Toward Synthetic Network Traffic Generating in NTN-Enabled IoT: A Generative AI Approach

Dingde Jiang, *Member, IEEE*, Zhihao Wang, *Student Member, IEEE*, Xinhui Liu, Qi Xu, Tao Zou, Ruyun Zhang, Lizhuang Tan, and Peiying Zhang, *Member, IEEE*

***Abstract*—Nonterrestrial networks (NTNs) enabled Internet of Things (IoT) extends connectivity to remote and underserved areas, enhances network reliability and coverage, and supports diverse IoT applications in challenging environments, such as rural, maritime, and disaster-stricken regions. As an emerging and fast-evolving IoT scheme, NTN-enabled IoT requires extensive evaluation to ensure effective deployment in real-world scenarios, such as connectivity, performance, and security evaluation. Since conducting testing in remote and diverse environments is logistically challenging and costly, we propose a generative artificial intelligence (GAI)-based synthetic traffic generation framework that facilitates comprehensive traffic analysis and performance evaluation. The proposed framework employs a GAI model to learn the traffic pattern and generate synthetic traffic from historical data. Our approach includes an embedding-based model for representing network flow attributes and a conditional generative adversarial network (CGAN) for generating traffic flows. Considering both source-destination information and statistical features achieves more comprehensive characterization of traffic flows. Finally, the simulation results demonstrate that the proposed approach can generate high quality traffic that conforms to real data distribution and shows obvious difference between multiple applications.**

***Index Terms*—Generative AI (GAI), Internet of Things (IoT), nonterrestrial networks (NTNs), traffic generation, word embedding.**

Received 27 May 2024; revised 15 August 2024; accepted 23 September 2024. Date of publication 14 October 2024; date of current version 9 January 2025. This work was supported in part by the National Natural Science Foundation of China under Grant U22A2005 and Grant 62471493; in part by the Natural Science Foundation of Shandong Province under Grant ZR2023LZH017, Grant ZR2022LZH015, and Grant 2023QF025; in part by Fund Projects under Grant 2020-JCJQ-ZD-016-11 and Grant 315197107; in part by the Open Fund of Digital Media Art, Key Laboratory of Sichuan Province under Grant 20DMAKL01; and in part by the Open Foundation of Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Qilu University of Technology under Grant 2023ZD010. *(Corresponding author: Qi Xu.)*

Dingde Jiang is with the School of Information and Communication Engineering and the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Chengdu 611731, China, also with the Laboratory of Electromagnetic Space Cognition and Intelligent Control, Beijing 100091, China, and also with the Digital Media Art, Key Laboratory of Sichuan Province, Sichuan Conservatory of Music, Chengdu 610041, China (e-mail: jiangdd@uestc.edu.cn).

Zhihao Wang and Xinhui Liu are with the University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: wangzhihao1998@foxmail.com; liuxinhui2502@gmail.com).

Qi Xu, Tao Zou, and Ruyun Zhang are with Zhejiang Lab, Hangzhou 311100, China (e-mail: xuqi@zhejianglab.org; zout@zhejianglab.org; zhangry@zhejianglab.org).

Lizhuang Tan is with the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center, Qilu University of Technology, Jinan 250013, China, and also with the Shandong Provincial Key Laboratory of Computer Networks, Shandong Fundamental Research Center for Computer Science, Jinan 250013, China (e-mail: tanlzh@sdas.org).

Peiying Zhang is with the Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China, and also with the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center, Qilu University of Technology, Jinan 250013, China (e-mail: zhangpeiying@upc.edu.cn).

Digital Object Identifier 10.1109/JIOT.2024.3468209

I. Introduction

NON-TERRESTRIAL networks (NTN) are revolutionizing the Internet of Things (IoT) by extending connectivity beyond traditional terrestrial infrastructure. NTNs include satellite, airborne, and other space-based communication systems, which provide global coverage and support connectivity in remote or underserved areas where terrestrial IoT networks are impractical or impossible. This enhancement is critical for IoT applications that require reliable communication in diverse and challenging environments, such as maritime, aviation, rural, and disaster-stricken regions. By integrating NTNs with IoT, it is possible to enable continuous, ubiquitous, and resilient connectivity, fostering advancements in sectors like agriculture, environmental monitoring, logistics, and emergency response [1]. This convergence of NTNs and IoT not only expands the reach of IoT networks but also enhances their robustness, flexibility, and overall performance. NTNs provide extensive applications for IoT networks. The most common scenario is to collect the IoT data with non-terrestrial nodes. The unmanned aerial vehicle (UAV) can collect the data from the remote IoT devices distributed over a region. For example, multiple sensors can be deployed around a specific range of farmland, which do not require permanent links to connect to the central server or controller [1]. Another promising and efficient application is to locate the IoT devices taking the advantage of the mobility of UAVs to support location-aware services. Considering the power constrain and security requirements of IoT devices, it is applicable to deploy global positioning system (GPS) inside them [2]. By integrating NTNs with IoT, it is possible to enable continuous, ubiquitous, and resilient connectivity, fostering advancements in sectors like agriculture, environmental monitoring, logistics, and emergency response. This convergence of NTNs and IoT not only expands the reach of IoT networks but also enhances their robustness, flexibility, and overall performance [3].

As an arising and developing network paradigm, NTN-enabled IoT requires extensive and comprehensive evaluation to demonstrate the effectiveness of deploying in real scenarios.

To evaluate the network connectivity, bottleneck, security, it is essential to inject real traffic into IoT networks. Although NTNs extend connectivity of IoT to remote and underserved regions, real-world network testing can be logistically challenging and expensive. Besides, IoT applications in NTNs must perform reliably in various environments, from rural areas to disaster zones [4], making it hard to conduct experimental tests. From the cost point of view, deploying physical network infrastructure for testing in NTN-enabled IoT is costly and resource intensive [5].

Therefore, employing generative AI (GAI)-based traffic generation in NTN-enabled IoT networks enhances network performance, security, and management by providing realistic and diverse traffic simulations. It allows for accurate modeling of IoT behavior, enabling thorough stress and load testing, and precise measurement of latency and throughput. GAI supports training of intrusion detection systems (IDSs) and anomaly detection by generating both normal and attack traffic patterns, thus improving security. It also reduces testing costs, scales to simulate large numbers of devices, and adapts to different network conditions for customized testing. This approach aids in capacity planning, QoS benchmarking, SLA validation, and facilitates testing new technologies and disaster recovery scenarios, ultimately ensuring robust, efficient, and secure IoT network operations. In NTNs, which rely on satellite, airborne, and other space-based communication systems, simulating realistic traffic is beneficial for evaluating how IoT devices and applications will perform under various conditions [6]. Traffic generation helps in understanding bandwidth requirements, latency, and the reliability of the network, ensuring that IoT solutions are robust and efficient [7].

In this article, we focus on the synthetic traffic flow generation for NTN-enabled IoT. By creating synthetic traffic that mirrors real-world patterns, network operators or service provider can identify potential bottlenecks, test new protocols, and optimize network configurations without the need for costly and complex field deployments. This capability is especially crucial for NTNs, which cover vast and often remote areas where traditional network testing would be challenging. Leveraging the powerful capability of GAI to learn complex patterns in network traffic, we introduce a GAI-based synthetic traffic flow generation approach. The main contribution of this article is concluded as follows.

1) We propose a GAI-based synthetic traffic flow generation framework for NTN-enabled IoT, aiming to improve the convenience of traffic analysis and performance evaluation. We use the GAI model to learn the distribution and behavior characteristics of historical traffic data and generate synthetic traffic that conforms to the characteristics of the real network. The framework enables unlimited traffic generation without the need for deploying real network services.
2) Given categorical attributes that machine learning models cannot process or learn, we introduce an embedding-based approach that represents the flow five-tuple with continuous vectors. The proposed embedding model, like Word2Vec in NLP tasks, also learns the similarity between addresses, ports, and protocols that frequently occur together.
3) To generate network traffic flow that fits the distribution and behavior of various applications, we propose a synthetic network traffic generation model based on conditional generative adversarial network (CGAN). We train the CGAN model using the vector representation of flow five-tuple and the statistical features of the network traffic, as well as the traffic type. After being generated, the traffic flow is transformed back from its generated state into its original space, which can be further used to describe the network behavior between IoT devices or users.

The remainder of this article is organized as follows. Section II presents the relate work. Section III presents the traffic flow embedding representation method based on Work2Vec. Section IV provides the overall traffic generation framework based on CGAN. In Section V, we conduct extensive simulations to evaluate the proposed method.

II. Related Work

During the past several years, GAI approaches have been extensively studied and applied in various fields. One of the representative directions is the deep generation models like ChatGPT and Llama, which have just experienced explosive growth all around the world [8]. Among the GAI models, the deep-learning-based GAN has attracted wide attention due to its capability in image, video, and audio generation tasks. The integration of GAI with NTN, IoT and NTN-enabled IoT has extensive application scenarios and shows great potential. Rong and Rutagemwa [9] leveraged the large language models (LLMs) for the intelligent control of NTNs and IoT. LLMs are used to learn the complex relationships between the elements in networks from large-scale data, to realize intelligent control on resource management, handover management, etc. By integrating GAI techniques in IoT, a new paradigm, Generative IoT (GIoT) is proposed in [10]. Levering the historical and real-time sensing data, GIoT is capable to generate realistic and context-aware data, which contributes to realize intelligent system, optimize the resource allocation, and improve the decision-making process. For example, Wen et al. [10] employed the Generative Diffusion Model to generate effective contract data for Internet of Vehicle traffic monitoring. Ghazanfar et al. [11] introduced GAN and variational autoencoder (VAE) to generate the synthetic channel states to optimize the training of actor-critic reinforcement learning model, which then estimates the line-of-sight probabilities of the communication channels between user equipment and satellite in NTN. Besides, the application of GAI can also improve the resource management of UAV-assisted IoT networks [12].

There is also abundant work exploiting the GAN in network data to generate synthetic traffic that can restore real network data distribution and behavior [13]. Lieto et al. [14] studied the network traffic modeling and generation problem in the production-aware industrial network. The authors introduced the conditional VAE (CVAE) model to learn the

conditional distributions of the packet interarrival time (IAT) and packet size to generate realistic production traffic data. Shahid et al. [15] employed an autoencoder and GAN to generate the packet size sequence of the network flow. Nukavarapu et al. [16] presented a GAN-based model to generate DNS synthetic traffic, which focused on packet generation with byte sequences. Iqbal et al. [17] introduced the semantic and network structure knowledge into GAN to incorporate the knowledge and device category for IoT traffic.

GAI-based network traffic generation framework for NTN-enabled IoT networks can be applied to a variety of use cases, enhancing network management, security, and performance. One of the challenges of the NTN channel in IoT network is the large delay, the generated traffic can be used to assess and optimize network latency and throughput in different conditions [7]. NTN-IoT aims to provide reliable, efficient, and consistent communication for IoT devices across diverse and often challenging environments. The synthetic can be used to assess QoS metrics, such as jitter, packet loss, and latency, which helps in maintaining high service quality and meeting the expectations of end-devices [17]. The synthetic traffic resembles realistic attack traffic patterns, such as Distributed Denial of Service (DDoS) attacks or data exfiltration attempts, to train the IDS in NTN-enabled IoT [18], [19]. Testing new or existing communication protocols under diverse traffic conditions usually requires realistic network environments and traffics, but conducting experiments in the operating network may leads to unpredictable issues. Thus, the synthetic traffic is practical for network experiments to ensure they are reliable and efficient in an NTN-enabled IoT environment [20]. Routing [21] and resource allocation [22] optimization relies on modeling different traffic loads and usage patterns, which can be simulated precisely by traffic generation models.

Generating synthetic traffic flow for multiple applications using GAI models involves two main challenges. The first challenge is that GANs cannot directly process categorical attributes, such as the five-tuple and flags [23]. Even though some of these attributes may be represented numerically, their values do not indicate their quantity or size. Inspired by the powerful and effective word embedding approach Word2vec [24], Ring et al. [25] proposed IP2Vec, an unsupervised method to learn the vector representations of network flow five-tuple. The second challenge is that basic GAN models cannot output traffic of a specific category, thus they can only generate traffic randomly based on the input training samples. When dealing with multiple types of traffic, it is necessary to undergo separate training for each type to synthesize the corresponding traffic. However, specifying the category is essential for the analysis of a particular application's traffic. However, most work either focus on the generation of statistical features or raw traffic bytes. Since the AI models cannot directly process the flow five-tuples, the generated statistical features cannot imply the direction or the location of the network traffic, which is essential in many use cases of NTN-enabled IoT. Hence, this article aims to generate network traffic that considers not only the statistical features but also the traffic location, so that we can simulate the communication between IoT devices distributed in different locations.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
s_{emb}	the embedding dimension of Word2Vec model
s_{win}	the context window size for IP2Vec
s_{voc}	the size of vocabulary
V	the set of vocabulary
w	the word in vocabulary
$w(o)$	context word
$w(c)$	central word
$x(t)$	one-hot representation of word $w(t)$
$\mathbb{X}$	the set of training samples
$\mathbb{X}'$	the set of training samples after embedding
$\mathbf{F}^{real}$	flow embedding attributes of real traffic
$\mathbf{F}^{syn}$	flow embedding attributes of synthetic traffic
$\mathbf{X}^{real}$	statistical attributes of real traffic
$\mathbf{X}^{syn}$	statistical attributes of synthetic traffic
c	service type of the traffic (conditional information)
D	discriminator network of CGAN
G	generator network of CGAN

III. FLOW EMBEDDING REPRESENTATION

The representation of vocabulary with vectors and the measurement of similarity between them is well-studied in NLP tasks [24]. One-Hot encoding is an effective approach for converting nonnumerical data into multidimensional vectors. According to the principle of One-Hot, the dimension of a word vector is proportional to the number of words in the corpus. This creates a large representation matrix, making it inconvenient for storage and calculation. If new words are added to the corpus, the word vector of all words will be altered. Simultaneously, the vector contains a high number of zeroes, resulting in the expression of words too sparse. Furthermore, One-Hot encoding fails to capture word relationships. To tackle this issue, word embedding techniques were developed to transform the vocabulary into a continuous, low-dimensional vector representation that draws context from the sentence or phrase, for instance, using Word2Vec [24], FastText [26]. The mapping of vocabularies with similar meanings close together in vector space resolves the curse of dimensionality and the lack of a semantic representation.

The IP addresses, ports, and protocols in the flow five-tuple possess characteristics that are similar to those of vocabularies. First, the fields in the flow five-tuple cannot be processed directly by machine learning techniques. Second, the space of the addresses is too large to be represented by an One-Hot encoder. It contains a vast number of addresses, leading to a dimensional explosion of the representation vector. Hence, Ring et al. [25] first introduced the IP2Vec based on Word2Vec. IP2Vec transforms IP addresses into vectors in

$R^{s_{\text{emb}}}$, where s_{emb} is the embedding dimension. The context is extracted from the flow-based traffic and used to train the neural network model. Being trained with a fake task that predicts the context word with the input word or vice versa, the weight matrixes of the hidden layer are extracted as the embedding vectors of the words.

A. Neural Network Model

The neural network model for IP2Vec used in this article contains an input layer, a single hidden layer, and an output layer, shown in Fig. 1. The frequently used notations in this work are summarized in Table I. The neurons in the hidden layer are fully connected with those in input and output layers. Here we show the state of the network model when two training samples (192.168.130.41, Port 54981), (192.168.130.41, TCP) are inputted. Given the target word 192.168.130.41, the context word are Port 54981 and TCP. Considering that the addresses cannot be input directly into the neural network, every address in vocabulary, namely the IP addresses, the ports and the protocol, is first represented as an $1 \times s_{\text{emb}}$ one-hot vector. The input layer is the target word to be predicted. The output layer is the context word. The input and output layers are connected to the hidden layer with a $s_{\text{voc}} \times s_{\text{emb}}$ and $s_{\text{emb}} \times s_{\text{voc}}$ weight matrices, respectively. The output layer employs a Softmax function to indicate the probability of input and output vocabularies appearing in the same context. The model is trained to capture the context and semantic similarity between words.

B. Training Sample Composition

A traffic flow data set obtained from the physical network typically includes the flow five-tuple and flow statistical attributes, as well as the label, which in this article is the application type. To train the IP2Vec model, we use the flow five-tuple attributes as our corpus. All five fields in the five-tuple are used as input words. Consequently, the vocabulary consists of all IP addresses, ports and protocols present in the data set. At the same time, the IP addresses, ports, and protocols are learned for their similarities.

Defining an appropriate context is necessary to train the neural network mentioned previously. In tasks related to natural language processing, the context can be extracted using a sliding window, which is a specific range of words that surrounds the target word within a sentence. In traffic flow data, semantic relationships exist between nonadjacent fields. For instance, the nonadjacent pair of source IP address and destination IP address is significant in separating different types of flows. It also infers similar information between different addresses. In this article, we consider the flow five-tuple as a sentence. We customize the rules for context extraction based on the relationship between fields instead of adjacency. The original IP2Vec [25] is designed for unidirectional flow-based data and aims to identify hosts with similar behavior. The source port is not selected as a word to be processed. Considering that we process bidirectional flow, we optimize the context extraction rules, shown in Fig. 2, extending the original IP2Vec. The input flow consists of the following attributes: Source IP,

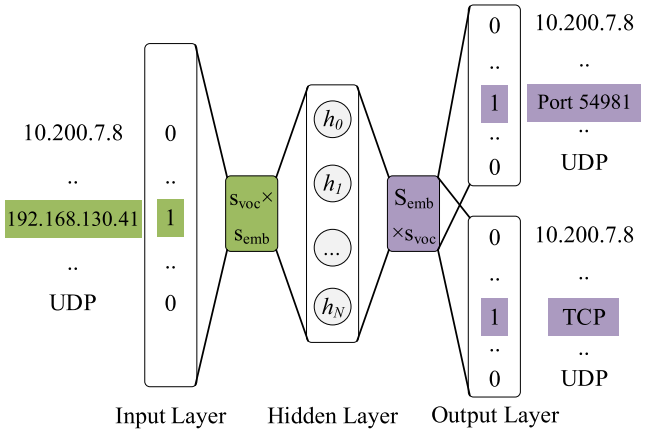

Fig. 1. Neural network structure of IP2Vec embedding.

Source port, Destination IP, Destination port, and Protocol. The top of the figure presents the composition process of the input flows. Each flow generates six training samples, depicted on the left part of the figure. The input word is highlighted with a colored background and the context is represented with a gray color. The bottom part illustrates an example of training sample composition. Each flow generates nine training samples. For each input word of source and destination IP address, three word-context pairs are generated. The remaining three samples are created with ports and protocol as input words. The training sample is first created using the source-destination address pair, along with the destination-source pair. As the ports utilized by applications on the server are frequently static or in a specific range, IP addresses are linked with the ports to generate context. The protocol is associated with both the source and destination IP addresses. Fig. 2 provides an example of the composition of the training samples. The generated samples are encoded by an One-Hot encoder defined with the vocabulary set, for training the neural network model defined in Fig. 1.

C. Model Training and Application

Continuous bag-of-words (CBOWs) and Skip-Gram are two architectures used to train the log-bilinear models in Word2Vec. CBOW learns the word representations by using the context to predict a word, while the skip-gram uses the word representation to predict its context. In this article, the skip-gram architecture is used to train the neural network in Fig. 1 using the training samples composed in Fig. 2. The goal of the skip-gram is to train the network to predict the probability that a randomly selected word in vocabulary is the neighbor, namely context, of a given word. However, the input/output neurons are not used in the classification tasks, only the weights of the hidden layer are extracted as the word vector. Considering the training samples generated by Fig. 2, we set the window size as 2. We train the network by feeding vocabulary V, the One-Hot vector representation of $w(t)$ is the word-context pairs into it. Given a word w from denoted as $x(t) \in R^{s_{\text{voc}}}$. The embedding vector of $x(t)$ is calculated as follows:

$$v(t) = \mathcal{V}x(t), \quad \mathcal{V} \in R^{s_{\text{emb}} \times s_{\text{voc}}} \tag{1}$$

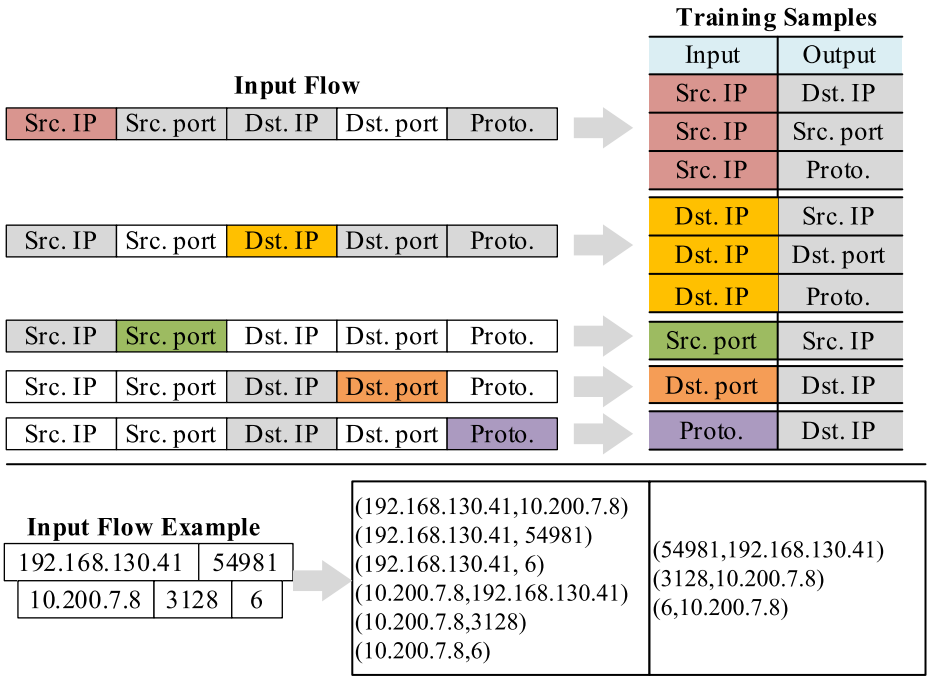

Fig. 2. Training samples composition.

where $\mathcal{V}$ is the input word matrix, s_{emb} is the size of the embedding space. The score vector is calculated by multiplying the output word matrix $\mathcal{U}$ by the $v(t)$, expressed as

$$z(t) = \mathcal{U}v(t), \quad \mathcal{U} \in \mathbf{R}^{|V| \times s^{\text{emb}}}. \tag{2}$$

Then, the probabilities are calculated using Softmax. The predicted probabilities of context words are denoted as

$$\widehat{y}(t - s_{\text{win}}), \ldots, \widehat{y}(t-1), \widehat{y}(t+1), \ldots, \widehat{y}(t + s_{\text{win}}). \tag{3}$$

The One-Hot representation vector of the context word is denoted as (4), which matches the predicted probability vector

$$y(t - s_{\text{win}}), \ldots, y(t-1), y(t+1), \ldots, y(t + s_{\text{win}}). \tag{4}$$

Assuming that the probabilities that using the center word to predict the context words are independent of each other, there will be a conditional probability shown as (5). The goal of the model is to minimize that probability:

$$\min \ J = -\frac{1}{T} \prod_{t=1}^{T} \prod_{-s_{\text{win}} \le j \le s_{\text{win}} j \ne 0} \log \mathbb{P}(w(t+j)|w(t)) \tag{5}$$

where T is the length of the sentence, namely the length of five-tuples in this article. And the probability that computes the context word $w(o)$ using the given central word $w(c)$ is expressed as:

$$P(w(o)|w(c)) = \frac{\exp\left(u_o^{\top} v_c\right)}{\sum_{i \in \mathcal{V}} \exp\left(u_i^{\top} v_c\right)} \tag{6}$$

where v_c is the word vector of the central word, namely the cth column vector of $\mathcal{V}$, and u_o is the word vector of the context word, namely the oth column vector of $\mathcal{U}$. The computing cost of $\nabla \log P(w(o)|w(c))$ is proportional to the size of vocabulary [27]. We use Adam as the optimization function rather than stochastic gradient descent (SGD) in the original Word2Vec to realize faster and more stable convergence. Because the vocabulary size is usually large, directly using this target to train is impractical. Hierarchical Softmax and Negative Sampling [27] are proposed to reduce the complexity and improve training efficiency. Here we take the Negative Sampling to optimize the IP2Vec model.

D. Negative Sampling

As explained earlier, the input layer, hidden layer, and output layer are interconnected. Assuming that the size of the vocabulary is s_{voc}, it also indicates the size of the input and output layer. The size of weight matrix between input and hidden layer is $[s_{\text{voc}}, s_{\text{emb}}]$. The size of the weight matrix between hidden and output layer is $[s_{\text{emb}}, s_{\text{voc}}]$. Since there are usually enormous words in vocabulary, the s_{voc} is quite large. Consequently, the size of weight matrix is also large, leading to high training costs. The cost of computing Softmax values is also considerable. Moreover, large training corpus is necessary to fine-tune the parameters and avoid overfitting. When training the neural network model, only the weights corresponding to the target word need to be updated significantly for each training sample.

During each backpropagation, all the weights in the neural network are updated, leading to highly inefficient operations. Therefore, we employ the Negative Sampling to optimize the training of embedding model. Negative Sampling allows the model to update only a small subset of weights rather than the whole weight matrix. Given a group of training samples, the algorithm randomly selects a small amount of the negative samples to update their weights. Negative samples refer to inputs for which the network outputs 0. After employing

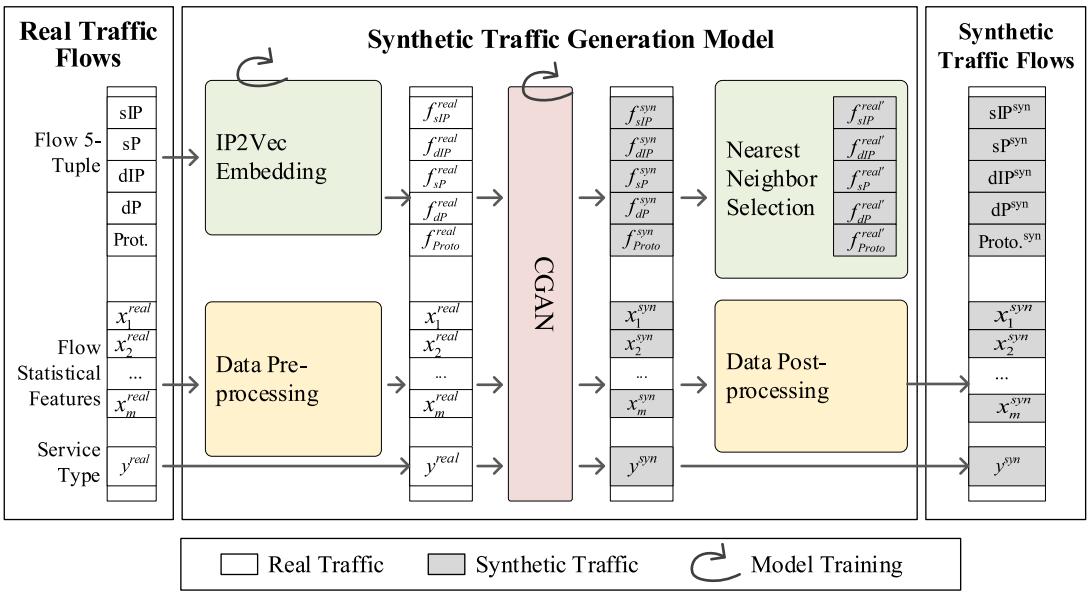

Fig. 3. Overall framework of the proposed GAI-based synthetic traffic flow generation approach.

negative sampling, the objective $\log P(w(o)|w(c))$ is replaced by

$$\log \sigma\left(u_o^\top v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}\left[\log \sigma\left(-u_i^\top v_c\right)\right]. \tag{7}$$

Then, the task for each target word $w(c)$ is transformed to distinguish the context word $w(o)$ from k negative samples drawn from the noise distribution $P_n(w)$ using logistic regression. The loss of target word is computed as the classification problem between the context word $w(o)$ and k negative samples.

Negative samples are selected based on a unigram distribution, where high-frequency words have a high probability of being selected and low-frequency words have a low probability of being selected. Mikolov et al. [27] founded that the best distribution is the unigram distribution $U(w)$ with 3/4 power, expressed as

$$P_n(w) = \frac{U(w)^{3/4}}{Z} = \frac{f(w)^{3/4}}{\sum_{j=0}^{n} f(w)^{3/4}} \tag{8}$$

where $f(w)$ is the occurrence time of word w, and $P_n(w)$ is the probability of selecting word w as a negative sample. Taking the training sample in Fig. 2 as an example, regarding the target-context pair ("192.168.130.41," "10.200.7.8"), the output probability is expected to be 1 when the input is 10.200.7.8 and to be 0 otherwise. The problem with the original skip-gram model is that it involves a four-class classification task to maximize the probability of P("10.200.7.8"|"192.168.130.41") and minimize the others. During backpropagation, the word vectors of "54981," "3128," and "6" are updated, which brings $5 \times s_{\text{emb}}$ times of calculation. In negative sampling, a single word from (54981, 3128, 6) is randomly selected to determine the probability of its neighboring words. This selection reduces the computing times from $5 \times s_{\text{emb}}$ to $3 \times s_{\text{emb}}$ because it only requires updating the weight of 192.168.130.41, 10.200.7.8, and the selected negative word. The training cost is reduced significantly through the aforementioned negative sampling.

IV. Synthetic Traffic Flow Generation Framework

A. Overall Framework

This section provides a detailed presentation of the synthetic traffic flow generation approach based on GAI. Fig. 3 illustrates the sequential process of collecting traffic from the physical network, training the CGAN network, and generating samples of traffic patterns that align with different application types in the NTN-enabled IoT networks. sIP and dIP represent the source and destination IP addresses of the flow, while sP and dP are the respective source and destination ports. Proto. refers to the protocol of the flow. $\mathbf{F}^{\text{real}}$ and $\mathbf{F}^{\text{syn}}$, in the size of $1 \times 5s_{\text{emb}}$, denote the stacked embeddings of the flow's five-tuple, namely $f_{\text{sIP}}^{\text{real}}$, $f_{\text{dIP}}^{\text{real}}$, etc. $f_{\text{sIP}}^{\text{real}}$ represents the embedding vector of source IP address, in the size of $1 \times s_{\text{emb}}$. $f_{\text{sIP}}^{\text{real}'}$ represents the nearest neighbor of $f_{\text{sIP}}^{\text{syn}}$. $\mathbf{X}^{\text{real}}$ denotes the feature matrix of the network flow, and $\mathbf{X}^{\text{syn}}$ represents the corresponding feature matrix in synthetic traffic flow. x_i^{real} represents the ith feature of $\mathbf{X}^{\text{real}}$, so on so forth. c denotes the application type of the flow.

Here, we discuss the overall framework and the crucial components of the approach, which includes traffic flow five-tuple embedding representation using IP2Vec, and traffic flow generation for multiple applications with CGAN. We explain the process of constructing context and corpus for training the IP2Vec model and training a GAN model with labeled data. The proposed approach works based on the structured network flow, which contains the flow identifier, i.e., five-tuple, and the statistical attributes. Thus, it can only generate network flows in an offline mode rather than generating live traffic. The network flows are transformed from the traffic captured from the physical network and then are input to the generation model. The flow five-tuple attributes are embedded as vectors

Algorithm 1 Flow Generation Algorithm

// Train the IP2Vec model
foreach flow f^{real} **in** training dataset $\mathbb{X}$
  compose corpus using $sIP, \ldots, Proto$ based on Fig. 2
  train $\mathcal{V}$ and $\mathcal{U}$ with corpus using SGD
end for
// Generate training dataset for CGAN
initialize new training dataset $\mathbb{X}' \leftarrow \varnothing$
foreach flow f^{real} **in** training dataset $\mathbb{X}'$
  extract the embedding $f^{real}_{sIP}, \ldots, f^{real}_{Proto}$ from $\mathcal{V}$
  $\mathbb{X}' \leftarrow \mathbb{X}' \cup [f^{real}, x^{real}, y^{real}]$
end for
// Training CGAN model
initialize D and G
for every iteration **do**
  update D with a minibatch of real samples from $\mathbb{X}'$, the classes Y and a minibatch of noise samples
  update G with a minibatch of noise samples
end for
// Generate synthetic flows
generate embedded flow $[f^{syn}, x^{syn}, y]$ using G
foreach field f_i^{syn} **in** f^{syn}
  search for the nearest neighbors $f_i^{real'}$ in embeddings
  map the word (IPs, Ports, Protocols) of $f_i^{real'}$ to f_i^{syn}
end for
concatenate $[sIP^{syn}, \ldots, Proto^{syn}, x^{syn}, y]$ as synthetic flows

that can be directly processed and learned by the CGAN model. The rest statistical features of the network flow are also input into the CGAN model after preprocessing. Meanwhile, the traffic application type is also included in the input for CGAN to generate traffic with labels. After training the CGAN model, it can generate traffic for different applications.

After being generated by the CGAN model, the synthetic embedding attributes need to be retransformed to the original space, i.e., the flow five-tuple. To achieve that, we select the nearest neighbor of the synthetic embedding attribute in the existing embedding space. To achieve that, we first calculate all embeddings of vocabularies, using the well-trained IP2Vec model. All correspondence between the vocabularies and the embeddings are stored for further searching. After generating a synthetic flow, we extract the embeddings of the flow five-tuple. Each of them is used to calculate the cosine similarity [23] between it and all embedding results stored before. The word that has the maximum cosine similarity between its trained embedding and the generated synthetic embedding is mapped to the corresponding fields of synthetic flows. Then, the flow five-tuple is transformed back to the address space rather than embedding space. The generated traffic flows can be further exploited in various downstream tasks in NTN-enabled IoT networks.

B. Traffic Flow Generation for Multiple Applications

As an adversarial model, GANs are usually composed of two neural network models counteracting each other, i.e., the generator network G and the discriminator network D. G is expected to extract and describe the distribution of the training samples, and the D is to evaluate the probability that the sample generated by G fits the distribution or original samples. In the synthetic network traffic generation task of this article, G builds a neural network to map the prior noise distribution $p_z(z)$ to the real flow space $G(z, \theta_g)$, and learn a distribution p^g over $\mathbf{X}^{\text{real}}$. G is trained to minimize $\log(1 - D(G(z)))$. D contains a neural network that outputs a value measuring the probability that the sample belongs to real flow rather than p^g. D is trained to minimized $\log(D(x))$ simultaneously with G. The objective function of a GAN model during the training of the generator and discriminator is to minimize a max value, defined as follows:

$$\min_G \max_D \; V(D, G) = \mathbb{E}_{x \sim p_{\text{real}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}\big[\log(1 - D(G(z)))\big]. \tag{9}$$

General GAN models are trained using unlabeled data. As a result, they can only generate samples that are random and unpredictable. The type of synthetic samples cannot be controlled or manipulated to imitate the traffic pattern of different applications in the NTN-enabled IoT. Therefore, Mirza and Osindero [28] proposed the CGAN, which is an extension of the GAN model that feeds a priori information c into both the generator and the discriminator. The information c can be various multimodal, such as class information of the samples or the character action information in the images. If considering the dynamics in training with square loss, it is helpful to obtain the better performance. In this article, c is defined as the application types of the traffic flow, such as Google and Skype. Fig. 4 shows the overall model structure of the CGAN model for generating synthetic network traffic. The generator network takes in a randomly generated noise vector and labels of the training samples to produce samples with labels. The training samples, the generator's output, and their respective labels are input together to the discriminator, whose task is to distinguish between real and generated samples. CGAN introduces conditional information to the generator network in addition to noise z, which are combined in a joint hidden representation. This information is also input to the discriminator network along with $\mathbf{X}^{\text{real}}$. By incorporating the additional conditional information c, the objective function of CGAN can be transformed into a two-player min-max game with conditional probability, denoted as

$$\min_G \max_D \; V(D, G) = \mathbb{E}_{x \sim p_{\text{real}}(x)}[\log D(x|c)] + \mathbb{E}_{z \sim p_z(z)}\big[\log(1 - D(G(z|c)))\big]. \tag{10}$$

The discriminator must determine not only if the input sample conforms $\mathbf{X}^{\text{real}}$ but also if it is consistent with the condition restricted by c. The discriminator only determines that the generated sample is true when the label represented by c matches the sample. After training with c, the generator can create synthetic traffic with specified labels, corresponding to the specified application types in this article. The training approach of CGAN is the same as vanilla GAN (VGAN), except for additional auxiliary information. The discriminator and generator are trained alternately through a two-player game, in which they enhance each other until they reach a Nash equilibrium state.

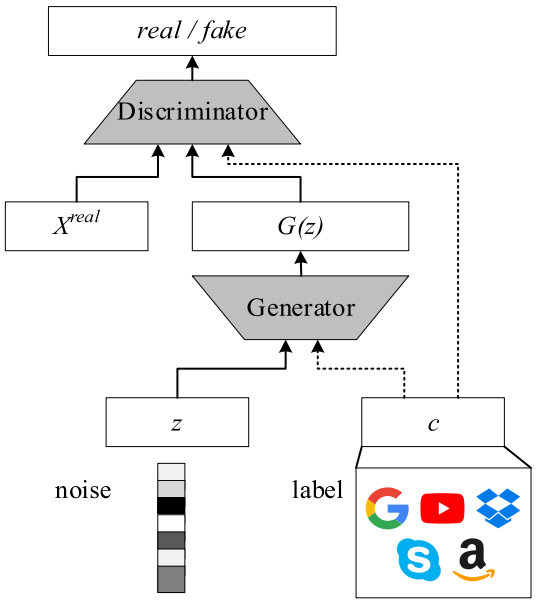

Fig. 4. Model structure of the CGAN.

The overall flow generation algorithm is described in Algorithm 1. There are mainly four stages. First, we use the network flow data set $\mathbb{X}$ to train the IP2Vec model, where only the flow five-tuples f^{real} are used. Second, after the embedding are trained, we compose new training data set $\mathbb{X}'$ for CGAN by concatenating the embedding of each filed in flow five-tuples with the statistical attributes and classes. Then, we use $\mathbb{X}'$ to train the CGAN model, i.e., D and G. After training, G is used to generate specific y type of flows that contains synthetic embeddings f^{syn} and synthetic statistical attributes x^{syn}. To map the embeddings to real flow five-tuples, we search for the nearest neighbor $f_i^{\text{real}'}$ of each field f_i^{syn} in the embedding space and use the corresponding words $\text{sIP}^{\text{syn}}, \ldots, \text{Proto}^{\text{syn}}$ as the synthetic flow five-tuples.

V. Simulation Results and Analysis

A. Evaluation Setup

We evaluate the performance of the IP2Vec-based embedding model and the CGAN-based traffic generation model using an open traffic data set captured in [29]. The data set captures the traffic from 75 network applications during the morning and afternoon over six days. We use the flow descriptor attributes in this data set to train the IP2Vec model to learn the embedding. The statistical attributes are used to train the CGAN model to learn the flow statistics. The vocabulary size of the traffic flow from the data set is 29 662, namely the size of the input layer and output layer. The embedding dimension is set to 128, which means every field in flow five-tuple is transformed into a 128-dim vector. Regarding the statistical attributes, data cleaning is first conducted to dropping the duplicate, incorrect, or incomplete data. Then, the application type labels are encoded into numerical classes, including Google, YouTube, Amazon, Dropbox, and Skype. Then, we conduct the Min-Max scaling to normalize the data. Using the random forest model, we select 20 most important attributes to represent the flow statistics. All experiments are conducted on a PC with Intel Core i9-13900HX, 32GB RAM and NVIDIA GeForce RTX 4060 Laptop GPU. We implement the IP2Vec model and GAN models using Python 3.8.18 and PyTorch 2.2.1. With respect to the IP2Vec model described in Section III, we model the task as a binary classification problem that distinguishes positive from negative contexts, so a binary cross-entropy loss is employed. Adam optimizer is used for the IP2Vec model. The IP2Vec model is trained for 2 epochs with learning rate 0.001. Then, we build a three-hidden-layer network as the generator of CGAN, where LeakyReLU is taken as the activation function of hidden layers and Tanh is taken as the activation function of the output layer. Similarly, a three-hidden-layer network is also used as the discriminator, where LeakyReLU and Sigmod are employed as the activation function of hidden layer and output layer, respectively. Batch normalization is conducted for better training speed and stability. Both generator and discriminator are trained for 50 epochs with learning rate 0.0002.

We allow the IP2Vec and CGAN models to generate a specified number of traffic samples after training them. Three widely used GAI models are introduced for comparison: 1) VAE [30]; 2) VGAN [31]; and 3) wasserstein GAN (WGAN) [32]. Since the mentioned methods cannot generate traffic in a specific category, we train the model separately using traffic from various categories [33]. Many metrics are used to assess the quality of the synthetic network traffic generated. Statistical distribution is an effective and intuitive method of verification, as discussed in [15]. In [23], GAI approaches were evaluated using distribution visualization, distance comparison, and domain knowledge checking to assess their performance. The traffic generated by the GAN model in [16] was assessed for correctness, authenticity, and similarity. We first introduce the t-SNE visualization [25] to demonstrate the embedding model's performance on the similarity measure, using commonly used metrics. We further evaluate the synthetic traffic generated by CGAN using statistical distribution and sample distance measures. The following sections show a detailed breakdown of the experimental outcomes.

B. Flow Embedding Results

To convert categorical attributes that GAN models cannot process into useful information, we propose an approach that employs IP2Vec-based flow five-tuple embedding. Every field in the five-tuple is converted into a 128-D vector. The network behavior of different applications is typically varied, such as the commonly used ports on the server and the transmission protocols. Similarity and differences amongst applications is an effective indicator of the embedding accuracy [25]. Because vector representations have high dimensionality, visualization can only be achieved through dimensionality reduction. High-dimensional data can be reduced through feature selection and feature extraction techniques, which are the main categories for performing this task. Feature selection involves directly selecting a subset of the original dimensions to include in the subsequent calculation and modeling processes while maintaining the most informative features. This can be

achieved using filter methods, wrapper methods, or embedded methods. Feature extraction combines or transforms original features to generate low-dimensional features. The techniques commonly used include principal component analysis (PCA), linear discriminant analysis (LDA), and t-SNE [34], which produce features with reduced dimensions. This article uses t-SNE to transform the high-dimensional embedding features into two dimensions that can be presented intuitively in 2-D figures. The transformed data points exhibit a higher degree of aggregation and smaller distances, which are indicative of their similarity.

Assuming the original distribution of flow five-tuple as P^{emb}, the goal of t-SNE is to find a new distribution of the low-dimension data Q close to P^{emb}. The t-SNE uses Student's t-distribution with a single degree of freedom to create a low-dimensional space, expressed as follows:

$$Q_{ij} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_k \sum_{k \neq l} \left(1 + ||y_k - y_l||^2\right)^{-1}}. \tag{11}$$

KL divergence is introduced to measure the distance between P^{emb} and Q, shown as follows:

$$D_{\text{KL}}\left(Q||P^{\text{emb}}\right) = \sum_{ij} Q_{ij} \log\left(\frac{Q_{ij}}{P_{ij}}\right). \tag{12}$$

One of the most important parameters of t-SNE is the perplexity that captures different scales in the data, defined as

$$\text{perp}\left(P^{\text{emb}}_{j|i}\right) = 2^{H\left(P^{\text{emb}}_{j|i}\right)} \tag{13}$$

where $P^{j|i}_{\text{emb}}$ is the probability that data point x_i select x_j as its neighbor, and $H(P_{\text{emb}})$ is the Shannon entropy calculated as follows:

$$H\left(P^{\text{emb}}\right) = -\sum_i P^{\text{emb}}_i \log\left(P^{\text{emb}}_i\right). \tag{14}$$

We tuned the variable to verify if the embedding model can extract implicit flow patterns between traffic from different applications. Figs. 5–8 shows the t-SNE visualization for the flow five-tuples under different perplexities. Once the neural network model of IP2Vec has been trained and converged, an experiment is conducted to evaluate its embedding performance. In this experiment, the vector representations for the network flow five-tuple of different applications are calculated. The vectors are then reassembled and input into the t-SNE algorithm with assigned perplexity values to obtain a 2-D representation. The dimensionality reduction results of traffic from different applications are compared in pairs to assess if they can be clearly distinguished. Fig. 5 shows that the data points from the flow embedding results of the five applications become relatively scattered after the dimension reduction process. There is a significant overlap between the data points of each application. Distinguishing between various applications through spatial distribution is difficult to do intuitively. As seen in Fig. 6, when the value of perp increases to 150, the internal aggregation level of data distribution for each application also increases, resulting in one to two clusters. However, most data points still have a relatively divergent distribution, with noticeable overlapping between them. As the perp value increases to 175, there are noticeable differences in the distribution of data between applications, except for Google and Amazon, as shown in Fig. 7. Finally, when the perp is set to 200, a clear gap is demonstrated between every two applications, shown in Fig. 8. Furthermore, the distribution between certain pairs of applications, for example, Amazon and Dropbox, Amazon and Google, and Dropbox and Google, can be easily distinguished using a linear regression algorithm. Based on the analysis above, it can be inferred that the IP2Vec-based embedding approach is effective in generating a vector representation of the flow five-tuple in the input traffic. The proposed approach extracts and expresses the similarity within the traffic of a single application and the differences between the traffic of every two applications. However, it is observed that relying solely on flow five-tuple information cannot fully distinguish the traffic type of different applications, thus requiring more refined flow features to be introduced.

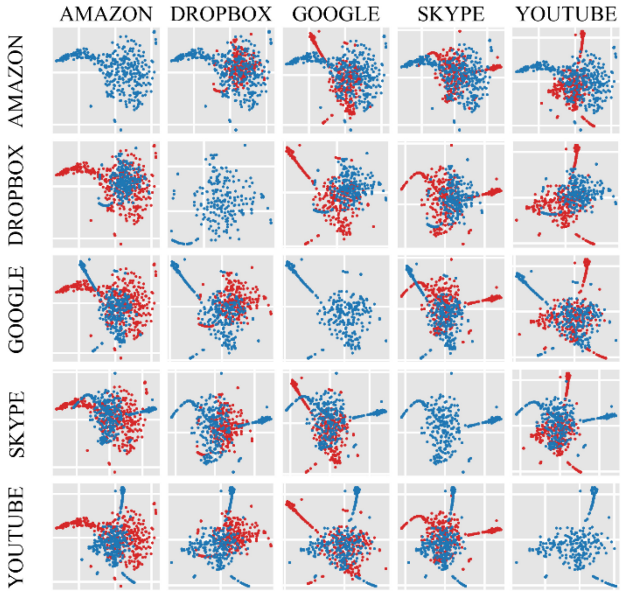

Fig. 5. t-SNE visualization when perp = 125.

C. Statistical Distribution

To assess the quality of the synthetic traffic, we analyze the distribution of various network flow attributes of both synthetically generated and real traffic flows. Figs. 9 and 10 illustrate the distribution of 20 000 real traffic flows and 20 000 synthetic traffic flows generated by VAE, VGAN, WGAN, and CGAN, respectively. We select two statistical flow attributes for analysis: 1) average packet size and 2) flow IAT mean. The attribute Average Packet Size refers to the mean size of each packet in the traffic flow. Flow IAT Mean denotes the mean value of IAT in the flow. Fig. 9(a)–(d) displays the distribution of Amazon flows created by four different approaches. The blue curves and bars display the statistical distribution of the average packet size feature of the real traffic, which was used to train the IP2Vec model and CGAN. The red ones represent the synthetic traffic produced

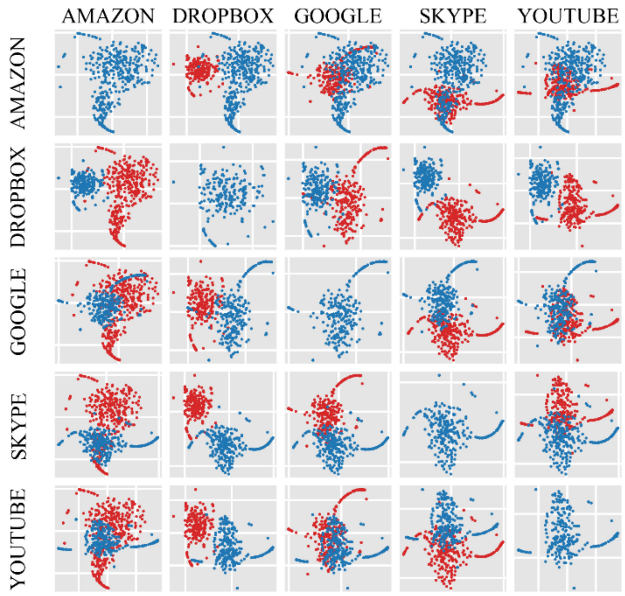

Fig. 6. t-SNE visualization when perp = 150.

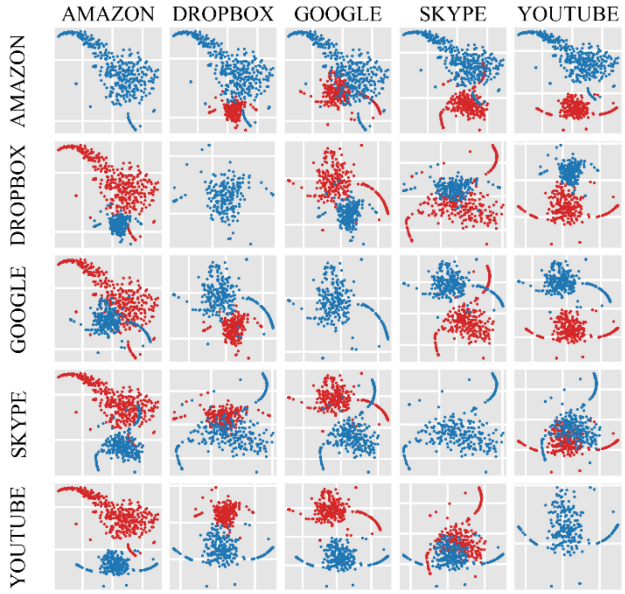

Fig. 7. t-SNE visualization when perp = 175.

by various GAI approaches. Subfigures (a)–(d) illustrate the traffic distribution comparison of Amazon, while (e)–(h) depict the traffic of Dropbox. Notably, VAE and VGAN produced fewer flows with lower average packet sizes compared to the real flows. WGAN generated more samples with an average packet size ranging from 30 bytes to 100 bytes than the real flows. The flows produced by CGAN exhibit the closest trend to the real traffic flow. Fig. 9(e)–(h) displays the distribution of Dropbox flows created by four different approaches. The traffic generated by VAE shows a poor fit with the real flows, particularly for an average packet size between 0 and 500. Flows from WGAN also show significant differences when compared to real flows. The flows generated by VGAN and CGAN show a better fit with the real traffic flow. Fig. 10 presents a comparison of the distribution of the

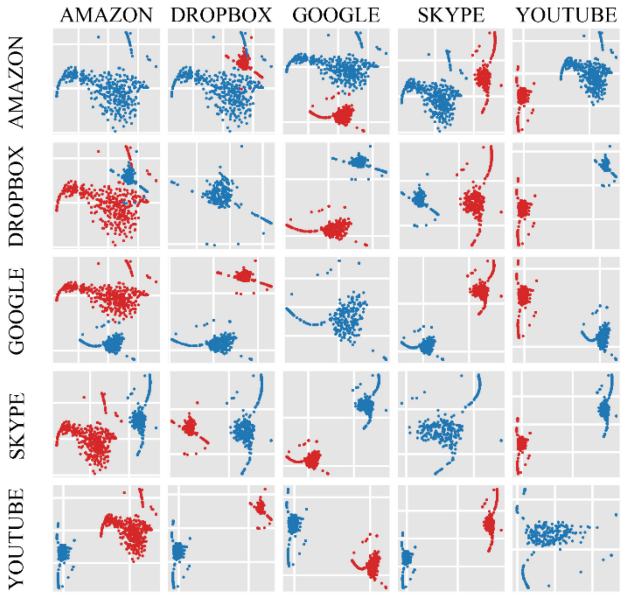

Fig. 8. t-SNE visualization when perp = 200.

flow IAT mean attribute between synthetic and real flows. The blue curves and bars display the statistical distribution of the Flow IAT Mean feature of the real traffic, which was used to train the IP2Vec model and CGAN. The red ones represent the synthetic traffic produced by various GAI approaches. Subfigures (a)–(d) illustrate the traffic distribution comparison of Skype, while (e)–(h) depict the traffic of YouTube. The traffic from Skype and YouTube will be used as examples in this study. The worst performance was observed on Skype and YouTube traffic, which resulted in the largest deviation in fitting curves. VAE and CGAN exhibit similar performance in these two applications. Based on the previous results analysis, it is concluded that CGAN outperforms the other three approaches in traffic flow generation tasks. The synthetic traffic flows generated by CGAN exhibit similar behavior to real traffic flows. Moreover, the flows generated by VAE, VGAN, and WGAN are produced using different models. New types of traffic require a new model to be trained. Other approaches have significantly higher training and storage costs in comparison to CGAN to generate the same amount of traffic types. Other GAN models mentioned in this article cannot generate specific type of flows unless they are trained with only one type of flows, thus multiple trained models are required for multiple types of flows. Our experiments have demonstrated that using CGAN is a feasible and effective method for generating network traffic flow. In the upcoming section, we will introduce the method of similarity measure to quantitatively evaluate the quality of synthesized samples.

D. Similarity Measure

Similarity is extensively used as a metric in classification or clustering and is measured based on the distance between samples. However, there is currently no widely recognized method to evaluate the similarity between network packets [35]. The choice of similarity measurement methods also influences the

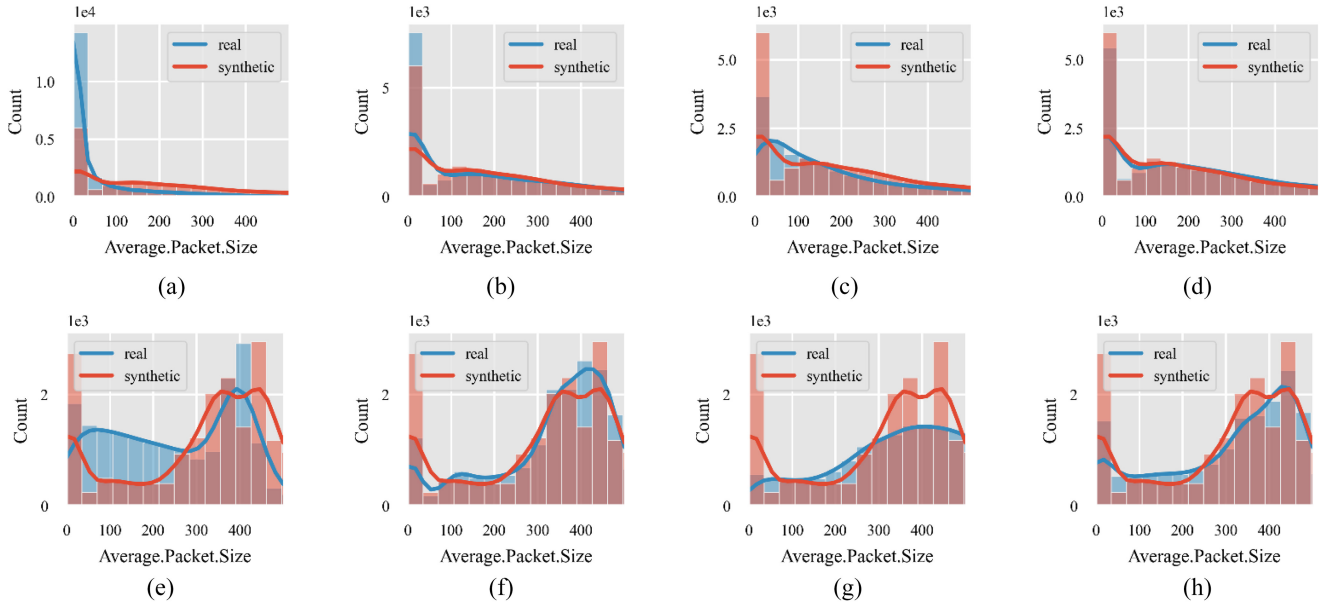

Fig. 9. Statistical distribution of average packet size. (a) Amazon/VAE. (b) Amazon/VGAN. (c) Amazon/WGAN. (d) Amazon/CGAN. (e) DropBox/VAE. (f) DropBox/VGAN. (g) DropBox/WGAN. (h) DropBox/CGAN.

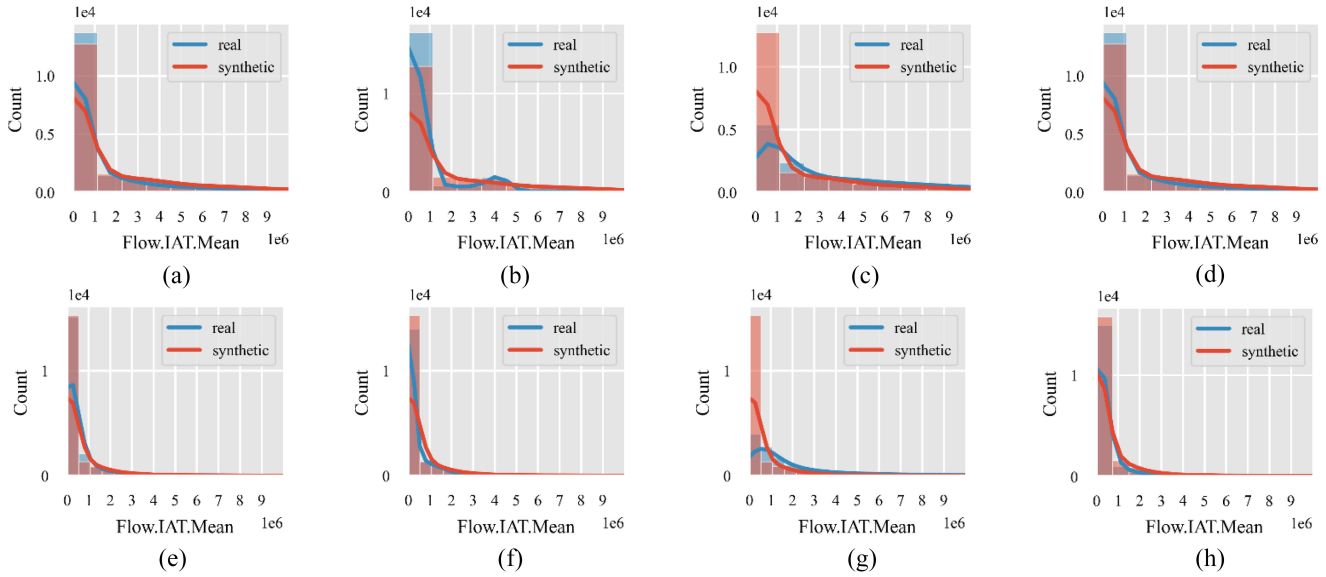

Fig. 10. Statistical distribution of flow IAT mean. (a) Skype/VAE. (b) Skype/VGAN. (c) Skype/WGAN. (d) Skype/CGAN. (e) YouTube/VAE. (f) YouTube/VGAN. (g) YouTube/WGAN. (h) YouTube/CGAN.

evaluation results. We introduce three similarity measurement techniques, commonly used in natural language processing tasks and data mining, namely the Earth mover's distance (EMD) [15], the cosine similarity [25], and the Euclidean distance [23]. These measures evaluate the similarity between synthetic and real traffic flows from different perspectives.

1) Earth Mover's Distance: First, we introduce the EMD, i.e., the first Wasserstein distance, which is used to measure the difference between two multidimensional distributions. It transforms the distance computing problem into a transportation problem that measures the least amount of work required to move the earth in one distribution with a mass of earth to the other with a collection of holes. EMD is calculated based on the distribution of features shown in Figs. 9 and 10. Consider the distributions of feature x in $\mathbf{X}^{\text{real}}$ and $\mathbf{X}^{\text{syn}}$ with c_1 and c_2 clusters, expressed as $P^x = \{(p_1^x, w_{p_1}^x), \ldots, (p_{c_1}^x, w_{q_{c_1}}^x)\}$ and $Q^x = \{(q_1^x, w_{q_1}^x), \ldots, (p_{c_2}^x, w_{q_{c_2}}^x)\}$. p_i^x, q_i^x mean the average value of ith cluster. $w_{p_i}^x, w_{q_i}^x$ mean the distribution fraction of ith cluster. Let $\mathbf{D} = [d_{ij}]$ be the ground distance matrix, where d_{ij} is calculated as the ground distance between p_i^x and q_j^x. Assuming f_{ij} is a flow between p_i^x and p_j^{syn}, the purpose is to find a flow $\mathbf{F} = [f_{ij}]$ that has the minimum overall cost

$$\text{Cost}(P^x, Q^x, \mathbf{F}) = \sum_{i=1}^{c_1} \sum_{j=1}^{c_2} f_{ij} d_{ij}. \tag{15}$$

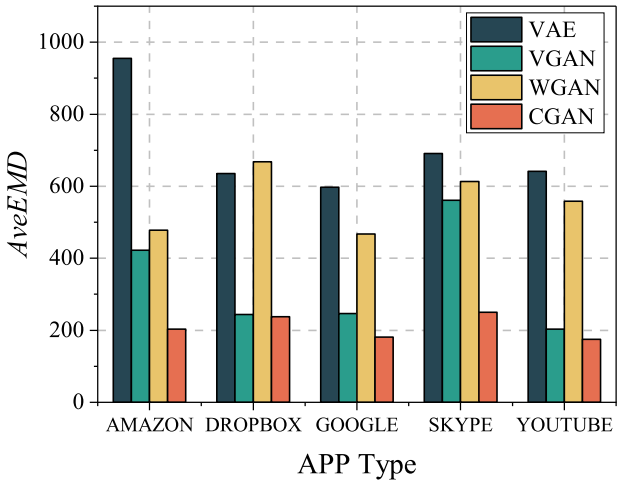

Fig. 11. Wasserstein distance between synthetic and real flow.

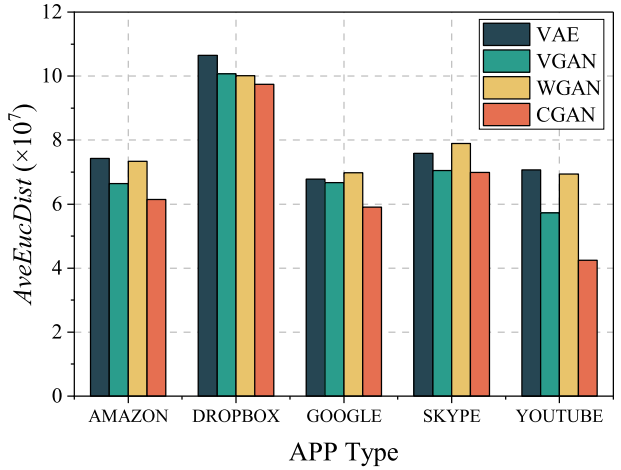

Fig. 12. Euclidean distance between synthetic and real flow.

After obtaining the optimal flow, EMD is defined as the work normalized by the total flow, as follows:

$$\mathrm{EMD}\left(P^x, Q^x\right) = \frac{\sum_{i=1}^{c_1}\sum_{j=1}^{c_2} f_{ij} d_{ij}}{\sum_{i=1}^{c_1}\sum_{j=1}^{c_2} f_{ij}}. \tag{16}$$

We calculate the average EMD of all features between $\mathbf{X}^{\mathrm{real}}$ and $\mathbf{X}^{\mathrm{syn}}$, expressed as

$$\mathrm{AveEMD} = \frac{\sum_{j=1}^{m} \mathrm{EMD}\left(P^j, Q^j\right)}{m}. \tag{17}$$

Fig. 11 displays the EMD comparison results of different approaches on five applications. According to the observations, CGAN manifests lesser EMD than the other approaches, implying that the distribution of synthetic traffic flows generated by CGAN better matches the real flows. The EMD of Dropbox, Google, and YouTube traffic flows generated by VGAN approximates CGAN, however, it is still larger compared to CGAN. VAE and WGAN exhibit poor performance on multiple traffic types. The comparison experiment on EMD reinforces that CGAN better learns the distribution characteristics of various traffic types. Therefore, the generated traffic flows conform better to the data distribution of traffic flows from various applications.

2) Earth Mover's Distance: Next, we compute the Euclidean distance between the synthetic and real traffic flows. The Euclidean distance is a commonly used measure of similarity. We randomly select n^{syn} samples from the generated synthetic traffic flows, expressed as $\mathbf{A} = \{a^{(1)}, a^{(2)}, \ldots, a^{(n)}\}$. Assuming there are n^{real} real flows in the training data set. For $a \in \mathbf{A}$, we calculate its Euclidean distance from the feature vector of a real network flow using the following formula:

$$\mathrm{euc\_dis}(a, b) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2} \tag{18}$$

where m is the feature number of the traffic flow, and b_i means the value of the ith feature of x^{real}. For example, b_1 is the flow duration. AveEucDist is defined as the average Euclidean distance between each selected synthetic sample and each real traffic flow, calculated as

$$\mathrm{AveEucDist} = \frac{\sum_{j=1}^{n^{\mathrm{real}}} \sum_{i=1}^{n^{\mathrm{syn}}} \mathrm{euc\_dis}\left(a^{(i)}, (x^{\mathrm{real}})^{(j)}\right)}{n^{\mathrm{real}} \times n^{\mathrm{syn}}} \tag{19}$$

where $(x^{\mathrm{real}})^{(j)}$ is the jth sample in the real flow data set.

Fig. 12 illustrates the AveEucDist comparison of various applications using different approaches. The AveEucDist between the generated synthetic traffic flows and the real traffic flows of VAE and WGAN is still lower than that of CGAN and VGAN. This is because the synthetic traffic flows generated by VAE and WGAN have a larger distance in vector space from the real traffic flows. VGAN exhibits performance similar to that of CGAN concerning the flows of Skype. However, it still does not outperform CGAN. From Fig. 12, it is evident that CGAN generates synthetic samples that are more spatially similar to real flows, as measured by Euclidean distance.

3) Cosine Similarity: Next, we calculate the cosine similarity to assess the similarity between two individual samples. Cosine similarity is a commonly used method of measurement in fields, such as information retrieval, data mining, machine translation, and document replication detection. For $a \in \mathbf{A}$, the cosine similarity between it and the feature vector of a real network flow b is calculated as follows:

$$\mathrm{cos\_sim}(a, b) = \frac{\sum_{i=1}^{m} (a_i \times b_i)}{\sqrt{\sum_{i=1}^{m} (a_i)^2} \times \sqrt{\sum_{i=1}^{m} (b_i)^2}}. \tag{20}$$

Then, we calculate the average cosine similarity of the s selected samples, denoted as AveCosSim

$$\mathrm{AveCosSim} = \frac{\sum_{j=1}^{n^{\mathrm{real}}} \sum_{i=1}^{n^{\mathrm{syn}}} \mathrm{cos\_sim}\left(a^{(i)}, (x^{\mathrm{real}})^{(j)}\right)}{n^{\mathrm{real}} \times n^{\mathrm{syn}}}. \tag{21}$$

The larger AveCosSim means the generated synthetic flow fits the real traffic more than others. It can be seen in Fig. 13, among all the application types, the traffic samples generated by CGAN has largest AveCosSim than the other three models.

E. Discussion

Capturing traffic from personal users in NTN-enabled IoT is challenging, and the issue of data privacy also

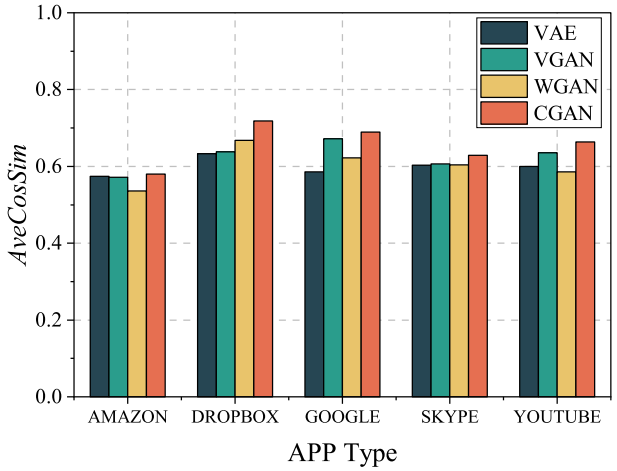

Fig. 13. Cosine similarity between synthetic and real flow.

restricts the comprehensive analysis and sharing of Internet traffic [17]. Additionally, collecting large-scale traffic from different Internet applications used by a large number of users in the real world is expensive and challenging. The cost of collecting, processing, storing and transmitting traffic data increases with the volume and type of data. Therefore, a GAI-based approach is introduced to generate network traffic that simulates the network behavior in different application scenarios. The GAI model learns the traffic characteristics from the physical network and produces traffic data for NTN-enabled IoT operation, maintenance, and algorithm testing at a faster speed and a lower cost.

Generally, network traffic generators are created by building packets, which rarely depict the traffic patterns that are accurate for the actual network. In addition, simulating the interaction between users and service providers requires deploying and configuring original applications, which ultimately increases the time and computing cost involved. With the assistance of GAI models, we propose a synthetic network traffic flow generation method based on CGAN. One of our important emphases is on the five-tuple data flow information that is challenging to process in machine learning models. Using the embedding technique in NLP scenarios, the address attributes can be learned and synthesized into a low-dimensional vector representation by GAN models. Due to the inability of traditional traffic generators to conform to specific application behaviors, we introduce a synthetic traffic flow generation model based on CGAN. After being trained, CGAN receives specific traffic type labels and generates corresponding traffic flows for multiple applications in NTN-enabled IoT networks.

Our study also has some limitations. The objective of this study is to learn and generate flow-level features, rather than considering packet-level features. Although some characteristics overlap between flow-level and packet-level features, they have different abilities to describe and analyze network behavior. Further research is needed to explore the use of GAI methods in co-generating packet-level and flow-level features. The features that describe traffic flows are limited by the capabilities of extraction tools. To train our CGAN model, we select the top 20 features using random forests with importance sorting, which leads to some loss of original traffic information. The network traffic data set used in this article is captured from the campus network of Universidad Del Cauca [29]. Due to user data privacy and data availability constraints, the underlying network topology of the collected data is not available. In addition, campus network traffic differs from traffic characteristics observed in the wide area network (WAN). Thus, our study is based solely on the characteristics of campus network traffic and does not consider the more complex traffic characteristics and application types that are often observed in the WAN scenario. The experiments conducted in this study have demonstrated the performance and potential application of the GAI method for generating traffic flow. It should be noted that the proposed approach aims to generate individual traffic flows rather than a sequence of flows that express behavioral patterns between users and applications over long connectivity connections. Generating continuous and correlated traffic flows by using models that integrate GAN with time series models could be an effective method for discovering traffic patterns in a complete connection. Finally, in the absence of widely recognized metrics or evaluation methods for synthetic traffic flows [35], we have used several metrics that are commonly employed in NLP tasks and data mining, such as Cosine Similarity and EMD. Although these metrics managed to evaluate the similarity between synthetic and real flows, it is hard to explain what the similarity means from the networking perspective instead of statistical perspective. The correlation between similarity and flow behavior is also unclear. Therefore, additional research is required to explore more accurate and appropriate metrics for evaluating the performance of traffic generation tasks.

VI. Conclusion

In conclusion, the proposed GAI-based synthetic traffic flow generation framework for NTN-enabled IoT significantly enhances traffic analysis and performance evaluation by learning from real network data to create realistic synthetic traffic. The use of IP2Vec-based models allows effective representation of network attributes, while CGANs generate traffic flows that mimic real network behaviors. Despite focusing on flow-level features and using a campus network data set, our approach demonstrates the potential for broad applications. Future research should address packet-level features, and sequence generation, and develop more accurate evaluation metrics to further refine and validate this innovative method.

References

[1] M. Vaezi et al., "Cellular, wide-area, and non-terrestrial IoT: A survey on 5G advances and the road toward 6G," *IEEE Commun. Surveys. Tuts.*, vol. 24, no. 2, pp. 1117–1174, 2nd Quart., 2022, doi: 10.1109/COMST.2022.3151028.

[2] V. R. Chandrika, J. Chen, L. Lampe, G. Vos, and S. Dost, "SPIN: Synchronization signal-based positioning algorithm for IoT nonterrestrial networks," *IEEE Internet Things J.*, vol. 10, no. 23, pp. 20846–20867, Dec. 2023, doi: 10.1109/JIOT.2023.3283989.

[3] M. M. Azari et al., "Evolution of non-terrestrial networks from 5G to 6G: A survey," *IEEE Commun. Surveys. Tuts.*, vol. 24, no. 4, pp. 2633–2672, 4th Quart., 2022, doi: 10.1109/COMST.2022.3199901.
[4] S. Barick and C. Singhal, "Multi-UAV assisted IoT NOMA uplink communication system for disaster scenario," *IEEE Access*, vol. 10, pp. 34058–34068, 2022, doi: 10.1109/ACCESS.2022.3159977.
[5] A. Bera, S. Misra, C. Chatterjee, and S. Mao, "CEDAN: Cost-effective data aggregation for UAV-enabled IoT networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 9, pp. 5053–5063, Sep. 2023, doi: 10.1109/TMC.2022.3172444.
[6] H. Nguyen-An, T. Silverston, T. Yamazaki, and T. Miyoshi, "IoT traffic: Modeling and measurement experiments," *IoT*, vol. 2, no. 1, pp. 140–162, 2021, doi: 10.3390/iot2010008.
[7] C. Amatetti, M. Conti, A. Guidotti, and A. Vanelli-Coralli, "NB-IoT random access procedure via NTN: System level performances," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Seoul, South Korea, 2022, pp. 2381–2386, doi: 10.1109/ICC45855.2022.9838552.
[8] H. Touvron et al., "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.
[9] B. Rong and H. Rutagemwa, "Leveraging large language models for intelligent control of 6G integrated TN-NTN with IoT service," *IEEE Netw.*, vol. 38, no. 4, pp. 136–142, Jul. 2024, doi: 10.1109/MNET.2024.3384013.
[10] J. Wen et al., "From generative AI to generative Internet of Things: Fundamentals, framework, and outlooks," *IEEE Internet Things Mag.*, vol. 7, no. 3, pp. 30–37, May 2024, doi: 10.1109/IOTM.001.2300255.
[11] S. Ghazanfar, F. Hussain, A. U. Rehman, U. U. Fayyaz, F. Shahzad, and G. A. Shah, "IoT-Flock: An open-source framework for IoT traffic generation," in *Proc. Int. Conf. Emerg. Trends Smart Technol. (ICETST)*, Karachi, Pakistan, 2020, pp. 1–6, doi: 10.1109/ICETST49965.2020.9080732.
[12] S. Sharif, S. Zeadally, and W. Ejaz, "Resource optimization in UAV-assisted IoT networks: The role of generative AI," 2024, *arXiv:2405.03863*.
[13] O. A. Adeleke, N. Bastin, and D. Gurkan, "Network traffic generation: A survey and methodology," *ACM Comput. Surv.*, vol. 55, no. 2, pp. 1–23, 2022, doi: 10.1145/3488375.
[14] A. Lieto, Q. Liao, and C. Bauer, "A generative approach for production-aware industrial network traffic modeling," in *Proc. IEEE Globecom Workshops*, 2022, pp. 575–580, doi: 10.1109/GCWkshps56602.2022.10008549.
[15] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative deep learning for Internet of Things network traffic generation," in *Proc. 25th Pac. Rim Int. Symp. Dependable Comput.*, 2020, pp. 70–79, doi: 10.1109/PRDC50213.2020.00018.
[16] S. K. Nukavarapu, M. Ayyat, and T. Nadeem, "MirageNet—Towards a GAN-based framework for synthetic network traffic generation," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Rio de Janeiro, Brazil, 2022, pp. 3089–3095, doi: 10.1109/GLOBECOM48099.2022.10001494.
[17] A. Iqbal et al., "Empowering non-terrestrial networks with artificial intelligence: A survey," *IEEE Access*, vol. 11, pp. 100986–101006, 2023, doi: 10.1109/ACCESS.2023.3314732.
[18] S. Hui et al., "Knowledge enhanced GAN for IoT traffic generation," in *Proc. ACM Web Conf.*, New York, NY, USA, 2022, pp. 3336–3346, doi: 10.1145/3485447.3511976.
[19] J. An, B. Kang, Q. Ouyang, J. Pan, and N. Ye, "Covert communications meet 6G NTN: A comprehensive enabler for safety-critical IoT," *IEEE Netw.*, vol. 38, no. 4, pp. 17–24, Jul. 2024, doi: 10.1109/MNET.2024.3379864.
[20] M. Abdel-Basset, L. Abdel-Fatah, K. A. Eldrandaly, and N. M. Abdel-Aziz, "Enhanced computational intelligence algorithm for coverage optimization of 6G non-terrestrial networks in 3D space," *IEEE Access*, vol. 9, pp. 70419–70429, 2021, doi: 10.1109/ACCESS.2021.3078585.
[21] Q. Liu, S. Wang, Z. Qi, K. Zhang, and Q. Liu, "Edge intelligence for IoT Services in 6G integrated terrestrial and non-terrestrial networks," *IEEE Netw.*, vol. 38, no. 4, pp. 80–87, Jul. 2024, doi: 10.1109/MNET.2024.3384389.
[22] C. Liu, W. Feng, X. Tao, and N. Ge, "MEC-empowered non-terrestrial network for 6G wide-area time-sensitive Internet of Things," *Engineering*, vol. 8, pp. 96–107, Jan. 2022, doi: 10.1016/j.eng.2021.11.002.
[23] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Comput. Secur.*, vol. 82, pp. 156–172, May 2019, doi: 10.1016/j.cose.2018.12.012.
[24] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," 2017, *arXiv:1712.09405*.
[25] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "IP2Vec: Learning similarities between IP addresses," in *Proc. IEEE Int. Conf. Data Min. Workshops (ICDMW)*, New Orleans, LA, USA, 2017, pp. 657–666, doi: 10.1109/ICDMW.2017.93.
[26] S. Ishikawa, S. Ozawa, and T. Ban, "Port-piece embedding for darknet traffic features and clustering of scan attacks," in *Proc. Int. Conf. Neural Inf. Process. (ICONIP)*, Bangkok, Thailand, 2020, pp. 593–603, doi: 10.1007/978-3-030-63833-7_50.
[27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA, 2013, pp. 3111–3119, doi: 10.48550/arXiv.1310.4546.
[28] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*.
[29] J. S. Rojas, Á. Rendón, and J. C. Corrales, "Consumption behavior analysis of over the top services: Incremental learning or traditional methods?" *IEEE Access*, vol. 7, pp. 136581–136591, 2019, doi: 10.1109/ACCESS.2019.2942782.
[30] R. Snoussi and H. Youssef, "VAE-based latent representations learning for botnet detection in IoT networks," *J. Netw. Syst. Manag.*, vol. 31, no. 1, p. 4, 2023, doi: 10.1007/s10922-022-09690-4.
[31] T. Zixu, K. S. K. Liyanage, and M. Gurusamy, "Generative adversarial network and auto encoder based anomaly detection in distributed IoT networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, 2020, pp. 1–7, doi: 10.1109/GLOBECOM42002.2020.9348244.
[32] S. Fathi-Kazerooni and R. Rojas-Cessa, "GAN Tunnel: Network traffic steganography by using GANs to counter Internet traffic classifiers," *IEEE Access*, vol. 8, pp. 125345–125359, 2020, doi: 10.1109/ACCESS.2020.3007577.
[33] Z. Zhou et al., "CreST: A credible spatiotemporal learning framework for uncertainty-aware traffic forecasting," in *Proc. 17th ACM Int. Conf. Web Search Data Min.*, 2024, pp. 985–993.
[34] Y. Guo, G. Xiong, Z. Li, J. Shi, M. Cui, and G. Gou, "Combating imbalance in network traffic classification using GAN based oversampling," in *Proc. IFIP Netw., Conf.*, 2021, pp. 1–9, doi: 10.23919/IFIPNetworking52078.2021.9472777.
[35] C. Wu, Y. Chen, P. Chou, and C. Wang, "Synthetic traffic generation with Wasserstein generative adversarial networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Rio de Janeiro, Brazil, 2022, pp. 1503–1508, doi: 10.1109/GLOBECOM48099.2022.10001157.