NetGlyph: Representation Learning to generate Network Traffic with Transformers

Gabin Noblet[†*], Cédric Lefebvre[*], Philippe Owezarski[†], William Ritchie[*]
[†]*LAAS - CNRS, Université de Toulouse, CNRS, Toulouse, France*
[*]*Custocy, Toulouse, France*
{gabin.noblet, philippe.owezarski}@laas.fr
{gnoblet, clefebvre, writchie}@custocy.com

***Abstract*—Network security has been a significant concern in recent years. Due to the rising number and complexity of cyber-attacks, Machine Learning (ML) models have been proposed to enhance intrusion detection. However, training these models requires extensive data, which is challenging to collect. To tackle this issue, previous work has focused on the generation of synthetic network traffic data using generative neural networks. Considering network traffic as a sequence of packets that contains continuous, discrete, and binary features, we propose a novel approach to learn a discrete representation of network traffic using Vector-Quantized Variational Autoencoders (VQ-VAE). In this paper, we adapt this model to learn how to represent network flows as a sequence of discrete tokens, called *NetGlyphs*. We evaluate the model on a dataset of *Command & Control* flows and compare performances to another model that uses a continuous representation. We show that our model is able to, reconstruct the data accurately and better preserve the original distribution. We also find promising results on network traffic generation using a state-of-the-art Transformer model to generate new *NetGlyphs* sequences that can be decoded back into real network traffic.**

***Index Terms*—Representation Learning, Autoencoder, Transformer, Generative Model, Network Security**

I. Introduction

The continuous growth and complexity of network traffic, along with the rise of sophisticated cyber-attacks, have emphasized the need for strong network security in today's digital environment. This has led to a growing interest in Machine Learning (ML) solutions applied to network traffic analysis and security. Despite their promising results on classification tasks and behavior analysis, the development of these models is still challenging due to the lack of realistic, updated, and diverse datasets. These data are hard to collect due to privacy concerns, and it requires a lot of time and resources to generate data by deploying enterprise-level infrastructures, performing attacks, and collecting data.

With this in mind, previous work focused on how to use generative models to generate realistic network traffic data that could be used to develop and test network security systems. Most of existing approaches have used Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs) to create new network traffic data. However, they differ in the way they represent the data. While some of them used flow-level features without capturing the packet-level information, others focused on packet generation without a proper flow context. Regardless of the chosen features used in network security solutions to analyze network traffic, they can be extracted from network capture, i.e. from a packet capture (Pcap) file.

In addition, the development of supervised models needs labeled datasets. Labeling is expensive, time-consuming, and requires a lot of expertise to classify the data correctly. Thus, the chosen network traffic representation should facilitate the labeling process while being able to reconstruct real network traffic in Pcap format.

In this paper, we suggest representing network traffic as a collection of bidirectional flows, where each flow is a series of packets. Each packet is represented by a set of different features dependent on the time and the direction in the flow. We believe that having a representation that captures the packet-level information is essential for the construction of realistic network traffic data in Pcap format and the flow-level representation is useful for labeling.

Network data is composed, in majority, of discrete and categorical features, thus we believe that a discrete representation of network traffic could be more efficient than a continuous one. We therefore propose a method to discretize network traffic by using representation learning with Vector-Quantized VAE (VQ-VAE), a variant of the VAE.

We modified the VQ-VAE to work with sequential data using Recurrent Autoencoders (RAE) and introduce FlowVQ-RAE. This model learns a way to transform a series of packets into a sequence of corresponding discrete vectors we called *NetGlyphs*. These *NetGlyphs* are learned during model training and regrouped in a codebook. We can thus represent network traffic as a sequence of *NetGlyphs*, similar to how words are represented in Natural Language Processing (NLP) tasks. Using an autoregressive model like the Transformer, we can generate new sequences of *NetGlyphs* that can be decoded back into packets. We show that FlowVQ-RAE better represents network data compared to a baseline model using continuous latent space. We also demonstrate that this approach generates network traffic with better precision than existing methods using GAN on a learned continuous representation.

This work was supported by ICO, Institut Cybersécurité Occitanie, funded by Région Occitanie, France.

978-3-903176-66-9 ©2024 IFIP

This study focused on malware *Command & Control* channel traffic to train and validate our approach. Real packet captures are available in the Stratosphere Laboratory (Czech Technical University in Prague) datasets [1]. The contributions of this paper can be summarized as:

- Proposing a new representation of network traffic as a collection of flows, where each packet is represented by a set of different features that allows reconstruction of a packet capture (Pcap) file.
- Introducing FlowVQ-RAE, a modified VQ-VAE, used to learn a discrete representation of network traffic (*NetGlyphs*), which showed good results in reconstruction assessment compared to continuous models.
- Using a state-of-the-art Transformer architecture to learn the distribution of *NetGlyphs* and generate new sequences that can be decoded back into real traffic flows.

In reference to hieroglyphs used in Ancient Egypt to represent ideas and concepts, we call codebook vectors used in the discrete representation of network traffic: *NetGlyphs*.

To the best of our knowledge this is the first time that Vector Quantized Variational Autoencoders are used to learn a representation of network traffic. And the first time that network traffic is represented as a multivariate time series composed of a collection of categorical and continuous attributes.

The paper is organized as followed: Section II reviews existing works on network traffic generation and presents the Vector Quantized Variational Autoencoders. Section III describes the chosen format and selected features to represent network traffic. Section IV introduces the compared models used for the representation learning. Section V presents the experiments along with network traffic generation. Finally, Section VI presents the results and discusses the experiments.

II. Related Work

A. Network Traffic Generation

In recent years, the success of generative models in various fields has led to an increased interest in using them to produce high-quality network traffic. Different approaches were adopted, some of them focused on flow-level features while others concentrated on packet-level generation.

Ring et al. used Generative Adversarial Networks (GAN) to create unidirectional flow-based data [2]. Based on Cisco *NetFlow* format [3], a flow is defined by the *5-tuple* that includes the *source IP address*, *destination IP address*, *source port*, *destination port*, and *transport protocol*. Metrics are associated to each flow like the number of packets, the total amount of bytes sent, the duration or the TCP flags for example. In their research, Ring et al. proposed three distinct methods to preprocess this heterogeneous data, composed of continuous, categorical and numeric data, thereby making it compatible with generative models.

In a similar approach based on flow-level features, Rigaki et al., proposed to use a GAN to generate flow metrics (duration, total bytes and inter-flow time) that mimic Facebook chat message traffic [4]. The authors demonstrated that GAN was able to generate realistic flow metrics that could be used to avoid detection of malware traffic by Network Intrusion Detection Systems (NIDS).

These studies were exclusively concentrated on flow metadata and metrics, which restricts their capacity to precisely replicate flow packets. Additionally, the use of unidirectional flow representation makes it difficult to consider the relationship between packets traveling in both directions, such as in client-server communications.

Other approaches in the literature focused on the generation of network packets. Taking advantage of the large capacity of GAN and Convolutional Neural Networks (CNN) in image generation, A. Cheng introduced a method to generate raw network packets using an image representation [5]. His approach consisted of representing a packet bytes stream into an image that allows the use of image-based generators to generate new, realistic packets. He successfully demonstrated the generation of realistic packets that can be transmitted over a network.

Dowoo et al. proposed PcapGAN [6], a method to generate realistic network traffic data using Style-Based Generative Adversarial Networks (Style-GAN) [7]. This method uses an encoder that decomposes Pcap data into IP graphs, time images, layer sequences, and option data, a generator that creates new data for these components, and a decoder that reconstructs the data into coherent Pcap files. The generated data is validated through tools like Wireshark and performance improvements in intrusion detection algorithms. The limit for this approach is that they do not consider a flow as we would like. They only consider the packets exchanged between two IP addresses, which is not sufficient to represent a flow.

Most of the existing works are separated in two categories. Firstly, the flow-level features generation, that does not represent accurately the distribution of packets within a flow. Secondly, methods that focus on network traffic generation at packet-level, but does not relate the generated packets to a flow entity that could be labeled as an attack procedure.

This conclusion was first made by Shahid et al. in their work on IoT network traffic generation [8]. In the paper they present a way to use GAN to mimic real IoT network traffic at packet level. They insist on the use of a hybrid traffic representation that combines the two approaches by considering bidirectional flows as sequences of packet sizes. However, they only consider the packet size as a feature, and that a packet sent is immediately followed by a packet received, i.e. that endpoints talk strictly one after another. In their context of IoT, they also consider the payload size as categorical data to avoid generating non-representative data that could lead to an easier detection of the generated traffic. To generate new sequences of packet sizes they used an autoencoder architecture with Long Short-Term Memory (LSTM) layers [9] to learn a continuous representation of their sequences of one-hot encoded packet sizes. A variant of GAN, the WGAN-

GP [10], is then used to generate new representations in the latent space that are then decoded back to sequences of packet sizes. Learning a continuous representation to then train a GAN is used to tackle the difficulties of generating sequence of discrete data with GAN [11].

Nevertheless, processing packet size as categorical data could be a problem if we want to consider all the possible values of the payload size in dataset augmentation purposes. Additionally, we need to consider the case where an endpoint sends multiple packets before receiving a response. Finally, they do not consider either the time distribution of packet in time, an important feature that could be used to identify traffic [12], [13], thus detect attack patterns.

B. Vector Quantized Variational Autoencoders

Vector-Quantized Variational Autoencoders (VQ-VAE) were first introduces by Van Den Oord et al. [14] in 2018. Their proposition was to modify the VAE architecture by adding a Vector Quantizer (VQ) to discretize the latent space, essentially transforming it into a VQ-VAE. This model excludes the layers necessary for distributional projection typically found in a standard VAE. The encoder produces a continuous latent representation that is then quantized using a codebook. The codebook is a collection of learned vectors that are used to represent the data in the latent space. The latent representation can then be reduced to a set of codebook vector indexes. The distribution of these codebook indexes can be learned using autoregressive models like PixelCNN [15] or Transformers [16]. The decoder then reconstructs the data from this quantized latent space.

This architecture has shown promising results on image generation, even in high-resolution with VQGAN architecture, a VQ-VAE coupled with a GAN discriminator to improve quality of reconstruction [17], [18].

III. Sequence of Packets

In this Section we present the chosen format and selected features to represent network traffic. We will first expose the context and motivation for our choice, then we will describe the selected features.

A. Background and Motivation

On their work on traffic characterization, in 2016, Draper et al. demonstrated that flow-based time-related features could be used to classify encrypted traffic over Virtual Private Network (VPN) and identify a type of application (e.g. browsing, streaming, etc…) [12]. In 2017, the same team shown similar results on encrypted traffic over Tor network [13]. This kind of analysis is only relevant on a flow or an aggregation of flows by computing metrics on the individual packets within those flows.

Considering this, our representation of network traffic must be dual, i.e. it should be able to represent a flow as a collection of packets. Representing a flow as a set of metrics could be inefficient for adaptability and compatibility with all possible network security systems that could need different metrics. In addition, the fact that the majority of network traffic is encrypted, we do not consider the payload content in our representation. However, we suppose the encrypted payload as random data, and therefore we consider its size as a key feature.

To summarize, generated traffic must be realistic. Its distribution, in terms of packet size, inter-arrival time, and direction must be similar to real traffic. We want to capture an accurate distribution of packets within a flow, from which we should be able to reconstruct a real packet capture file (Pcap). This format is essential for maximum compatibility with existing and future network security systems based on flow metrics analysis.

B. Traffic Representation and Selected Features

Network traffic data is composed of an ensemble of packets that have multiple features. These features can be divided into two categories: flow-level features and packet-level features.

Flow-level features are metadata that describe the endpoints of the communication, such as the source and destination IP addresses, the source and destination ports, and the protocol used. These five features are known as the *5-tuple*, they are used to identify a flow as they are common to all packets in a same flow. The flow is bidirectional, so the *5-tuple* can be used to identify the flow in both directions by reverting source and destination features. This allows us to account for the relationship between packets traveling in both directions between the endpoints.

In addition to these, we have the total number of packets, bytes, the total duration, etc…

Packet-level features are the characteristics of each packet in the flow. Depending on the protocol, these features can be different. However, we can identify three main features that are common to all protocols. The inter-arrival time: it is the time spent since the last packet in the flow was encountered. The direction: it is used to identify the sender of each packet. Then, the packet size, which is the total size of the packet in bytes. The advantage of working at flow level is that we can consider only the payload size as a feature instead of the total packet size. That way, we avoid considering the header size, which is protocol-dependent and avoid generation of short packet size (lower than header size).

Additionally, our work focuses on Transmission Control Protocol (TCP) traffic, well known as a connection-oriented protocol, that guarantees the delivery of packets in the correct order. To do this, TCP uses a set of features used to control the connection. They are exchanged in the packet's header during the communication.

We believe that considering all of these features in our representation is counterproductive, as it could lead to many errors. For example, generate counters like sequence number and acknowledgment number could be inaccurate with the size of the payload, but can be derived from it. Other features like TCP flags are important to consider. Indeed, they are used as control signals to establish, maintain, and terminate the connection. Famously, the SYN flag (Synchronize) is used

to initiate a connection, the ACK flag (Acknowledgment) is used to acknowledge the receipt of a packet, and the FIN flag (Finish) is used to properly terminate the connection. Other flags are relevant like the PSH flag (Push), used to notify the application that there is data to be read (e.g. a chat message) or the RST flag (Reset), used to reset the connection (abrupt termination).

Finally, other features that focus on congestion control, or additional options, are not considered in this study. We suppose that the traffic is well-formed and fully protocol-compliant, i.e. there are no malformed packets or protocol misuse.

To summarize, the features relevant to this study for packet representation are shown in Table I. We used the three main features along with the TCP flags. As we can see in the table, the type of each feature is different. The inter-arrival time (IAT) is continuous while payload size is considered a numerical feature (it is usually set between 0 and 1460). The direction, is represented as a binary attribute depending on the direction of the packet in the flow (forward or backward packet). We represented flags not as single features but as combinations of flags. That way, we ensure to avoid errors in the usage of flags. For example, we want to avoid forbidden combinations like SYN and FIN in the same packet. They are categorical features that are one-hot encoded to be processed by ML.

TABLE I
LIST OF PACKET FEATURES USED IN NETWORK TRAFFIC REPRESENTATION

	Type	Example
IAT (Inter Arrival Time)	*Continuous*	$1.389s$
Payload size	*Numeric*	388
Direction	*Binary*	0 (forward)
Flags	*Categorical*	PA (Psh/Ack)

To conclude, we propose a representation of network traffic as a collection of packet features belonging to a bidirectional flow. This flow is identified by the same *5-tuple*. The selected features allow us to reconstruct a packet in a Pcap format. The main features (IAT, Direction and Payload size) are used to capture the distribution of traffic in the flow. The TCP flags are used to identify the different steps of the connection. Together with the main features, we can compute additional TCP attributes and reconstruct a valid packet header. This approach can be seen as an extension of the work of Shahid et al. [8] on IoT traffic generation.

IV. REPRESENTATION LEARNING MODELS

The following Section presents the compared models used for the representation learning of network traffic. As discussed in Section III, network traffic data is represented as a sequence of packet features. These features are heterogeneous, composed of continuous, categorical, and discrete data. The chosen approach for generation is to first learn a representation of the data, then train a generator to produce new instances in this learned latent space that can be decoded back to the original data format.

Representation learning can be done using the autoencoder architecture. It is composed of an encoder that learns a projection of the data in a given latent space, and a decoder that reconstructs the data from this representation. Considering the sequential nature of the data, we use a Recurrent Autoencoder (RAE) architecture (Fig. 1). This autoencoder is composed of Recurrent Neural Networks (RNN) layers that process sequences iteratively and maintain a hidden state that captures the context of the sequence. These models are well-suited for sequential data.

The chosen RNN layers are Long Short-Term Memories (LSTM) layers. They are known to learn long and short-term dependencies in sequences. These layers are used in both the encoder and decoder blocks for this study. Differences between the compared models rely on the latent space representation. The baseline model, FlowRAE, learns a continuous representation of packet sequences. It performs a sequence-to-vector transformation. The proposed model, FlowVQ-RAE, learns a discrete representation of the sequence of packets, a sequence of packets is transformed into a sequence of indexes, corresponding to learned vectors in a codebook.

Through this work, we want to know if VQ-VAE is a suitable approach for representation learning on network data.

A. Flow RAE - Baseline model for comparison

Based on the work of Shahid et al. [8], we use a Recurrent Autoencoder (RAE) to transform our sequence of packet features into a single vector. The encoder is composed of LSTM layers that process the sequence of packets. These layers can either return the last output of the sequence or the full sequence. In the case of sequence-to-vector, we use the last vector of the encoder output as latent representation. This last vector is meant to retain the context of the sequence as it is computed considering all previous vectors in the sequence. The decoder then tries to reconstruct the sequence of packets from this latent representation performing a vector-to-sequence transformation. Similarly, the decoder uses LSTM layers to distill the latent representation into a sequence of packets.

The structural limitation of this model is the non-consistency of the flow length in the model. As we keep only the last vector of the encoder output, the flow length is lost during the flow processing by the model.

To process sequences with this kind of model, we must fix a sequence length N for the sequences and apply padding and truncation to the input data. In the model architecture, the decoder needs to know N to reconstruct the sequence. The model is limited, by design, to reconstruct sequences of fixed length N. It is unable to produce sequences of length greater than N, thus we should set N big enough to handle long sequences. This leads to shorter sequences requiring a lot of padding that the model will process uselessly.

In our case we set the flow length N to 32 packets, which corresponds to almost 98% of our dataset. As the input sequences are padded with zeros, we detect the end of a flow

by monitoring the *l2-norm* of decoder output vectors. If the value falls below a certain threshold, it is interpreted as the end of the flow (similar to a noise threshold in signal processing). This provides a mechanism to handle the variable length of flows.

B. FlowVQ-RAE

Despite the continuous inter-arrival time between packets, network data is composed of a lot of categorical and discrete features. Considering this, we believe it would be easier trying to learn discrete data representations, using VQ-VAE architecture, rather than a continuous representation.

Based on the FlowRAE model, we modify the encoder to output the full sequence of vectors rather than a single vector. Thus, latent representation is a sequence. The first benefit of this, is that flow length is kept along the model, allowing the model to efficiently process sequences of different lengths. The decoder is also modified to reconstruct the sequence of packets from the latent sequence. We call this model FlowVQ-RAE.

For a flow F composed packets P_i, the encoder produces a continuous latent sequence Z_e of corresponding vectors z_{e_i}. As we can see in Fig. 1, vector quantization is applied to discretize the latent sequence Z_e into sequence Z_q. Each vector z_{e_i} is mapped to the closest vector e_k in a codebook, s.t. $z_{e_i} \approx e_k$ as follows:

$$z_{q_i} = e_k, \quad \text{where} \quad k = \ argmin_j ||z_{e_i} - e_j|| \tag{1}$$

The quantized latent sequence Z_q is then used as input to the decoder that reconstruct the flow F' as sequence of packets P'_i. Using this model, a sequence of packets is associated to a list of discrete vectors as a set of codebook indexes. Sampling new indexes sequences from this latent space can be done using autoregressive models, like Transformers. Once the sequence of indexes is generated, it can be decoded back to network data using the decoder.

A limitation of this model is that it suffers from low codebook usage (Table II). This is usually due to a poor codebook vectors initialization.

C. Codebook Improvement

To improve the usage of the codebook, we implemented a similar method to that proposed by Yu et al. [18]. In this method we normalize and reduce the dimension of the codebook. The *l2-norm* constraint is applied to the encoder output and codebook vectors. The reduction of dimension in the latent space is done by adding a fully connected layer to the encoder output applied to each vector in the sequence. This reduces the dimension of the sequence and the codebook vectors. A similar layer is added to the decoder input to retrieve the original dimension. Yu et al. [18] proposed in their paper to apply linear projection to the latent space. However, we experienced that applying hyperbolical tangent activation function to the fully connected layer output helps in training stability of the codebook.

In this work, the model with enhanced codebook is called FlowVQ-RAE-EC.

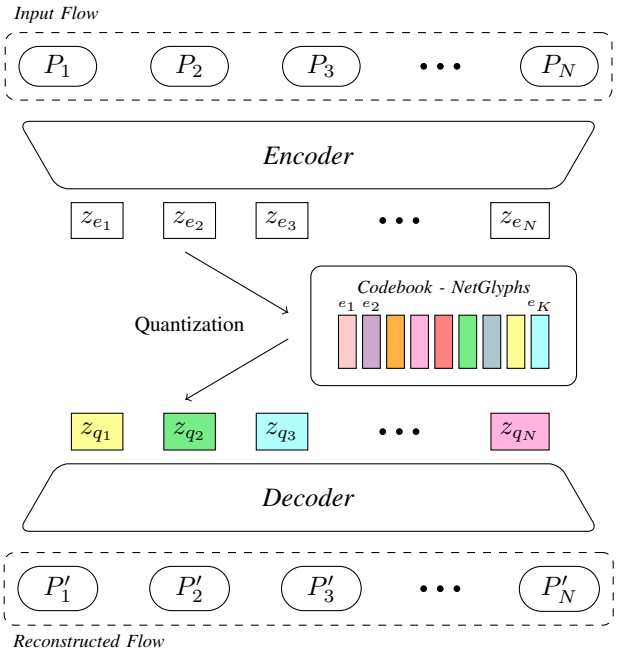

Fig. 1. Flow VQ-RAE model architecture. Input flows are cut up into individual packets, $F = \{P_1, P_2, \ldots, P_N\}$. The **Encoder** transforms this sequence of packets into a sequence of continuous vectors $\{z_{e_1}, z_{e_2}, \ldots, z_{e_N}\}$ corresponding to each packets. These vectors capture the dependencies between input packets. The **Quantization** layer then maps each continous vector to the nearest vector from a codebook. Mapping continuous data to discrete data allows us to better generate flows which are by nature discrete. For the purpose of this study we call these discrete vectors ***NetGlyphs***. The **Decoder** then reconstructs the input flow F' from the sequence of NetGlyphs using LSTM layers.

V. Experiments

In this Section we present the conditions of the experiments. We first want to evaluate the representation learning, we want the model to learn a meaningful latent space representation, i.e. can it reconstruct the data with a good accuracy? We then use generative models to generate new latent representations and compare to existing methods.

A. Dataset

We evaluated our system on data provided by Stratosphere Laboratory at Czech Technical University in Prague (CTU) [1]. The selected data contains approximately 500k flows of *Command & Control* (C&C) traffic of presumed *Trickbot* malware. It consists of packet captures of real network traffic. The malware is deployed in a controlled environment and the traffic is captured. As we focused on TCP flows, we filtered the dataset to keep only TCP traffic from the original dataset.

We build a tool to extract the data into the specified format in Section III. It takes a packet capture file as input, identifies the different flows and extracts the features of each packet. Because there is some reused ports and IP addresses, we use the first sequence number as sixth criteria to parse flows. The dataset is split into a training set composed of 80% of network flows. We keep the remaining 20% for tests and evaluation.

TABLE II
COMPARATIVE METRICS FOR FEATURE RECONSTRUCTION ON NETWORK TRAFFIC

		Flow RAE	Flow VQ-RAE	Flow VQ-RAE with Enhanced Codebook			
Latent shape		(32)	(32, 12)	(32, 6)	(32, 4)	(32, 3)	(32, 2)
IAT	Wasserstein	0.0131	0.0081	0.0037	0.0043	0.0055	**0.0018**
Payload	Distance	0.0536	0.0046	0.0021	0.0034	**0.0019**	0.0020
Direction	Error	0.53%	0.1%	**0%**	**0%**	**0%**	**0%**
	F1-Score	0.995	0.999	**1.0**	**1.0**	**1.0**	**1.0**
Flags	Error	0.47%	0.22%	<0.1‰	**0%**	**0%**	**0%**
	F1-Score	0.983	0.881	>0.999	**1.0**	**1.0**	**1.0**
Codebook Usage		*N/A*	1.76%	10.35%	23.24%	58.50%	**85.74%**

B. Preprocessing

As discussed in Section IV, RNN require fixed-size sequences to be processed. We choose to fix the flow length at 32 packets because it represents at most 98% of the dataset. The longer flows are truncated, and shorter ones are padded with zeros. Note that padded data carry a mask to indicate where the padding is located. It allows models that are consistent with the sequence length, like the FlowVQ-RAE, to skip processing on padded data.

The packet features are normalized in the range $[0, 1]$ prior to training. The normalization on the payload size and inter-arrival time (IAT) is done using Quantile Transformer from Scikit-Learn [19] to get a normal distribution and applying min-max scaling to get the values in the range $[0, 1]$. Direction does not need to be normalized as it is a binary attribute. Furthermore, we apply one-hot encoding to flag combinations after removing unwanted flags used for congestion control.

After this step the variable-length sequences of 4 features are transformed into a sequence of 32 vectors of 11 features that regroup IAT, Direction, Payload size and 8 categories for flags.

C. Model size

To compare the two architectures, both models, FlowRAE and FlowVQ-RAE, share a common architecture of two LSTM layers of 12 units in the encoder and decoder. This number of units is close to the size of sequence vectors (11 in this case) and can be easily divided for dimension reduction analysis in the latent space. A Dense layer is applied to each member of sequences (time-distributed layer) at the output of the decoder to get the original dimension of the data (11 units) and a sigmoid activation function is applied to get the values in the range $[0, 1]$. Nevertheless, the models differ in the latent space representation:

FlowRAE has an LSTM layer of 32 units at the encoder output that returns the last vector of the sequence. The latent representation is a vector of dimension 32. The decoder is preceded with a specific layer that repeats the latent vector 32 times to regain the sequence of 32 vectors. We denote the latent shape (32) as it is a single vector of dimension 32.

FlowVQ-RAE produces sequences of 32 vectors of dimension 12 at the encoder output. This latent representation is then quantized using a codebook of 1024 vectors of dimension 12. The decoder takes as input the sequence of discrete vectors directly. We denote the latent shape (32, 12) as it is a 32-vectors sequence composed of dimension 12.

FlowVQ-RAE-EC has the same structure of FlowVQ-RAE with additional codebook improvements. The encoder output is followed by a time-distributed Dense layer that reduces the dimension of the latent space. It is also followed by a time-distributed UnitNorm layer that normalizes the latent sequence latent vectors. The codebook is then a collection of 1024 vectors, with unit-normalization. Finally, the decoder is preceded by a time-distributed Dense layer of 12 units that restores the original latent space dimension. We present four latent vectors dimension reductions to compare their efficiency in codebook usage. The latent shape is respectively (32, 6), (32, 4), (32, 3) and (32, 2).

D. Training

The six models were trained on the same dataset during 300 epochs. The optimizer used was Adam [20], with a learning rate set to 2×10^{-4}. The loss function used was the mean-squared error. Learning codebook vectors is done through an additionnal loss function defined in the first work on VQ-VAE [14] as follow:

$$L_{VQ} = ||sg[E(x)] - e||_2^2 + \beta||E(x) - sg[e]||_2^2 \qquad (2)$$

The first term is used to minimize the distance between the encoder output $E(x)$ and the codebook vector e. The second term is known as commitment loss. It is used to force the encoder to produce outputs close to codebook vectors. The β parameter is set to 0.25 as suggested in the original work [14]. $sg()$ is the stop gradient function that prevents the gradient from backpropagating. Therefore, the codebook vectors are optimized by the first term only, and the encoder optimizes the second term.

E. Evaluation methods

Our data is made of different types of features, such as numerical, categorical, and binary. We evaluate the models using different methods adapted to these types of features (Table II). We refer to the test data, as the original data, and the reconstructed data is the test data after being processed by

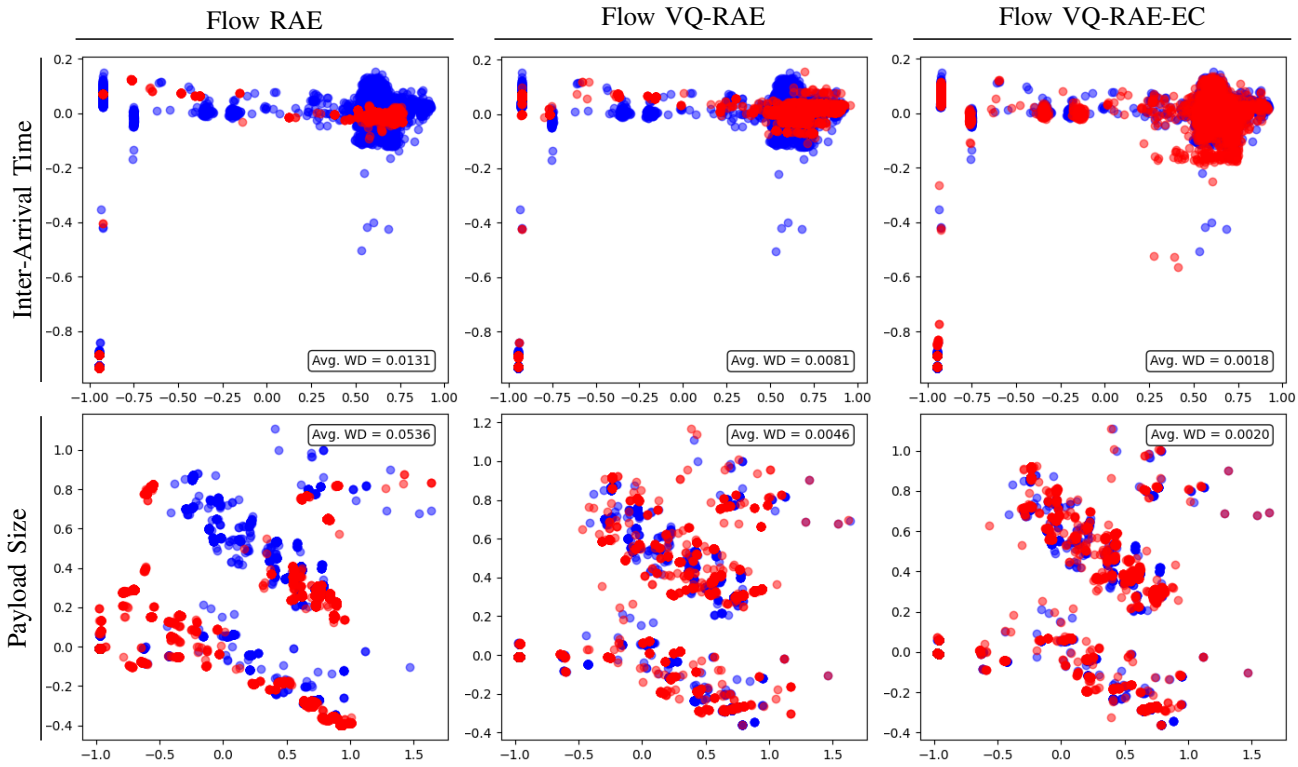

Fig. 2. PCA plots for reconstruction analysis of inter-arrival time (IAT) and payload size features for three different models. The plots show the distribution of the original data (blue) and the reconstructed data (red) for three models. FlowRAE, is the baseline model, using a single vector as continuous latent representation. FlowVQ-RAE architecture discretizes the latent representation using a codebook. FlowVQ-RAE-EC, enhances the codebook usage by normalization and reducing the codebook dimension. The results show that the FlowVQ-RAE-EC model is able to reconstruct the data with a distribution close to the original data, thus more accurately than the other models. In the IAT plots, principal components 1 and 2 represented on axes x and y explain 93.86% and 3.48% of the variance respectively. For Payload Size plots, they explain 81.69% and 7.40%, respectively.

the model. We expect the reconstructed data to be close to the original data.

1) Reconstruction metrics: To compare the distribution between original and reconstructed data on IAT and Payload size we calculate the Wasserstein distance. Also known as the Earth Mover's Distance, it represents the minimum amount of work required to transform one distribution into the other. A good reconstruction would lead to a Wasserstein distance close to 0, meaning that the distributions are close. The value for this distance on each feature is shown in Table II.

For categorical and binary data, we use the F1-score and an error rate to evaluate the model. We do not compare sequences to sequences like in PCA analysis, but we compare the values individually, e.g. the i^{th} flag of the n^{th} test flow must be the same as the i^{th} flag of the n^{th} reconstructed flow. Table II presents the results for these metrics for each tested models. The error rate is the percentage of misclassified data, i.e. the sum of false negatives and false positives, the closest to 0 the better. The F1-score is a metric used in classification models analysis, the closest to 1 the better. This evaluation is done on the flags and the direction features.

2) Codebook usage: For FlowVQ-RAE models, we want to assess the codebook usage. Codebook usage is represented as a percentage of codebook vectors used to represent the test data. We count the number of different *NetGlyphs* used to represent the test dataset, against the total number of vectors in the codebook. The closer to 100% the better. The results are shown in Table II.

3) Principal component analysis (PCA): The Principal Components Analysis (PCA) is well suited for continuous or numerical data. We use it to analyze the distribution of IAT and payload size. The features are analyzed in their normalized format to have better visualization. The results are shown in Fig. 2 where each point is a projection of a sequence. In blue, we can see the distribution of the original data and in red, the distribution of the reconstructed data. We expect the points of the reconstructed data to recover the distribution of the original data. This visual method allows us to see if the model is able to reconstruct the data correctly and if the distribution of the data is preserved.

F. Generation of new flows

Having learned a discrete representation of network packet sequences, we now wish to create new flows by generating discrete sequences of *NetGlyphs*. Generation of discrete sequences can be done using autoregressive models. Literature supports the use of Transformers with VQ-VAE architecture, highlighting their ability to generate high resolution images [17], [18]. We use then a state-of-the-art Transformer model

to generate new sequences of *NetGlyphs* indices that can be decoded back to network data.

The chosen model is a model derived from the first architecture proposed by Vaswani et al. [16]. It is composed of 6 layers of multi-head attention with 4 heads. The model is trained on the full dataset transformed into sequences of *NetGlyphs* indexes by the FlowVQ-RAE-EC model with a compression dimension of 2. The model is trained for 100 epochs. The loss function used is the sparse categorical cross-entropy with a scheduled learning rate as proposed in [16].

The generation of new sequences is done by iteratively predicting the next *NetGlyph* index using the previously generated indexes. The model stops generating when the end of the sequence token is predicted. The generated indexes are then decoded back to network data using the decoder of the FlowVQ-RAE-EC model. Because all first packets in a flow are the same, they all correspond to the same *NetGlyph* index. We use this index to initialize the generation of a new sequence.

We compare this approach to existing literature. Based on the work of Shahid et al. [8], we use the FlowRAE that learned a continuous representation of packet sequences coupled with a Wasserstein GAN with Gradient Policy (WGAN-GP) model [10] to generate new packet sequences. WGAN-GP is a variant of GAN that improves training stability. It produces new continuous representations reconstructed into network data using the FlowRAE decoder.

The evaluation of generated sequences is done using PCA analysis for IAT and payload size distribution. Additionally, we analyze the generated sequences by verifying the correct protocol usage.

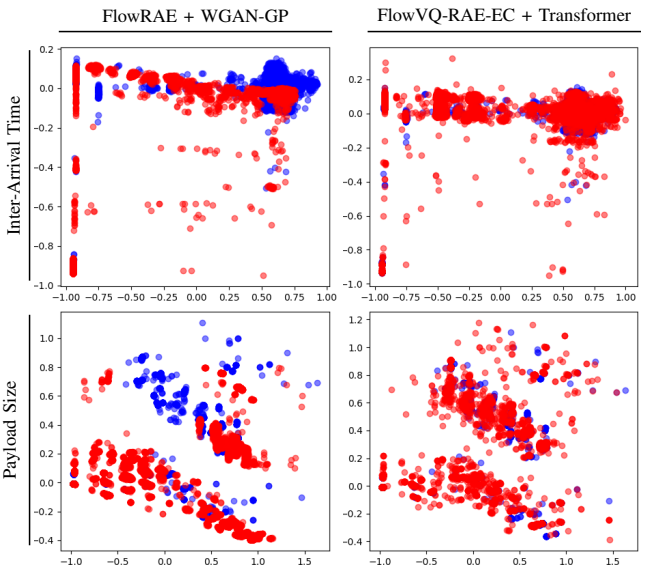

Fig. 3. PCA plots for generated flows analysis of inter-arrival time (IAT) and payload size features for two different generative models. The plots show the distribution of the original test data (blue) and the generated data (red). On the left, a WGAN-GP model is used to generate new latent vectors decoded back to packets with FlowRAE, used as baseline. On the right, a Transformer decoder-only architecture is used to generate new sequences of *NetGlyphs* decoded back to packets with FlowVQ-RAE-EC, our proposed approach. The results show that, using a WGAN-GP that learns continuous latent vectors decoded with the FlowRAE, results in generated data that could not fit the original distribution. On the other side, the FlowVQ-RAE-EC model coupled with a Transformer can generate data with a distribution of IAT and Payload Size close to the original data. In the IAT plots, principal components 1 and 2 represented on axes x and y explain 93.86% and 3.48% of the variance respectively. For Payload Size plots, they explain 81.69% and 7.40%, respectively.

VI. Results and Discussions

A. Network Flow reconstruction

The results for all metrics presented in Table II show that the FlowVQ-RAE architecture with its discrete latent representation outperforms the baseline model that uses a continuous latent space. The accuracy on categorical features is higher for FlowVQ-RAE than for FlowRAE, reaching perfect reconstruction for some versions on flags and packet direction in the flow. Additionally, the adjustment made to improve codebook usage proved their efficiency as the codebook usage reach 85.74% against 1.76% for the FlowVQ-RAE without these modifications.

The Wasserstein distance between the original and reconstructed data distribution is also decreased with our solution. This results in a better representation of the original distribution. The PCA analysis shown in Fig. 2 confirms this result by showing a better covering from red dots (reconstruction data) over blue ones (original data). The distribution of continuous and discrete data is better preserved with FlowVQ-RAE than with FlowRAE.

Considering these results, we chose to keep the FlowVQ-RAE with Enhanced Codebook and a compression dimension of 2 as the best model for our study. Its higher codebook usage is preferred to its higher Wasserstein distance than the one with a compression dimension of 3.

B. Traffic Generation

Using the two generative approaches, we generated 100,000 new flows. The analysis of IAT and Payload size distributions are shown in the PCA plots in Fig. 3. These plots show the distribution of the original test data in blue, and the generated data in red. We can see that IAT and Payload size distributions are not perfectly preserved with the use of WGAN-GP and a continuous latent space (FlowRAE). Nevertheless, generated data using our approach better fits the original data distribution. Because the WGAN-GP generates accurate latent codes, this result is due to a poor representation of distribution data in the continuous latent space.

In addition, the *3-way hanshakes* were successfully generated without error in 99.99% of the cases with both models. They also generated correct connection attempts flows by early termination (SYN/RST) or timeout (two SYN replay). Finally, the two approaches differ on the second aspect. Our approach using Transformer verify the rule of a packet with a PSH flag having a non-zero payload size at 98.5%, against 88.6% for the baseline model.

Some flows have a correct termination, although it is challenging to verify. It is recommended that future work should consider the use of network tools to assess the conclusion of communication.

VII. Conclusion and Future Works

In this paper, we presented FlowVQ-RAE, a model used to learn a discrete representation of network traffic. This model is based on the Vector Quantized Variational Autoencoder (VQ-VAE) architecture. We adapted its architecture to work with sequential data using LSTM layers. The model learns a way to transform a series of packets into a sequence of corresponding discrete vectors we called: *NetGlyphs*. These *NetGlyphs* are learned during model training and regrouped in a codebook. This representation allows the use of a Transformer to generate new sequences of *NetGlyphs* that can be decoded back into packets. FlowVQ-RAE analysis showed it has better results in discrete representation learning for network data, compared to a baseline model using continuous latent space.

Additionally, the generation of individual network flows using this discrete representation demonstrates promising results. Protocol usages like the *3-way handshake*, has been successfully replicated. Additionally, a variety of different flows, including connection attempts, have been generated. Our generative model showed better accuracy for protocol usages like the *Psh* with non-zero payload size. The model also generated flows with a similar distribution of the original data.

For future work, we should train this model with more diverse data including different class of traffic like benign browsing or streaming along with other attack techniques. We should also consider extend *NetGlyphs* to include more features and protocols. For the assessment of the generated data, we should use more metrics and rules to verify the generated flows, especially to assess the end of communication. Finally, additional work should be done on the context or attack scenario that links several individual flows together.

References

[1] Stratosphere, "Stratosphere Laboratory Datasets," 2015. [Online]. Available: https://www.stratosphereips.org/datasets-overview

[2] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using Generative Adversarial Networks," *Computers & Security*, vol. 82, pp. 156–172, May 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404818308393

[3] B. Claise, "Cisco Systems NetFlow Services Export Version 9," Internet Engineering Task Force, Request for Comments RFC 3954, Oct. 2004, num Pages: 33. [Online]. Available: https://datatracker.ietf.org/doc/rfc3954

[4] M. Rigaki and S. Garcia, "Bringing a GAN to a Knife-Fight: Adapting Malware Communication to Avoid Detection," in *2018 IEEE Security and Privacy Workshops (SPW)*. San Francisco, CA: IEEE, May 2018, pp. 70–75. [Online]. Available: https://ieeexplore.ieee.org/document/8424635/

[5] A. Cheng, "PAC-GAN: Packet Generation of Network Traffic using Generative Adversarial Networks," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. Vancouver, BC, Canada: IEEE, Oct. 2019, pp. 0728–0734. [Online]. Available: https://ieeexplore.ieee.org/document/8936224/

[6] B. Dowoo, Y. Jung, and C. Choi, "PcapGAN: Packet Capture File Generator by Style-Based Generative Adversarial Networks," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. Boca Raton, FL, USA: IEEE, Dec. 2019, pp. 1149–1154. [Online]. Available: https://ieeexplore.ieee.org/document/8999252/

[7] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4217–4228, Dec. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/8977347/

[8] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative Deep Learning for Internet of Things Network Traffic Generation," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. Perth, WA, Australia: IEEE, Dec. 2020, pp. 70–79. [Online]. Available: https://ieeexplore.ieee.org/document/9320384/

[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, _eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," Dec. 2017, arXiv:1704.00028 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1704.00028

[11] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," Aug. 2017, arXiv:1609.05473 [cs]. [Online]. Available: http://arxiv.org/abs/1609.05473

[12] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic using Time-related Features:," in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*. Rome, Italy: SCITEPRESS - Science and and Technology Publications, 2016, pp. 407–414. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005740704070414

[13] A. Habibi Lashkari, G. Draper Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor Traffic using Time based Features:," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*. Porto, Portugal: SCITEPRESS - Science and Technology Publications, 2017, pp. 253–262. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006105602530262

[14] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu, "Neural Discrete Representation Learning," May 2018, arXiv:1711.00937 [cs]. [Online]. Available: http://arxiv.org/abs/1711.00937

[15] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," Aug. 2016, arXiv:1601.06759 [cs]. [Online]. Available: http://arxiv.org/abs/1601.06759

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Dec. 2017, arXiv:1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762

[17] P. Esser, R. Rombach, and B. Ommer, "Taming Transformers for High-Resolution Image Synthesis," Jun. 2021, arXiv:2012.09841 [cs]. [Online]. Available: http://arxiv.org/abs/2012.09841

[18] J. Yu, X. Li, J. Y. Koh, H. Zhang, R. Pang, J. Qin, A. Ku, Y. Xu, J. Baldridge, and Y. Wu, "Vector-quantized Image Modeling with Improved VQGAN," Jun. 2022, arXiv:2110.04627 [cs]. [Online]. Available: http://arxiv.org/abs/2110.04627

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017, arXiv:1412.6980 [cs]. [Online]. Available: http://arxiv.org/abs/1412.6980