USING DIFFUSION MODELS FOR IP HEADER TRACE GENERATION IN "PRACTICAL GAN-BASED SYNTHETIC IP HEADER TRACE GENERATION USING NETSHARE"

Divyanshu Daiya
ddaiya@purdue.edu

Shyaman Jayasundara
sjayasun@purdue.edu

1 ABSTRACT

The paper "Practical GAN-based Synthetic IP Header Trace Generation using NetShare" (Yin et al., 2022) proposes a method for generating synthetic IP header traces using Generative Adversarial Networks (GANs) to solve the problem of limited and costly real-world data. While the proposed approach is effective, it suffers from certain limitations, such as the GAN's inability to model long-term dependencies, difficulties in training, and issues with mode collapse. To overcome these limitations, we propose exploring the use of diffusion models as an alternative method for IP header trace generation. Diffusion models have shown promise in addressing the limitations of GANs and have demonstrated the ability to generate high-quality samples with fewer training iterations (Cao et al., 2022; Ho et al., 2020). By investigating the potential of diffusion models, we aim to contribute to the advancement of IP header trace generation and its applications in network security and research. We see that our developed Condionnal GAN based DDPM archtiecture generated synthetic traces with better feature feilds than NetFlow but perform relatively poorly on generating attribute feilds in the synthetic flows.

2 INTRODUCTION

Header traces at the packet and flow levels are crucial for network management tasks, such as developing new anomaly detection algorithms. However, obtaining access to such traces is difficult due to privacy and business concerns. A solution is to generate synthetic traces, and various methods have been used in the past, such as simulation-driven methods, model-driven techniques, and machine-learning models. However, these methods have limitations, as they require domain knowledge and human effort to determine workload features, and they do not generalize well across applications. On the other hand, ML models generalize easily but fail to capture domain-specific properties.

This approach explores the possibility of using machine learning to generate synthetic packet and flow header traces through GANs. However, existing GAN-based methods face several challenges such as fidelity, scalability-fidelity tradeoffs, and privacy-fidelity tradeoffs. To address these limitations, the authors propose NetShare (Yin et al., 2022), which can handle these issues by considering the limitations of GAN-based methods. They employed several key ideas in developing NetShare, including learning synthetic models for merged flow-level traces across epochs, introducing data parallelism learning to improve scalability, and using differentially-private model training to handle privacy concerns associated with sharing traces.

We will firstly anlayse their work closely and replicate their results and then we will develop a Diffusion based Synthetic Flow Generating Model, with the goal of generating better Synthetic Packets than one generated by the authors. We will test our model using the same metrices used by the authors.

3 DESIGN

3.1 NETSHARE USING GENERATIVE ADVERSARIAL NETWORKS

3.1.1 PRELIMINARIES

Generative Adversarial Networks (GANs) are a class of deep learning models that estimate generative models through a minimax two-player game. Two models with adversarial objectives are trained simultaneously: a generator G that captures a data distribution to produce real-like data, and a discriminator D that distinguishes between real and generated data.

The generator $G(z;\theta_g)$ is a differentiable function that maps the input noise variable $z \in Z$ to the data space X, i.e., $G : Z \to X$. Conversely, the discriminator $D(x;\theta_d)$ is a function that maps input from the data space to the probability that the input data is real, i.e., $D : X \to [0,1]$. The discriminator is trained to maximize the probability of assigning correct labels to both real and generated data. At the same time, the generator is trained to minimize $\log(1 - D(G(z)))$, which represents the probability of the discriminator correctly identifying generated samples as not real.

The objective of the zero-sum game is to formulate an optimization problem that can be solved using an iterative numerical approach. The problem aims to minimize the generator function G while simultaneously maximizing the discriminator function D. This can be mathematically represented as:

$$\min_G; \max_D; \mathbb{E}_{x\sim pdata(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1}$$

The objective of the discriminator is to maximize the probability of assigning the correct label to both real and generated data. The generator, on the other hand, aims to minimize the probability of the discriminator correctly identifying generated samples as not real.

Time series GANs

Recent advancements in GANs have proposed effective techniques for generating time series data. One such method is TimeGAN (Yoon et al., 2019), which combines the flexibility of unsupervised GANs with control over conditional temporal dynamics offered by supervised autoregressive models. TimeGAN introduces an embedding and recovery network to enable reversible mapping between features and latent representations, while reducing the high dimensionality of the adversarial learning space. This approach leverages the fact that temporal dynamics are often driven by factors aligned with lower dimensional variation.

Relying solely on the discriminator's binary adversarial feedback, as captured by the unsupervised loss L_U described earlier, may not be sufficient incentive for the generator to capture temporal dynamics in the data. Effectual embedding and recovery functions are therefore learned using reconstruction loss L_R in TimeGAN. The generator is trained to first generate latent representation using noise coupled with a temporally earlier latent representation, and an additional supervised loss L_S is introduced to further discipline learning.

Let θ_e, θ_r, θ_g, and θ_d denote parameters of the embedding, recovery, generator, and discriminator networks, respectively, and γ and η be two hyperparameters. The optimization procedure is as follows:

$$\min_{\theta_e,\theta_r} (\gamma L_S + L_R)$$

$$\min_{\theta_g}, \left(\eta L_S + \max_{\theta_d} L_U\right)$$

The first line represents the optimization of the embedding and recovery parameters, while the second line represents the optimization of the generator and discriminator parameters.

Below are the formal descriptions of the losses, where x and $\hat{x}$ represent the actual and recovered features, y and $\hat{y}$ correspond to whether a sample is real or synthetic, h denotes the latent repre-

sentation, z denotes noise, and g is the generator function implemented using a recurrent neural network.

The reconstruction loss L_R is defined as the expected value of the sum of L_2 distances between the actual and recovered features over the time steps 1 to T:

$$L_R = \mathbb{E}_{x_{1:T} \sim p} \sum_{t=1}^{T} ||x_t - \tilde{x}_t||_2$$

The unsupervised loss L_U is defined as the sum of the logarithms of the discriminator outputs y_t for real samples and $(1 - \hat{y}_t)$ for synthetic samples over the time steps 1 to T:

$$L_U = \mathbb{E}_{x_{1:T} \sim p} \sum_{t=1}^{T} \log y_t + \mathbb{E}_{x_{1:T} \sim \hat{p}} \sum_{t=1}^{T} \log(1 - \hat{y}_t)$$

The supervised loss L_S is defined as the expected value of the sum of L_2 distances between the latent representations and the generator outputs over the time steps 1 to T:

$$L_S = \mathbb{E}_{x_{1:T} \sim p} \sum_{t=1}^{T} ||h_t - g(h_{t-1}, z_t)||_2$$

DoppelGANger

A recent study by a team mostly composed of the same researchers who worked on NetShare proposes a time series GAN specifically designed for networking data called DoppelGANger (Lin et al., 2020). They discovered that the standard approaches are not capable of effectively capturing long-term temporal correlations between measurements, such as packet loss rate, bandwidth, and delay, and metadata such as ISP name or location. To address these issues, they introduce various innovations such as an auxiliary discriminator and a batched recurrent neural network generator. However, DoppelGANger is mostly a variation of the more standard TimeGAN.

3.1.2 NETSHARE

NetShare uses DoppelGANger as the foundation to enhance fidelity, scalability, and privacy while concentrating on generating IP header traces. This DoppelGANger is used as the generative model to produce synthetic data in this data pipeline. The following are the key changes and their associated rationale in this work:

1. *Improved data formatting* To achieve better fidelity, the data formatting was enhanced. Rather than utilizing a per-epoch tabular approach, the header trace generation was restructured as a flow-based time series generation problem, which should help capture both intra-measurement and inter-measurement correlations.
2. *Improved data preprocessing* To enhance fidelity, the data features, particularly those with a large number of unique values such as IP addresses, were encoded using both domain knowledge and machine learning to make the representation space more conducive to learning. To handle numerical semantics such as packets/byte per flow with a large support value, log transformation was utilized to decrease the range. In addition, the authors used IP2Vec, a representation learning algorithm that transforms inputs so that comparable data based on network context information are close together in the new representation space, to transform the categorical port numbers and protocols. However, IP2Vec was not used for IP addresses, and bitwise encoding was employed instead because IP2Vec relies on training data, which could lead to privacy concerns.
3. *Smart parallelization* To improve scalability, the authors of NetShare parallelized the training pipeline. However, dividing the entire trace into chunks without consideration of correlations across the chunks could cause a loss of information. Additionally, splitting the trace by a fixed number of packets per chunk could negatively impact differential privacy guarantees, as the presence of a single packet could significantly alter the trained model. To address these issues, NetShare splits the trace into fixed time intervals and adds flow tags

to each flow header to preserve correlations. The authors also utilize fine tuning to enable parallel training across chunks while capturing cross-chunk correlations. Specifically, they use the first chunk to obtain an initial model and then use this as a warm start for fine-tuning subsequent chunks in parallel.

4. *Transfer learning with differentially private optimizer* To ensure privacy, NetShare employs DP-SGD [1] during training. However, training a model from scratch using DP-SGD can lead to decreased fidelity due to the tradeoff between privacy and fidelity. To address this issue, NetShare utilizes a transfer learning approach in which the model is first trained without DP-SGD on a public dataset. The pretrained model is then further trained on the private dataset of interest using DP-SGD. This approach helps preserve both fidelity and privacy, particularly when the public and private datasets share some similarity, allowing the model trained on the private dataset to converge faster.

The NetShare pipeline, as shown in Figure 1, takes an input packet/flow trace and splits it into a set of flows based on their 5-tuples, denoted as K. Each flow is further divided into N chunks based on the duration of the flow, where N is 10 in the NetShare implementation. These packet/flow fields of these chunks are then encoded using continuous and categorical encoding functions. The encoded chunks are then fed into DoppleGANger time-series GANs. There are N number of GANs in total, with each learning to generate a specific chunk of any flow. Once trained, these models are used to generate synthetic encoded chunks, which are decoded and assembled into a full trace by sorting all the packets/flows by their timestamps. This is the overall pipeline of NetShare.

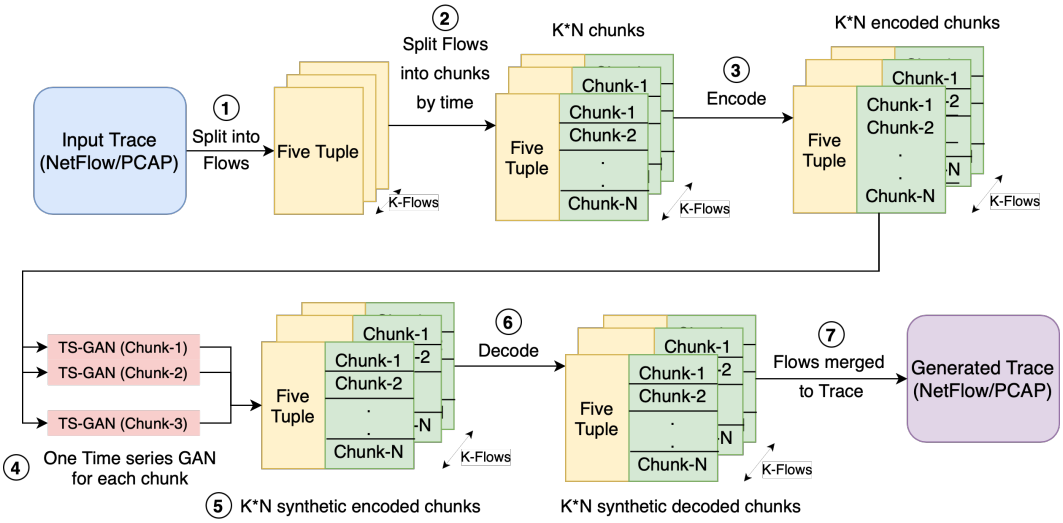


Figure 1: NetShare trace generation pipeline

3.2 DDPM APPROACH FOR SYNTHETIC FLOW GENERATION

4 IMPLEMENTATION

To reproduce the work presented in the original paper, we chose the most recent PyTorch-based version. However, using flow data for generation necessitates substantial GPU or CPU resources. Consequently, we set up a Google Cloud (GC) evaluation testbed for NetShare, which consisted of 10 Cloudlab machines, each equipped with two Intel Xeon Silver 4114 10-core CPUs running at 2.20 GHz and 192GB DDR4 memory. Unfortunately, our $300 GCP credits were insufficient to fully run the model.

Considering the substantial cost of procuring or creating such resources, we decided to establish a single machine with a moderately powerful CPU and GPU for compute-intensive training tasks. Additionally, we encountered issues with the current PyTorch version, as it was not compatible with GPU when we attempted to run it directly. As a result, we had to modify their code in several places and rework some parts to enable execution on a single machine. We made numerous adjustments to

accommodate these changes.
In the end, we utilized a virtual machine on Google Cloud with 32 cores, 80GB RAM, 1TB disk space, and access to an A30 GPU for our training and evaluation tasks.

5 Evaluation and Results

In order to evaluate the performance of original NetShare implementation, we measured its fidelity using various metrics for generated traces. We used PCAP and NetFlow datasets, including CAIDA (Walsworth et al., 2015) and UGR16 (Maciá-Fernández et al., 2018), which were used in the original NetShare paper. The flow traces in UGR16 were obtained from NetFlow v9 collectors operating in a Spanish ISP network, while the packet traces in CAIDA were generated from anonymized traffic originating from a commercial backbone link. We used the synthetic data provide by authors to generate the plots, as we couldn't train their full model.
For testing the NetFLow against our extension(DDPM) we utilised 1000 flow traces from UGR16 for training the extension results comparison for this is done in the end of report and also included as PDFs along with the report.

5.1 NetShare with GANs (Original implementation)

Fidelity measured through header correlations of packets/flows

We analyzed the URG16 dataset to reconstruct the cumulative distribution function (CDF) of flow volume (bytes per flow), CDF of flow size (packets per flow), and CDF of records with the same five tuples. Additionally, we reconstructed the CDF of flow size (number of packets) for the CAIDA dataset. To evaluate the fidelity of NetShare-generated traces, we compared the CDFs of these properties between the original and synthetic traces. The comparison is shown in Figure 3b and Figure 2, which demonstrates a high degree of similarity in the distribution of these properties, indicating the effectiveness of NetShare in preserving the statistical properties of the original traces. This can also be observed in Figure 3a and Figure 3c, where the synthetic records have similar distributions in flow volume and records with the same five tuples.

Fidelity measured through header field properties

We analyzed the most frequently occurring service and source ports in the synthetic and real traces and calculated their relative frequencies. The distribution of these ports is shown in Figure 4a and 4b, which demonstrates a high level of similarity between the synthetic and real traces. Furthermore, we compared the distributions of three protocol sets (TCP, UDP, and all others) in the real and synthetic traces, as shown in Figure 4c. This comparison confirms that the synthetic traces generated by NetShare maintain fidelity with the real traces.

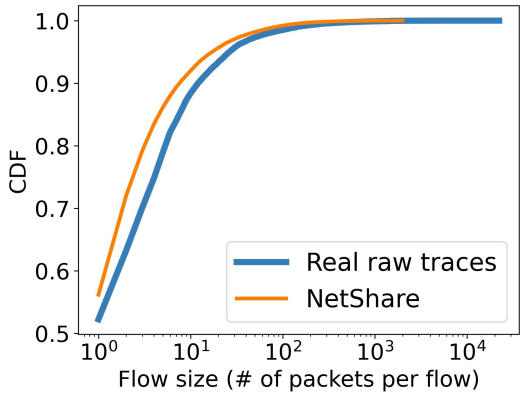
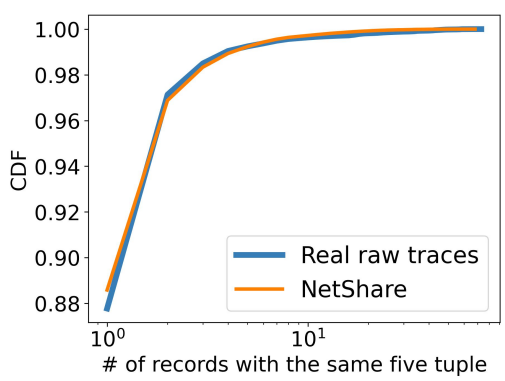
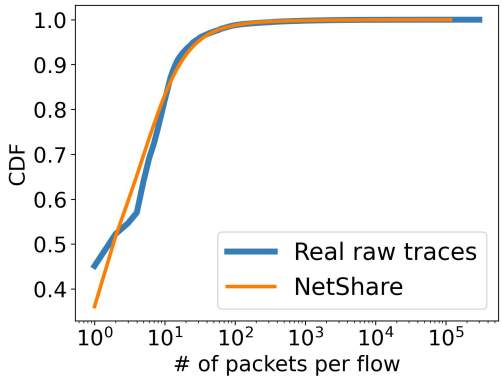


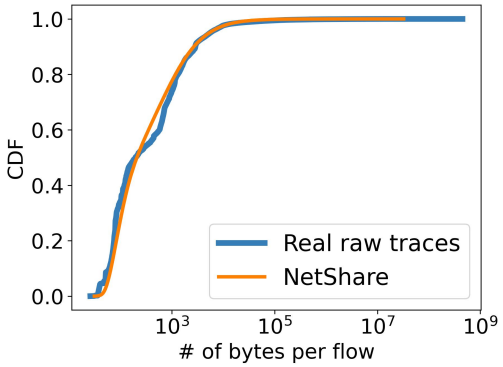
Figure 2: CDF of flow size (# of packets) on CAIDA



(a) CDF of records with same five tuples

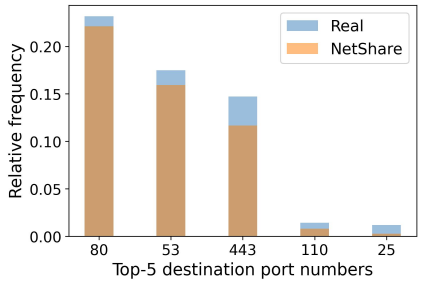


(b) CDF of # packets per flow

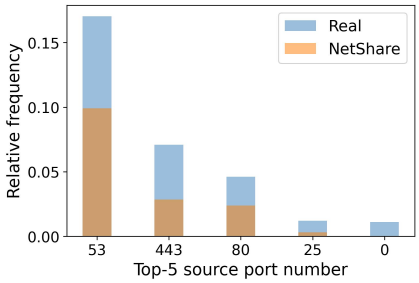


(c) CDF of # bytes per flow

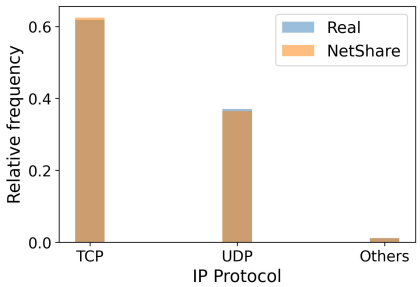
Figure 3: CDF comparisons of raw vs. NetShare-generated Netflow traces on UGR16



(a) Destination port distribution



(b) Source port distribution



(c) Protocol distribution

Figure 4: Comparisons of packet/flow properties of raw vs. NetShare-generated Netflow traces on UGR16

6 DDPM APPROACH FOR SYNTHETIC FLOW GENERATION

(In this section the x is intercahngbly used to denote D_flow, so, x_t would mean D_flow_t and vice-versa. And, p_θ and ϵ_θ represent the same denoising function.)

We have utilised the same input pipeline as decribed by (Yin et al., 2022), in their work D_flow is provided an input to GAN Network. We plan to use same high level training routine decribed by them, and only change their GAN network with DDPM-based architecture (Rasul et al., 2021), the inner training routine for DDPM and a brief introduction to DDPM is provided below.

Diffusion probabilistic models (Ho et al., 2020) add a small amount of Gaussian noise ϵ to the original signal D_flow (Yin et al., 2022) from time $t = 0$ and repeat the process until the signal is destroyed. As shown in Eq. below, q represents the process of restoring the original signal D_flow_0 from the noise, while $q\left(\mathrm{D\_flow}_t \mid \mathrm{D\_flow}_{t-1}\right)$ represents the diffusion process. If q is repeated sufficiently, $\mathrm{D\_flow_T}$ follows a normal distribution (Ho et al., 2020; Rasul et al., 2021).

$$q\left(\mathrm{D\_flow}_{1:\mathrm{T}} \mid \mathrm{D\_flow}_0\right) := \prod_{t=1}^{\mathrm{T}} q\left(\mathrm{D\_flow}_t \mid \mathrm{D\_flow}_{t-1}\right)$$

$$q\left(\mathrm{D\_flow}_t \mid \mathrm{D\_flow}_{t-1}\right) := \mathcal{N}\left(\mathrm{D\_flow}_t; \sqrt{1-\beta_t}\mathrm{D\_flow}_{t-1}, \beta_t, \mathrm{I}\right)$$

Where β indicates noise level and $\alpha := \prod_{i=1}^{t} 1 - \beta_t$. In addition, $\mathrm{D\_flow}_t$ can be sampled using reparameterization based on Gaussian distribution (Ho et al., 2020) in which Eq. below.

$$\mathrm{D\_flow}_t = \sqrt{1-\beta_t}\mathrm{D\_flow}_{t-1} + \sqrt{\beta_t}\epsilon$$

$$\min_\theta L(\theta) := \min_\theta \mathbb{E}_{t, \mathrm{D\_flow}_0 \sim q(\mathrm{D\_flow}_0), \epsilon \sim \mathcal{N}(0,\mathrm{I})}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\alpha_t} D\_flow_0 + (1-\alpha_t)\,\epsilon, t\right)\right\|^2\right]$$

ϵ_θ is a denoising function, and this function estimates the noise vector ϵ which is added to $\mathrm{D\_flow}_t$. Where t is uniform between 1 and T.

6.0.1 ISSUE WITH DIRECT USE OF DDPM ARCHITECHTURE

When generating synthetic data, our goal is to capture the underlying data ditribution in flow traces to be able to generate new data from noise/or from a given sample. However, given the multiple modalities in the flow data, the DDPM archtitecture would require a huge amount of training steps T to be able to learn the data distribution, as discussed by Xiao et al. (2022). So, this would make the model very computationally intensive to train.

Further DDPM, the denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is often approximated as Gaussian. This approximation is only accurate , when the step size β_t is sufficiently small, resulting in a Gaussian denoising distribution; and when the data marginal $q(\mathbf{x}_t)$ is Gaussian. We might be able to get to achieve the first scenario by using thousands of steps with small β_t. But, the second scenario is impossibly challenging, as transforming data the Flow Time series Data to the Gaussian distribution is difficult.

Now we know that the Conditional GANs as used in the original work have been shown to model complex conditional distributions in the vareity of domains Isola et al. (2018), so instead of directly adopting the DDPM training routine we can adopt GANs to approximate the true denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Our goal is to have small ($T \leq 8$) and hence diffusion step with bigger β_t.

6.0.2 DDPM UTILISING CONDITIONAL GAN

We use a variation of approach developed by Xiao et al. (2022). They have demonstarated the efficacy of such networks for image generation only, their efficacy in the temporal setting as in the task of Flow Generation haven't been tested by any of the present works to the best of our knowledge.

Builing on the ideas developed by Xiao et al. (2022), we set up the forward diffusion similar to the naive diffusion models Ho et al. (2020).

Our training is formulated by matching the conditional GAN generator $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ using an adversarial loss that minimizes a divergence D_{adv} per denoising step:

$$\min_\theta \sum_{t\geq 1} \mathbb{E}_{q(\mathbf{x}_t)} \left[D_{\text{adv}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \right], \tag{2}$$

where D_{adv} is *softened reverse KL* , which is different from the forward KL divergence used in the original diffusion model training in Eq. above.

For setting this architechture training, we denote the time-dependent discriminator as $D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t) : \mathbb{R}^N \times \mathbb{R}^N \times \mathbb{R} \to [0, 1]$, with parameters ϕ. It takes the N-dimensional $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ as inputs, and decides whether $\mathbf{x}_{t-1}$ is a plausible denoised version of $\mathbf{x}_t$. The discriminator is trained by:

$$\min_\phi \sum_{t\geq 1} \mathbb{E}_{q(\mathbf{x}_t)} \left[\mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}[-\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t)] + \mathbb{E}_{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}[-\log(1 - D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))] \right], \tag{3}$$

where fake samples from $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are contrasted against real samples from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The first expectation requires sampling from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ which is unknown. However, we use the identity $q(\mathbf{x}_t, \mathbf{x}_{t-1}) = \int d\mathbf{x}_0 q(\mathbf{x}_0) q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0) = \int d\mathbf{x}_0 q(\mathbf{x}_0) q(\mathbf{x}_{t-1}|\mathbf{x}_0) q(\mathbf{x}_t|\mathbf{x}_{t-1})$ to rewrite the first expectation term in Eq. 3 as:

$$\mathbb{E}_{q(\mathbf{x}_t) q(\mathbf{x}_{t-1}|\mathbf{x}_t)}[-\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))] = \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{t-1}|\mathbf{x}_0) q(\mathbf{x}_t|\mathbf{x}_{t-1})}[-\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))].$$

Given the discriminator, we train the generator by $\max_\theta \sum_{t\geq 1} \mathbb{E}_{q(\mathbf{x}_t)} \mathbb{E}_{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}[\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))]$, which updates the generator with the GAN objective decribe by **?**.

6.0.3 Redefining the implicit noise reduction model

Rather than explicitly estimating $\mathbf{x}_{t-1}$ during the noise reduction process, diffusion models (Ho et al., 2020) can be viewed as redefining the noise reduction model through $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0 = f_\theta(\mathbf{x}_t, t))$. First, the denoising model $f_\theta(\mathbf{x}_t, t)$ is used to predict $\mathbf{x}_0$, and subsequently, the posterior distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is employed to sample $\mathbf{x}_{t-1}$ based on $\mathbf{x}_t$ and the anticipated $\mathbf{x}_0$ (refer to Appendix **??** for further details). The distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$ intuitively represents the range of $\mathbf{x}_{t-1}$ values when reducing noise from $\mathbf{x}_t$ towards $\mathbf{x}_0$. For the diffusion process in Eq. **??**, this distribution consistently takes a Gaussian form, irrespective of the data distribution's complexity and step size (Appendix **??** provides the expression for $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$). In a similar vein, we establish $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as follows:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \int p_\theta(\mathbf{x}_0|\mathbf{x}_t) q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) d\mathbf{x}_0 = \int p(\mathbf{z}) q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0 = G_\theta(\mathbf{x}_t, \mathbf{z}, t)) d\mathbf{z}, \tag{4}$$

where $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ denotes the implicit distribution enforced by the GAN generator $G_\theta(\mathbf{x}_t, \mathbf{z}, t) : \mathbb{R}^N \times \mathbb{R}^L \times \mathbb{R} \to \mathbb{R}^N$ that generates $\mathbf{x}_0$ based on $\mathbf{x}_t$ and an L-dimensional latent variable $\mathbf{z} \sim p(\mathbf{z}) := \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$.

Benefits of redefinition: First, our $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ shares similarities with DDPM (Ho et al., 2020), which allows us to incorporate some inductive biases, such as network structure design, from DDPM. The primary distinction is that in DDPM, $\mathbf{x}_0$ is predicted as a deterministic function of $\mathbf{x}_t$, whereas in our case, $\mathbf{x}_0$ is generated by the generator using a random latent variable $\mathbf{z}$. This crucial difference enables our noise reduction distribution $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to become multimodal and complex, as opposed to the unimodal denoising model in DDPM. Second, it is important to note that for varying t values, $\mathbf{x}_t$ has different degrees of perturbation, which might make it challenging for a single network to directly predict $\mathbf{x}_{t-1}$ at different t. However, in our approach, the generator only needs to estimate the unperturbed $\mathbf{x}_0$ and then reintroduce perturbation using $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$. The visualization of our training pipeline is similar to

In summary, Generating synthetic data using DDPM (Denoising Diffusion Probabilistic Models) involves a two-step process: training the denoising model and sampling new data points.

- **Training the denoising model:**
  1. Initialize the generator network G_θ and the time-dependent discriminator network D_ϕ with random weights.
  2. Set the number of diffusion steps T to a small value (e.g., $T \leq 8$) and define a pre-determined variance schedule β_t for each step.
  3. For each epoch:
     (a) Sample a batch of real data $\mathbf{x}_0$ from the data-generating distribution $q(\mathbf{x}_0)$.
     (b) Perform the forward diffusion process using Eq. for $q\left(\mathrm{D\_flow}_t \mid \mathrm{D\_flow}_{t-1}\right)$ in original diffusion model, to obtain the noisy samples $\mathbf{x}_T$.
     (c) Train the generator G_θ to generate denoised samples $\mathbf{x}_{t-1}$ given the noisy samples $\mathbf{x}_t$ by minimizing the adversarial loss as described in Eq. 2.
     (d) Train the time-dependent discriminator D_ϕ using Eq. 3 to differentiate between real samples from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and generated samples from $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$.
  4. Repeat step 3 for the desired number of epochs.
  5. After training, use the learned generator G_θ to denoise samples in the reverse diffusion process.
  6. To generate new samples, first sample a random noise vector from the distribution $p(\mathbf{x}_T)$, and then perform the reverse diffusion process using the trained generator.
- **Sampling new data points i.e. Synthtetic Data Generation:** Once the Conditional GAN is trained, synthetic data can be generated by performing the reverse diffusion process:
  1. Ensure that the trained conditional generator G_θ and the pre-determined variance schedule β_t are available from the training process.
  2. Generate or obtain a set of conditioning variables C(here Conditioning variable would be Flow attributes in case you want to provide Flow Attributes to generate Flow Features), ensuring they match the format used during training.
  3. Sample a random noise vector $\mathbf{x}_T$ from the distribution $p(\mathbf{x}_T)$, which typically follows a Gaussian distribution with mean 0 and standard deviation 1.
  4. Initialize the generated sample $\mathbf{x} = \mathbf{x}_T$ and set the current time step $t = T$.
  5. For each time step t in reverse order $(T, T-1, \ldots, 1)$:
     (a) Concatenate the conditioning variables C with the current noisy sample $\mathbf{x}_t$ to form the input for the generator: $\mathbf{z}_t = [\mathbf{x}_t, C]$.
     (b) Use the trained generator G_θ to generate a denoised sample $\mathbf{x}_{t-1}$ given the concatenated input $\mathbf{z}_t$: $\mathbf{x}_{t-1} = G_\theta(\mathbf{z}_t)$.
     (c) Perform the reverse diffusion process using to obtain the denoised sample for the next time step: $\mathbf{x}_{t-1} = (\mathbf{x}_t - \sqrt{1-\beta_t} \cdot G_\theta(\mathbf{z}_t))/\sqrt{\beta_t}$.
  6. The final denoised sample $\mathbf{x}_0$ represents the generated data conditioned on the input C.
  7. Repeat steps 3-6 for each desired sample or set of conditioning variables C.

  The generated synthetic Flows will have similar properties to the original data, and can be used for various purposes like data augmentation, privacy-preserving data sharing, or enhancing the performance of machine learning models when the available data is limited.

7 Results (Conditional GAN based diffusion Models) and Limitations

The training for both models, specifically NetFlow and Conditional GAN-based Diffusion Models, was carried out using a limited dataset of 1000 flow traces from UGR16. We employed an A30 GPU to train both models, with each requiring approximately 20 minutes to complete the training process.

Upon examining Table-1 and the associated plots for single-feature distributional similarity, we can observe that our Conditional GAN-based Diffusion Model surpasses NetFlow in performance across all features. However, when scrutinizing Table-2 and the corresponding plots for single-attribute distributional similarity, we discover that the DDPM performs worse than NetFlow. This raises

the question of why the advanced and sophisticated training regime in the Conditional GAN-based DDPM does not yield superior generation results.

There is a straightforward explanation for the poorer performance on the Attribute Data: almost all the attributes are categorical, and during the forward process, i.e., $q\left(\mathrm{D_f low} t \mid \mathrm{D_f low} t-1\right)$, when we add noise to the original Flow Data, the categorical attributes receive a noise addition that transforms them into a continuous domain rather than the discrete one. This causes them to deviate from their underlying distribution. Since our model takes these continuous attribute noisy inputs instead of the desired discrete ones, it fails to learn their actual distribution (discrete). This is why the performance significantly deviates from that of NetFlow, which, during the attribute-based generation, takes in the discrete attributes. Further proof of this can be found in the fact that, as all the features in the Flow data are continuous, we can effectively learn their inherent data distribution and achieve good performance.

Our superior performance in single-feature distributional similarity serves as evidence that our DDPM architecture does work. However, its functionality is limited to continuous data generation. Given more time, we could have developed an architecture capable of handling noisy samples with categorical data or devised an alternative method to address this issue.

Table 1: Comparison of DDPM and NetFlow models based on single attribute distributional similarity

Attribute Type	**NetFlow Model (Best, Worst)**	**DDPM Model (Best, Worst)**
srcip	0.281 (0.000, 1.000)	0.666 (0.000, 1.000)
dstip	0.244 (0.000, 1.000)	0.284 (0.000, 1.000)
srcport	0.537 (0.000, 1.000)	0.819 (0.000, 1.000)
dstport	0.700 (0.000, 1.000)	0.792 (0.000, 1.000)
proto	0.258 (0.000, 1.000)	0.547 (0.000, 1.000)

Table 2: Comparison of DDPM and NetFlow models based on single feature distributional similarity

Feature Type	**NetFlow Model (Best, Worst)**	**DDPM Model (Best, Worst)**
td	25.409 (0.000, inf)	9.206 (0.000, inf)
pkt	1186.327 (0.000, inf)	1044.531 (0.000, inf)
byt	559951.111 (0.000, inf)	530550.662 (0.000, inf)
type	0.019 (0.000, 1.000)	0.019 (0.000, 1.000)
ts	39073456.532 (0.000, inf)	11363010.510 (0.000, inf)

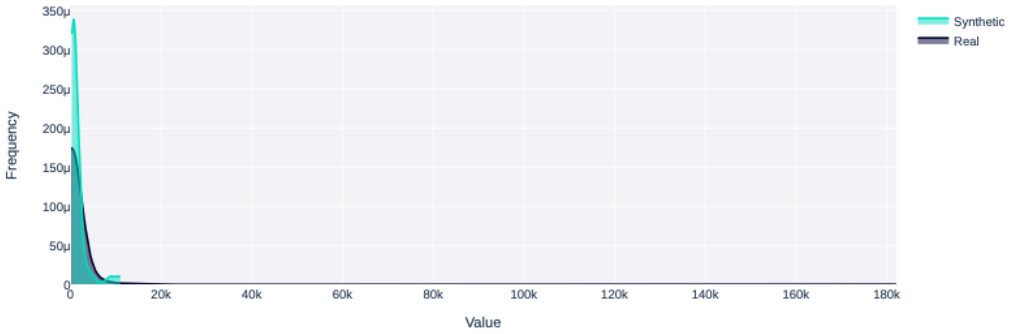


Figure 5: Single Feature distributional similarity of pkt for DDPM model (score-1044.53)

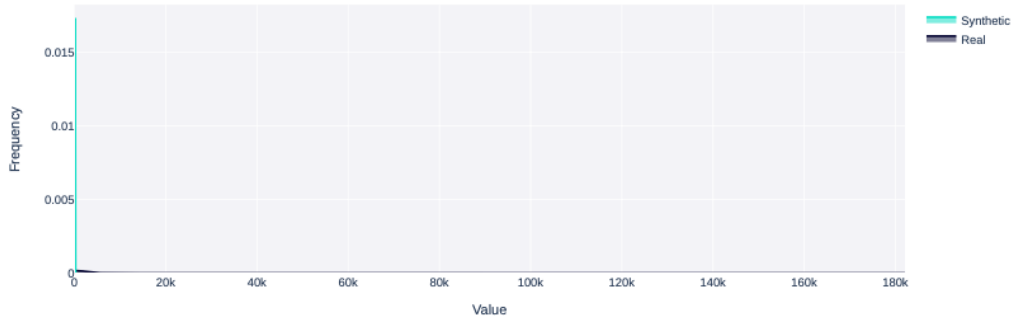


Figure 6: Single Feature distributional similarity of pkt for NetFlow model (score-1186.33)

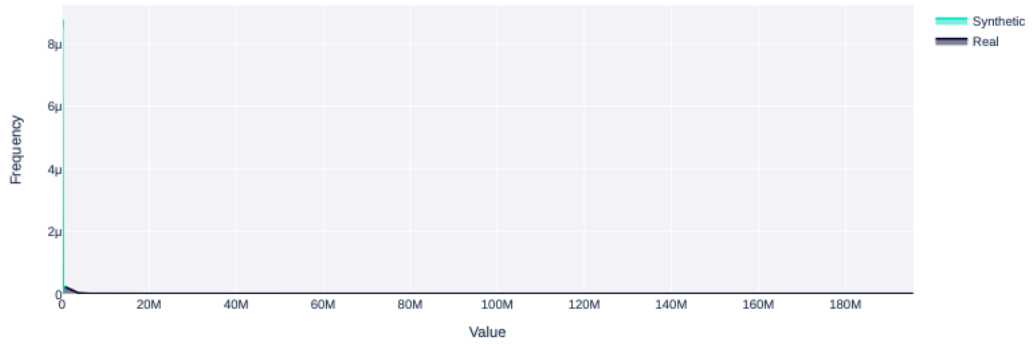


Figure 7: Single Feature distributional similarity of byte for DDPM model (score-530550.66)

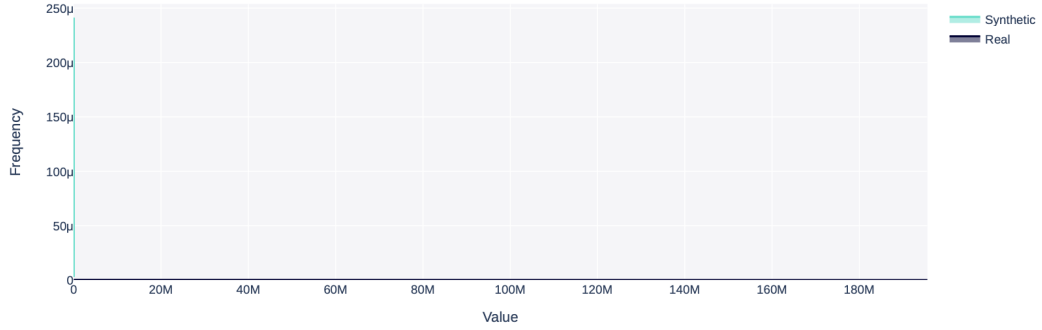


Figure 8: Single Feature distributional similarity of byte for NetFlow model (score-559951.11)

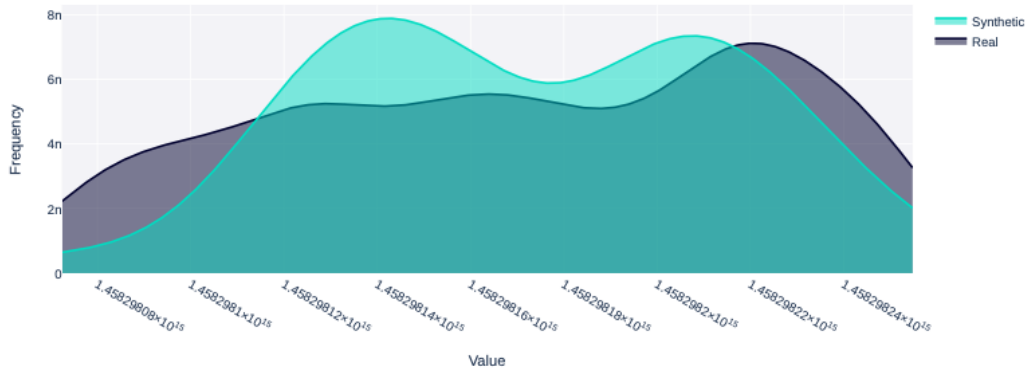


Figure 9: Single Feature distributional similarity of ts for DDPM model (score-11363010.51)

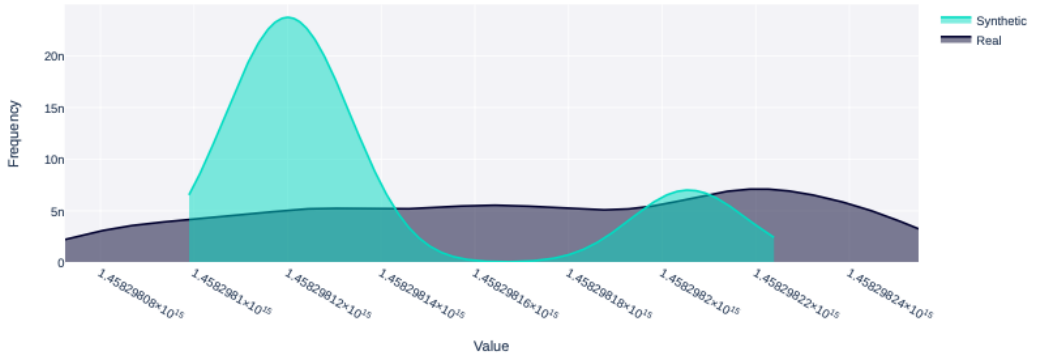


Figure 10: Single Feature distributional similarity of ts for NetFlow model (score-39073456.53)

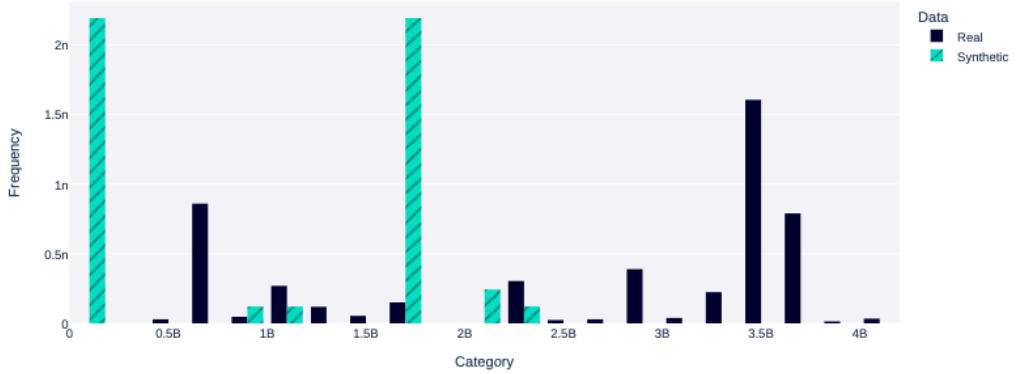


Figure 11: Single attribute distributional similarity of source IP for DDPM model (score-0.66)

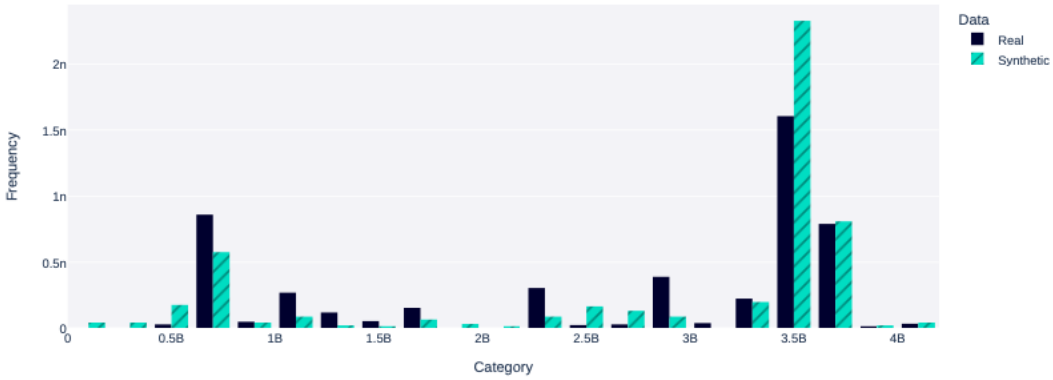


Figure 12: Single attribute distributional similarity of source IP for NetFlow model (score-0.28)

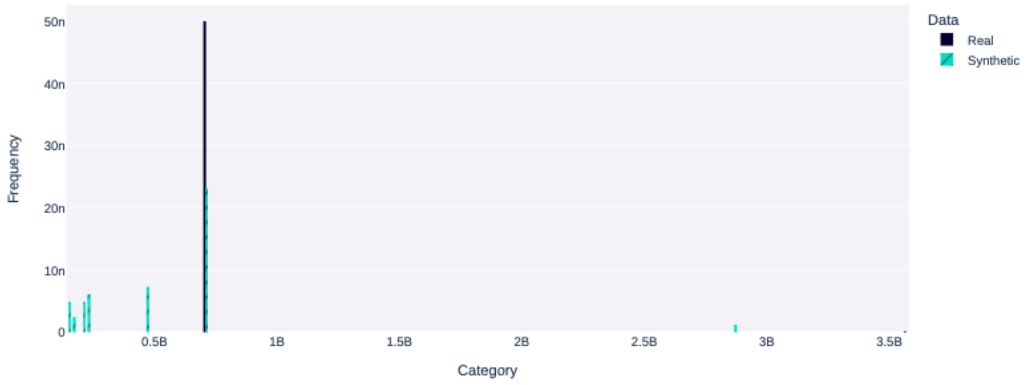


Figure 13: Single attribute distributional similarity of destination IP for DDPM model (score-0.28)

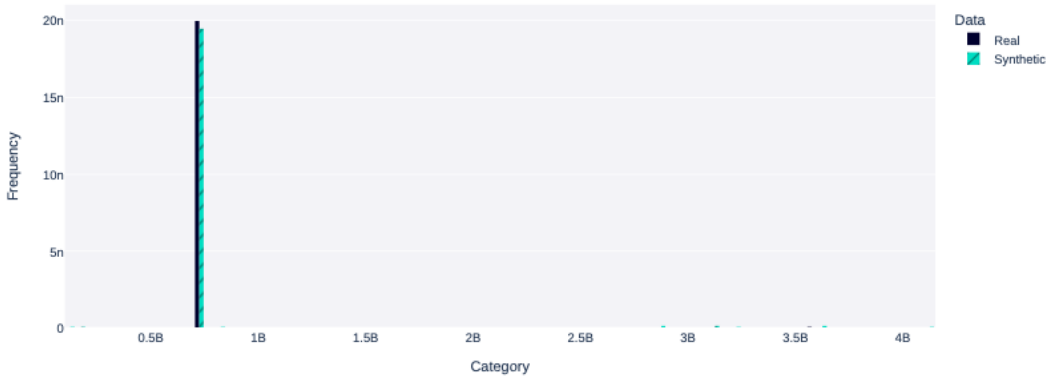


Figure 14: Single attribute distributional similarity of destination IP for NetFlow model (score-0.24)

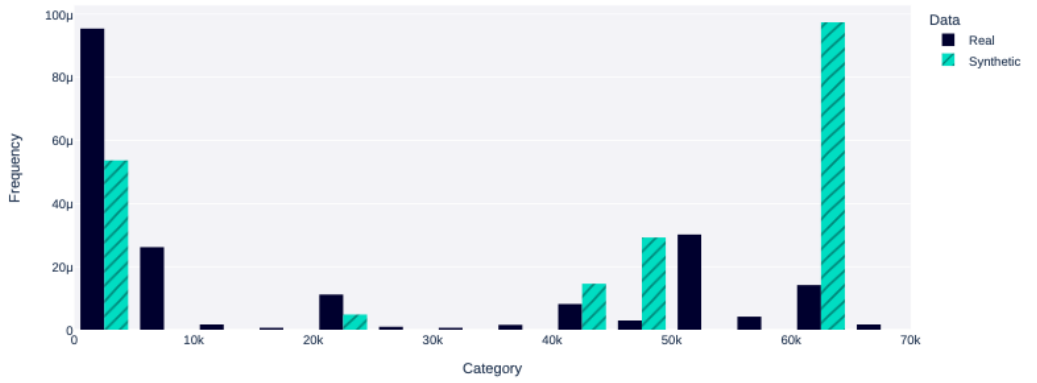


Figure 15: Single attribute distributional similarity of destination port for DDPM model (score-0.79)

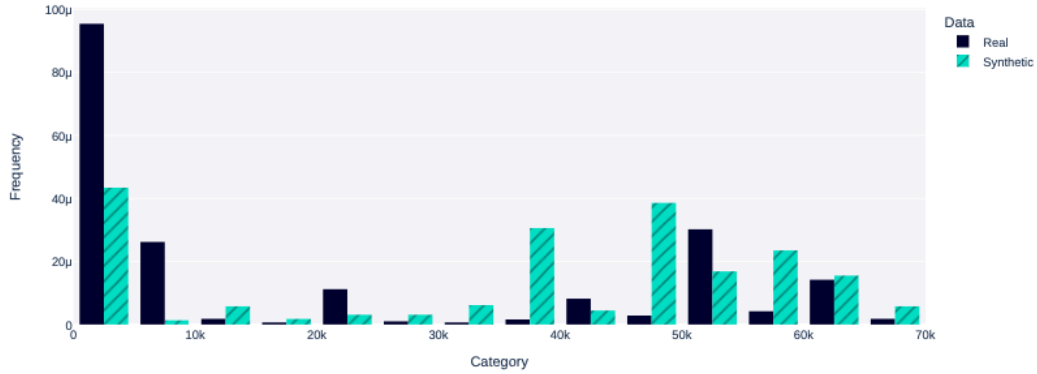


Figure 16: Single attribute distributional similarity of destination port for NetFlow model (score-0.70)

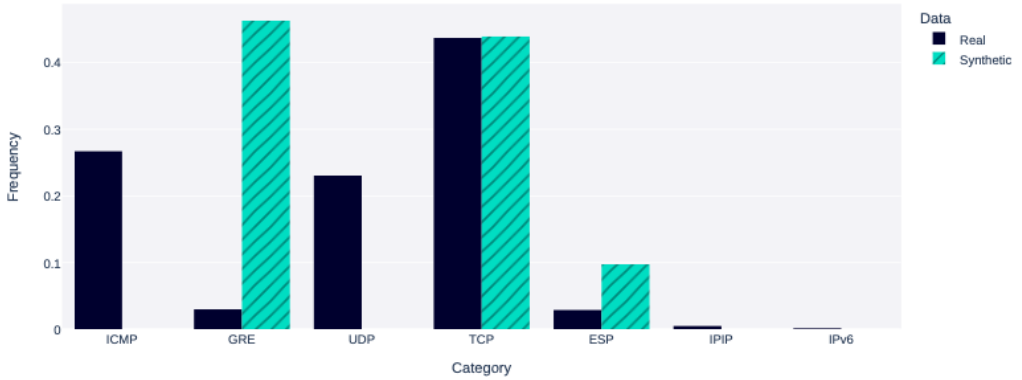


Figure 17: Single attribute distributional similarity of protocols for DDPM model (score-0.55)

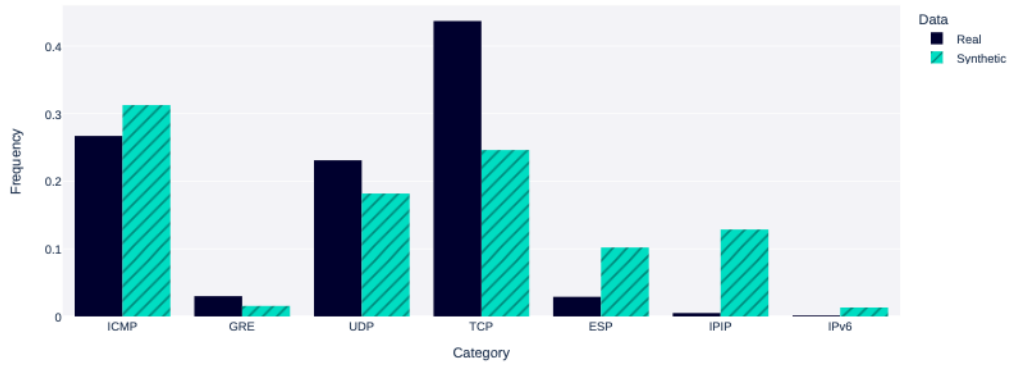


Figure 18: Single attribute distributional similarity of protocols for NetFlow model (score-0.26)

REFERENCES

Hanqun Cao, Cheng Tan, Zhangyang Gao, Guangyong Chen, Pheng-Ann Heng, and Stan Z Li. A survey on generative diffusion model. *arXiv preprint arXiv:2209.02646*, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pp. 464–483, 2020.

Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. Ugr '16: A new dataset for the evaluation of cyclostationarity-based network idss. *Computers & Security*, 73:411–424, 2018.

Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, pp. 8857–8868. PMLR, 2021.

C Walsworth, E Aben, K Claffy, and D Andersen. The caida ucsd anonymized internet traces. *Center Appl. Internet Data Anal., La Jolla, CA, USA, Tech. Rep*, 2015.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans, 2022.

Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pp. 458–472, 2022.

Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.