is currently serving on the Journal Editorial Board for *IET Communications* journal and MDPI *Sensors*.