FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Data-driven Traffic Generation Model for Network Digital Twins

Catarina Mouro de Sousa

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Helder Martins Fontes

Co-Supervisor: Hélder Filipe Pinto de Oliveira

October 17, 2023

Resumo

O rápido aumento da complexidade de redes suscita uma série de preocupações, não apenas em relação a segurança, mas também na sua gestão e Qualidade de Serviço (QoS). Dados de tráfego da rede, quer se tratem de dados estatísticos ou de pacotes mais específicos, facilitam a compreensão sobre o comportamento da rede e de todos os seus utilizadores, permitindo a melhoria da QoS de uma rede verdadeira.

O problema surge quando se considera a precisão oferecida pelos ambientes de teste, tais como a simulação de cenários reais de rede durante o desenvolvimento de novas soluções. Geradores de tráfego sintéticos estão actualmente disponíveis, mas muitas vezes não fornecem soluções de tráfego realistas ou sofrem de viés na geração de dados. Por conseguinte, existe a possibilidade de aproveitar algoritmos de aprendizagem para este fim, mais especificamente Redes Adversárias Generativas (GANs), uma vez que estas são capazes de se adaptar a uma multiplicidade de cenários de treino.

O trabalho no estado da arte centra-se nestes algoritmos adaptados a cenários simples em que o tráfego da Internet é gerado, seja ao nível da geração de pacotes ou através da replicação de metadados após simulação. Com este trabalho pretendemos expandir sobre os trabalhos existentes para criar um modelo capaz de gerar tráfego sintético realista para uma rede específica desejada, aprendendo assim especificações da rede escolhida.

Para atingir este objectivo, pretendemos desenvolver um algoritmo que, através da utilização de estruturas GAN adaptadas à previsão de séries temporais, consegue atingir previsões precisas quando se trata de geração de pacotes sintéticos de *Internet* através da modelação de dependências em atributos de tempo de fluxo destes pacotes. Ao desenvolver este algoritmo, esperamos ajudar a preparar o caminho para um maior desenvolvimento na geração sintética do tráfego do mundo real. Os resultados adquiridos a partir do algoritmo podem depois, por exemplo, ser introduzidos num simulador de rede tal como o *Network Simulator 3* (ns-3) permitindo a criação de *digital twins* de redes do mundo real e a validação de soluções de rede como se elas fossem executadas numa rede real; ou até ser utilizadas como dados adicionais no treino de outros algoritmos de classificação.

Abstract

The rapid rise of network complexity raises a number of concerns regarding, not only security, but also network management and Quality of Service (QoS). Network traffic data, be it statistical or more specific packet data, facilitates the understanding upon the network and user's behaviour and allows for the improvement of a real world network's QoS.

The problem arises when considering the accuracy offered by the testing environments, such as the simulation of real network scenarios upon the development of new network solutions. Synthetic traffic generators are currently available, but often fail to provide realistic traffic outputs or suffer from data generation bias. Therefore, there is an opportunity to leverage learning algorithms for this purpose, more specifically Generating Adversarial Networks (GANs), as they are capable of adapting to a multitude of training scenarios.

Previous work focuses on these algorithms adapted to simple scenarios where generic Internet traffic is generated, be it at packet generation level or through metadata replication upon simulation. With this work we intend to expand on existing works to create a model capable of generating realistic synthetic traffic for a specific desired network, learning this chosen network's specifications.

To achieve this, we aim at developing an algorithm that, through the use of GAN frameworks adapted to time series forecasting, can attain accurate predictions when it comes to the generation of synthetic Internet packets by modelling dependencies in packet flow time attributes. By developing this algorithm, we hope to help pave the way for further development in the synthetic generation of real-world traffic. The outputs acquired from the algorithm can, for instance, then be fed into a network simulator such as Network Simulator 3 (ns-3) allowing the creation of digital twins of real-world networks and the validation of networking solutions as if they were run in the real-world network; or even be used as additional data in the training of other classification algorithms.

Acknowledgements

This work is a result of the project "DECARBONIZE – DEvelopment of strategies and policies based on energy and non-energy applications towards CARBON neutrality via digitalization for citIZEns and society" (NORTE-01-0145-FEDER-000065), supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

I'd want to express my sincere gratitude to my supervisors Professors Helder Martins Fontes and Hélder Filipe Pinto de Oliveira, who always helped and guided me through this dissertation. Without a doubt, I would not have completed this work without their help and contributions, I thank them both for their patience and support. I would also like to thank João Pedro Loureiro and Professor Rui Lopes Campos who also guided me and helped me in the first steps of this dissertation.

To my family and all my friends, thank you for standing by my side and providing me with the unwavering support I needed to succeed. Your friendship has brought both laughter and solace during moments of despair, from all our study sessions to the countless sleepless nights. You have not only made this journey enjoyable but have also helped me navigate through all the difficulties I have faced. I will forever cherish the memories we have created together, I love you all.

Catarina Mouro de Sousa

"Before we work on artificial intelligence
why don't we do something about natural stupidity?"

Steve Polyakl

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ARP	Address Resolution Protocol
BCE	Binary Cross-Entropy
CSV	Comma-Separated Values
DNS	Domain Name System
DT	Digital Twin
DTN	Digital Twin Network
FTP	File Transfer Protocol
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MLPL	Machine Learning based Propagation Loss
MSE	Mean Squared Error
MTCN	Multivariate Temporal Convolutional Network
ns-3	Network Simulator 3
QoS	Quality of Service
ReLU	Rectified Lineal Unit
RNN	Recurrent Neural Network
RMSE	Root Mean Squared Error
RRE	Request-Response Exchange
RTT	Round Trip Time
SL	Supervised Learning
TCN	Temporal Convolutional Network
TCP	Transmission Control Protocol
TM	Traffic Matrix
TS	Trace-based Simulation
UDP	User Datagram Protocol
WGAN	Wasserstein Generative Adversarial Network

Chapter 1

Introduction

Over the last decade, due to significant advances in technology, the impact of computer networks on everyday operations has grown significantly. Consequently, so did the demands put on these modern networks to achieve better values for throughput and capacity. As a result, demand for computer network research increased, as the evolution of existing network infrastructures and protocols is essential to overcome new adherent challenges of modern networks.

In order to develop these solutions for next-generation networks, researchers need to evaluate the algorithm's performance in realistic scenarios. The often preferred approach being experimental testbeds, where all environment's characteristics are considered. However, this approach presents some limitations, such as the availability of the testbeds (may be busy, private, or even non-existing) and the difficulty to repeat and reproduce experiments, which may lead to results that differ from those obtained originally [1]. These limitations hinder the researchers' capacity to validate the results of others, a key component of the scientific method. As a consequence, the replication of past experimental conditions is, often, only possible through the use of simulators adapted to reproduce those specific conditions.

Researchers typically rely on trace-based simulation, which is a method of evaluating the performance of a computer system or network by employing a recorded sequence of events, or "trace", to simulate the system's behaviour. The trace is a record of the system's operations across time, and it might include details such as the instructions executed, the data accessed, and the interactions between different system components. Trace-based simulation allows researchers and engineers to analyze the behaviour of a system without having to build or access the actual system, however, they do not offer that much flexibility, as they are limited by the fact that the simulation setup must match exactly the original experiment setup, including network topology.

In this context, the concept of a digital twin can be introduced with the aim of improving the realism of simulation environments and, furthermore, allowing the evaluation and validation of the performance of new network solutions in realistic conditions. A digital twin defines a digital model of a physical system and its ongoing processes in which the performance testing of various solutions can be evaluated without affecting the physical system.

A great option for digital twin deployment, in the context of network development and simulation, is the Network Simulator 3 (ns-3). ns-3 is a highly realistic discrete-event network simulator that allows for repeatability and reproducibility upon networking experimentation. It does so by providing a framework capable of truthfully replicating a given network configuration and corresponding communications links between nodes, through which different scenarios can be run without interfering with real network operations. Current trace-based simulation approaches for ns-3 have been proposed, more specifically [1], in which data collected from real experiments is integrated into the network simulation in order achieve better approximations to real-world conditions. In the same study, network traces from previous experiments allowed for the repetition and reproduction of the same exact conditions on the ns-3 model, demonstrating that accurate predictions could be made from data observed in a real environment.

1.1 Problem and Motivation

Despite the presented advantages of network simulation, these simulations may not be able to accurately represent real network scenarios, especially when replicating very volatile and unpredictable environments.

As stated, the development of novel solutions depends greatly on realistic simulations in order to correctly assess their performance. These simulations, although fully controllable and easily reproducible, take on a simplified view on real system conditions and, by doing so, often produce optimistic results. While simulations allow full control over the scenario conditions, real world experiments are influenced by external random phenomena such as noise, which comes as a limitation due to its impact on the faithfulness of these simulations.

One of the resulting simplifications comes in the form of the generated traffic. Although studies may currently be carried out in which path loss is accurately simulated from real-world experiments, simulation traffic is still produced in a highly simplified manner in which the network cannot be fully represented.

Furthermore, another limitation arises upon data extraction for trace-based simulations. Not only do user privacy and network security issues restrict the use of real network traffic for these simulations, since it may contain sensitive and personal information, but simulations must also not perfectly replicate real experiments, but rather must mirror in a way that would be realistic for the real scenario to work. Thus, developers must find alternative methods for obtaining network traffic, typically by resorting to traffic generation algorithms.

To this day, many algorithms in the field of Machine Learning have been proposed and explored regarding network traffic generators. The most powerful and, therefore, the most developed frameworks in this area being Generative Adversarial Networks (GANs), that have been able to outperform other contender algorithms. Despite this, development is still in its early works and still suffer from a few shortcomings.

As such, state-of-the-art works even now deal with very simplistic and generic traffic generation scenarios, which proves as an obstacle to the creation of realistic testing environments

capable of dealing with the complexity of real network traffic loads. Given this, it is clear that further improvements can still be done regarding network traffic generation algorithms and that, by doing so, we would be prompting the development of extended testing environments for potential next-generation networking solutions.

1.2 Objectives

The main objective of this work was then to develop a machine learning model capable of accurately generate network traffic data. With this in mind, this work also explores the effects of different preprocessing techniques, input transformations, and data augmentation methods on model performance, using a variety of evaluation metrics.

The objectives set for this work were:

- the development of preprocessing techniques to effectively address imbalanced datasets, the identification of optimal data augmentation techniques to enhance model performance, and the study of how different input transformations affect the behavior of the model;
- the development of a novel machine learning model capable of accurately generating network traffic data, the identification of the optimal model architecture and hyperparameters to achieve high accuracy, and the exploration of the effects of different loss functions on model performance;
- the demonstration of the model's ability to generalize to some extent, despite being trained on a limited portion of the data and further identify the model's limitations.

1.3 Contributions

Considering the previously mentioned work objectives, the following contributions have been accomplished:

- **Novel machine learning model for traffic generation based on GANs and LSTMs** - This model consisted on a new architecture, inspired by the state of the art on GANs and evolved for the specific domain of network traffic generation. In the end, it was capable of reproducing both TCP and UDP traffic flows with accuracy. This model can be used in the context of Digital Twins, but also to generate realistic traffic flows in real networks. Other applications may include its use as a source for data augmentation for classifier training in the fiels of cybersecurity, traffic forensics, QoS optimization, among others.
- **Methodology for Optimal Model Architecture and Hyperparameters** - We provide a comprehensive methodology designed to identify the optimal architecture and hyperparameters for high-accuracy traffic generation models. This approach, developed specifically for this class of GAN models, underlines the significance of hyperparameter fine-tuning during the training process.

- **Anonymized Traffic Capture Methodology** - Our approach to capturing and anonymizing network traffic retains critical flow-level information while preserving anonymity. Unlike existing solutions, this method maintains the concept of flows, enabling detailed analysis of packet inter-arrival times and packet size distribution specific to each flow. This methodology enhances the accuracy and granularity of traffic characterization, offering a unique advantage in network analysis.

1.4 Document Structure

This document is organized in the following structure:

- Chapter 2 - Background and Literature Review - This chapter presents the literature review alongside the needed background information to provide context for the main dissertation topics.
- Chapter 3 - Data Analysis and Processing - This chapter describes the approaches taken in order to prepare the data for the model.
- Chapter 4 - Traffic Generator Model - This chapter presents and describes the details of the proposed model.
- Chapter 5 - Evaluation and Result Analysis - This chapter presents and discusses the results of this work.
- Chapter 6 - Conclusions and Future Work - This chapter concludes the dissertation by exposing the main conclusions and key ideas to be retained.

Chapter 2

Background and Literature Review

This chapter covers the relevance and need for synthetic network traffic data, as well as possible real-world applications where this data would prove useful. Furthermore, an analysis is made on the literature review by reading the work's proposed technologies and solutions to the problem exposed in Chapter 1. It is divided into the following sections, which bring to light the main topics for this dissertation:

- **Synthetic Network Traffic Data Relevance and Applications** — Description on the need of synthetic traffic data and its most common employment scenarios
  - **Digital Twin Technology** — An initial introduction to the concept of digital twin networks and current implementations available, followed by a description of the ns-3 framework and its importance on digital twin deployment;
  - **Classifier Training** — Exposition on some possible classifiers in which network traffic data could be used for data augmentation
- **Network Traffic Generation** — Brief description upon the existing solutions available for network traffic generation;
- **Algorithm Validation Techniques** — Explanation on the different techniques used in literature for model validation;

2.1 Synthetic Network Traffic Data Relevance and Applications

2.1.1 Digital Twin Technology

Digital Twin (DT) is an emerging technology that can be best defined as a digital model capable of representing a physical system and its ongoing processes. Although some writers define a DT as something that can mirror the state of a physical asset, while enabling monitoring, control, and modification of that asset; other authors chose to define it as a simulation model that mimics physical systems and enables their simulations.

These models prove themselves advantageous independently of the application area, having gained particular attention of researchers in the fields of industry [2] and next-generation computer networks [3].

2.1.1.1 Context in Network Simulations

In computer networks, DT technology can be used to generate virtual models of the network infrastructure and can be utilised to analyse, monitor, and predict the network's behaviour, thereby assisting network administrators and engineers in optimising network performance and reliability. This is of special interest for this work as the implementation of a digital twin networks (DTN) [4] can help demonstrate the viability of current network solutions in terms of effectiveness, latency, security, and other factors.

Due to the popularity of network modelling, there is a substantial number of proposals in the literature. In a previous work on DTN deployment [4], the DTN's execution has been proven to be efficient and light when compared to traditional network simulation technologies. This is due to the fact that machine learning (ML) methods were used upon the DTN's creation. With the aid of ML, the concept of DT of a particular network can be expanded in order to better fit its physical counterpart, as it is done in [5] with a Graph Neural Network based model called TwinNet. Here, the complex relationships among the various nodes in the network layout, along with their scheduling algorithms, could be understood by the algorithm and, furthermore, successfully generalised to its input parameters.

Another notable example of ML implementation on DTN development is done in [4] with the development of a Machine Learning based Propagation Loss (MLPL) module which will be further explored in Section 2.1.1.2.

Despite these advancements into ML algorithms, most state-of-the-art DTN deployments require real world input data to achieve accurate modelling of the physical system. Trace-based simulations [6] [1] take this approach by making use of data recorded during live experiments, either by traffic capture or low level variable abstraction. This comes as a challenge still to be addressed as, depending on the network and testing environment desired, accessibility to these resources may be limited or, if available, may not correspond to a correct representation of the real network's typical behaviour.

In [7] a similar approach is taken, where network traffic acquisition is based on real packet sampling. Here, in order to fulfil their reciprocal correlation, physical network elements and DTN elements are assigned a unique identifier across the entire network. A data management unit present in the DT, then, constantly gathers and manages the traffic data of the physical network before distributing it to each element, maintaining consistency between both network twins. This DTN implementation allows for similar values of forwarding delay and jitter, though only permits a fixed time processing delay in simulated traffic. Consequently, it is unable to completely represent the real world system.

In [5] and [8] the authors present yet another strategy, introducing Traffic Matrices as inputs to each DTN model. Working upon previous work, RouteNet [8] particularly effectively captures traffic routing across network topologies by simulating the connections between links with source-destination paths. The accuracy is then assessed using a dataset produced, the Traffic Matrix (TM), by a specially created packet-level Omnet++ simulator. The TM is then able to model the traffic exchanged by every source-destination pair, which gives an excellent estimate of delay, jitter, and loss when tested against topologies, routing, and traffic that wasn't seen during training. Though, since these models currently work primarily on the metadata level, there is still room for improvement in terms of expansion to the byte or packet level of input data.

2.1.1.2 Trace-based Simulations

Trace-based simulation (TS) is a technique that can be used for network simulation with the use of recorded network data, also known as a trace, as simulation input. Current trace-based modelling methodologies allow for the replication and recreation of the identical circumstances observed in previous experiments, which in turn allow to provide accurate representations of actual network traffic. This is useful for multiple purposes, including testing network performance, evaluating network protocols, and training machine learning models as it has been done in recent years by INESC TEC [6] [1].

In their most recent work [1], the MLPL module was introduced, which allows for the prediction of propagation loss using traces acquired from previous trials. The MLPL module is then capable of simulating the experimental circumstances measured in a real environment, with the flexibility to allow any network architecture, mobility pattern, provided traffic, and simulation length.

To do this, this module is based on a supervised learning model that was trained using network traces obtained in the physical twin. It is then subdivided into two models: the path loss model and the fast-fading model. The first is deterministic and calculates the route loss for a given distance between two nodes, while the second is stochastic and creates random samples based on the Probability Distribution Function that characterises the phenomenon. Here, the collected findings demonstrate that the MLPL module can properly predict propagation loss seen in a real environment and duplicate the experimental circumstances of a specific testbed, allowing the production of digital twins of wireless network environments.

The research proposed in this dissertation expands within the context of this project conducted by INESC TEC that researches methodologies for digital twin deployment in network scenarios. While their current trace-based modelling approaches enable the replication and reconstruction of similar conditions seen in prior experiments, this is only true for network topology and link properties. Until now, all traffic transmitted between nodes has consisted of basic, UDP only, packets [6] [1] created by the iperf2 algorithm (later explored in Subsection 2.2.1) that do not accurately represent the real traffic exchanged in normal network functioning. As a result, we intend to expand on this work and improve the model by integrating a mechanism for traffic generation that can accurately replicate real-world packets.

2.1.1.3 The ns-3 Software

The Network Simulator 3, or ns-3, software is a discrete-event network simulator and one of the most commonly used for wireless networks. ns-3 provides excellent control over the simulation environments, where each event in the simulator has a corresponding execution time, with the simulation moving forward by carrying out events in the chronological order of the simulation time. Events are consumed as a simulation runs with additional events that may (or may not) be generated as a consequence.

This tool was created with the needs of the research community in mind, not only allowing for validation or peer review tasks among various groups, but also promoting a community-based collaboration model to support the development of new modules, as demonstrated in [1].

This software can therefore constitute as a solution for digital twin deployments as it allows simulations of a desired network in different scenarios without interfering on the real network's operations. In this context, the TS approach can rely on the ns-3 framework, combining the best features of simulation and experimentation to enable network trace repeatability. This is done in [6] with the implementation of the aforementioned Propagation Loss Module, where Figure 2.1 depicts a high level comparison between pure ns-3 simulations and the TS approach simulation.

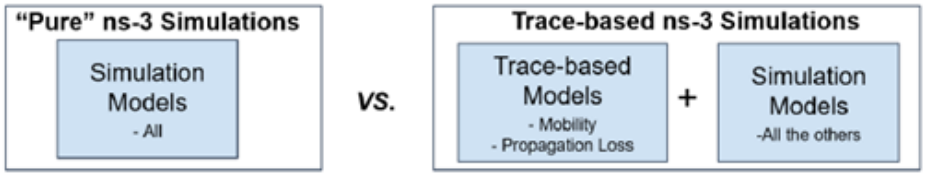

Figure 2.1: High-level comparison between pure simulation and TS approaches [1]

2.1.2 Classifiers

Data augmentation has become an important component in the field of machine learning as it helps a variety of models in numerous ways. However, its importance becomes especially clear in classifiers, where the generation of network traffic data is an essential element of the research process in a wide variety of networking-related fields.

One of the main benefits of data augmentation is that it can improve the performance and generalisation of models. By expanding the training dataset, models are exposed to a wider range of examples which, in turn, enables them to better understand the data's robust features and adapt to the complexities and variations present in real-world scenarios. As a consequence, augmented models are then able to perform better when categorising unknown data and to produce more dependable predictions.

The problem arises in collecting the augmented data since, in many machine learning applications, obtaining a big labelled dataset may be a difficult task, and it is not always viable to

transform the existing data. This is the case of network packets as a transformation to the original packet could render the packet invalid in a real scenario. Synthetic data generation provides a solution to this problem by creating additional training samples which correctly represent the original dataset, without directly duplicating its samples. This augmented dataset allows models to learn from a more diversified and representative set of examples, which reduces the possibility of overfitting and improves the model's capacity to generalise successfully to unobserved data.

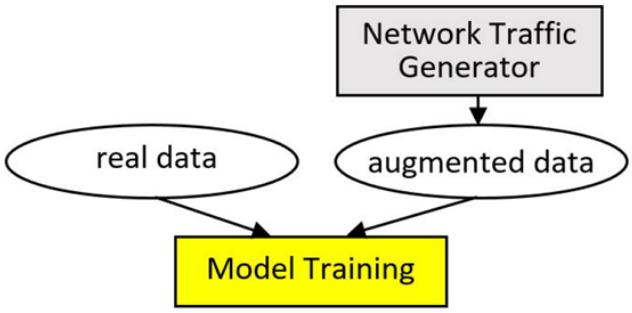

Figure 2.2: Synthetic Traffic Data used for Data Augmentation [9]

Data augmentation is very helpful for classifiers in particular as these models are designed to categorize inputs into specific classes, making the diversity and balance of the dataset crucial for accurate predictions. Furthermore, supplementing the data with class-specific variants improves the model's capacity to capture key traits and efficiently differentiate between distinct classes.

Moreover, the use of generated data is critical for the replication of real-world traffic situations, allowing researchers, developers, and network administrators to assess the performance of various systems or algorithms, particularly in the area of networking. This is because understanding network behaviour, traffic patterns, and anomalies is essential for developing effective networking algorithms, protocols, security mechanisms, intrusion detection systems, and network management methods, offering valuable insights into network performance and vulnerability. It then becomes possible to accurately identify and analyze various forms of network traffic, including web browsing, video streaming, and file transfer and for researchers to generate variations that simulate diverse network conditions, attacks, and events.

With this in mind, building on the output of the generator, multiple classifiers can be trained using the augmented dataset, each serving specific objectives and further enhancing traffic analysis and control. These classifiers can encompass:

- **Protocol Classifier:** Classifier that is trained to categorize network traffic based on the protocols used, such as TCP (Transmission Control Protocol), UDP (User Datagram Protocol), ICMP (Internet Control Message Protocol), HTTP (Hypertext Transfer Protocol), and FTP (File Transfer Protocol). It allows for the identification and differentiation of different protocol-based communication patterns.

- **Intrusion Detection/Prevention Classifier:** Classifier that is trained to identify various types of network attacks or malicious activities, including DoS attacks, DDoS attacks, port scanning, SQL injection, XSS attacks, and others.

  This is the case of the paper [9], where a generator with the purpose of IoT traffic generation was designed and the output used to evaluate the attack detection rate of a network intrusion detection classifier.

- **Application Classifier:** Classifier that is trained to classify network traffic based on the application or service being used, such as web browsing, email, video streaming, file sharing, VoIP, online gaming, and more.

- **Quality of Service (QoS) Classifier:** Classifier that is trained to differentiate network traffic based on QoS requirements, distinguishing between real-time traffic (voice or video) and best-effort traffic (web browsing or file downloads), for example.

- **Traffic Anomaly Detection Classifier:** Classifier that is trained to identify abnormal network traffic patterns that deviate from regular behavior, which can indicate potential security threats or network performance issues.

- **Traffic Engineering Classifier:** Classifier that is trained to classify traffic based on its source, destination, or other parameters to assist in traffic engineering tasks. For instance, traffic can be classified based on the geographical location of the endpoints, allowing network operators to optimize routing and resource allocation strategies for different regions or to identify traffic patterns that require specific network infrastructure improvements.

2.2 Network Traffic Generation

Network traffic generators are algorithms capable of reproducing real network traffic and, therefore, able to emulate communication between devices in a network. Consequently, these generators could serve as a means to get around the logistical and privacy problems associated with trace-based or other solutions that use real-world traffic.

Network traffic generators are essential because they allow network administrators and researchers to emulate real-world traffic patterns to test the performance and capacity of a network. This can help identify possible bottlenecks and other difficulties that may occur in a live network environment, allowing administrators to make the required adjustments to enhance performance. By producing various types of traffic, network managers and engineers may assess how a network will perform under varying conditions and make any necessary improvements to guarantee that it operates efficiently and reliably.

Many traffic generation methods have been implemented and compared in existing literature [10], the most significant of which are explored in the following sections. Though the concept of traffic generation is not novel, to the best of our knowledge, previous works on the topic have

not aimed for digital twin implementation of a specific network but instead for more generalized reproductions of network traffic.

2.2.1 Traditional Algorithms

A traditional network traffic generation algorithm is usually a mathematical model used to generate synthetic network traffic data for simulation purposes, and typically designed to produce traffic patterns that are similar to those observed in real-world networks. The first models were mostly based on simple probability distributions, for example, the Poisson process, the Pareto distribution, and the Markov chain. Though, since then, these algorithms have developed dramatically [11].

More recent and complex network traffic generators have emerged since then, with the purpose to evaluate network performance, test network protocols, and develop network optimization strategies. The most popular of these were collected in [10] based on their usage during a 13-year period (from 2006 to 2018). Their objective was to analyze the existing characteristics and features of regularly used tools, offer an organized digest of these qualities, and then present a systematic process for selecting appropriate generators for individual research goals. Figure 2.3 depicts the findings of this investigation, listing the top ten software-based generators in descending order of usage.

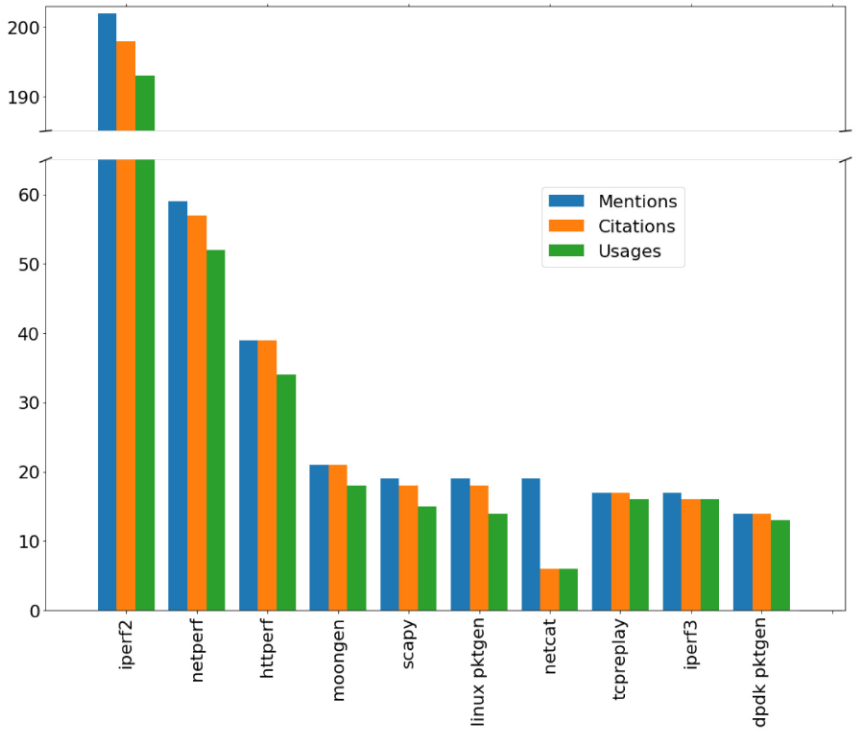

Figure 2.3: Top traffic generators between 2006 and 2018 [10]

According to the findings, constant or max throughput traffic generators, particularly iperf2, continue to dominate in terms of usage. While more recently created traffic generators based

on stochastic models are not as frequently cited, even when controlling the smaller set of recent articles. An example of these are the swing [12] and the d-itg [13] algorithms.

This paper also further classifies different generators by taxonomy [10]:

- **Constant or Maximum Throughput Generators** — A packet is produced with specific application-layer fields and then repeatedly transmitted on a network interface at a constant or maximum rate. Popular examples in this category are iperf2 and netperf, which, despite being the easiest to use, allow little to no variety in the packet header and payload contents. As a result, they are only applicable to a small subset of applications and may not accurately represent network traffic in normal topologies.

- **Application Level Synthetic Workload Generators** — These generators, such as the httperf, produce network packet traffic for a particular kind of application or higher-layer protocol. This method is frequently capable of producing realistic modifications on packets for the particular application or protocol in question. However, using this strategy exclusively may result in network behaviour that is devoid of realistically occurring concurrent background traffic and application-user interaction.

- **Trace File Replay Systems** — In the case of replay systems, packets from a previously existing trace file are injected into a network channel at the time intervals that are specified in the capture file. These systems are capable of producing traffic workloads that are identical to the actual traffic, and this is especially true if the workload can be run on an experiment topology that is analogous to the topology of the actual network. This would mean that it would function seamlessly in combination with a DT. However, the vast majority of replay systems are stateless, and continuous replays of the same trace file over a network will keep generating the same events at regular intervals, leading to patterns that are impractical for many use cases.

- **Model-Based Traffic Generators** — Another popular method of realistic traffic generation is the creation and transmission of packets in accordance with random distributions of various parameters, such as time intervals and packets sizes. Users of these generators have the ability to specify a random distribution model with parameters that are specific to the desired outcome.

- **Trace Driven Model-Based Generators** — Some traffic generators go a step further by allowing experiments to provide a trace file or log file as input. This file will then be analysed to generate a model that fits the different parameters. The swing algorithm [12] is a great illustration of this approach. The characteristics of users, connections, individual packets, and the network as a whole are analysed by Swing as well as Request-Response Exchanges (RREs). It is confirmed against genuine traffic by extracting the Hurst Parameter from both the trace that was created and the traffic that was actually being sent [11]. Surprisingly, its performance is quite good even at time-frames that are shorter than the Round Trip Time (RTT).

- **Script Driven Traffic Generators** — Moongen and DPGK pktgen are just two examples of the many script-driven traffic generators that have been developed in recent years. These generators make it possible for users to construct any kind of packet, with practically any header value, and also to dynamically alter the packets at runtime. At the time of publishing, only a small number of script-driven traffic generators covered the list that was constructed; nonetheless, the authors' speculation led them to anticipate that these algorithms would have an especially valuable use in the near future, which was later verified.

Another study that may be worthy of consideration is one that details some early software-based methods for traffic generation algorithms. Here, some of the most widely used packet-level traffic generators were chosen, and the respective log files produced by each were analyzed.

Here, it was verified that the generators tend to act similarly, saturating upon reaching a particular maximum value and achieving a deviation from the predicted behavior that is only increasing beyond this threshold. The authors claim that these disparities demonstrate that, beyond hardware limitations, software design has a substantial influence on the capability to achieve the necessary speeds. Here, the limitations of these algorithms, such as the d-itg which was explored in the previous paper [10], are outlined. The conclusion is that these algorithms are far too simplistic and are incapable of keeping up with the increasing complexity and rate values imposed by modern networks and, as a result, there is a need to adapt the algorithms used.

2.2.2 Supervised Learning Algorithms

With the rising popularity of Machine Learning (ML) algorithms in recent years, traditional traffic generation algorithms are becoming somewhat obsolete for researchers. The incorporation of learning algorithms into this topic enabled more packet information to be processed and used during synthetic traffic generation. Therefore, increasing the algorithm's complexity and, consequently, its output's realism to mimic real network communication.

Supervised Learning (SL) is one paradigm of ML which consists of having an algorithm that learns by using data sets where all the data is labelled. That is to say that, it is known beforehand what the data represents and what the outputs are for sets of inputs. With this paradigm, an algorithm is trained with a labelled dataset, the training set, until the algorithm is able to detect the underlying patterns between input data and output labels. After completing the training phase, the algorithm should then be able to receive unlabelled data as input and correctly label it.

Despite the fact that other studies have been performed on alternative ML paradigms, this chapter will focus entirely on supervised algorithms, as they are the most prevalent and have produced the best results on this topic.

2.2.2.1 Generative Adversarial Network Models

Generative adversarial networks (GANs) are a type of unsupervised learning algorithm, meaning they do not require labelled training data. Instead, GANs use a two-part model, consisting of a generator and a discriminator, to learn the distribution of the data and generate new samples that

are similar to a given training dataset [14]. A representation of the general GAN architecture can be observed in Figure 2.4.

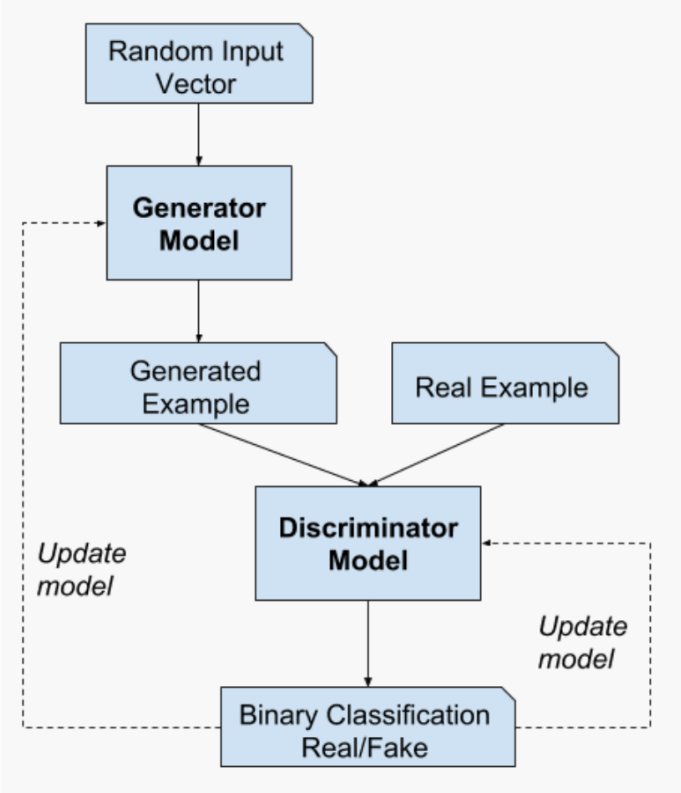

Figure 2.4: GAN architecture [15]

However, a clever property of the GAN architecture is that the training of the generative model is framed as a supervised learning problem. This is due to the fact that the two models, the generator and discriminator, are trained together where the generative model generates new samples, while the discriminative model evaluates the generated samples and try to distinguish them from the real ones. This is a form of classification algorithm where the supervised learning algorithm uses the labels to learn to sort the input based on whether they are real or generated samples. Both networks are iteratively trained in competition until the discriminator can no longer distinguish between genuine and created data, indicating that the generator model can provide realistic results. Consequently, this can be useful for improving the performance of the supervised learning algorithm by providing it with additional training data.

In the context of this work, the algorithm should be able to receive cataloged real-world network traffic and, from this input, generate similar packet types, with slight but realistic variations. Within the topic of traffic generation, we can further observe numerous application areas, such as Internet of Things (IoT) [9] [16] or intrusion detection and cyber security [17] [18] [19].

Some papers go a step further and employ a similar variation of the GAN model, the Wasserstein GAN (WGAN). This particular kind of GAN uses a loss function called the Wasserstein distance, which provides a more stable and effective training signal for the generator network,

allowing it to generate higher-quality samples. An example of these models was developed by [18], in which a traffic generator was able to learn the characteristics of recorded network traffic for the testing of intrusion detection systems. Their contributions are centred on three processing approaches for transforming continuous data into the category properties typically associated with traffic flows. Existing GAN systems, however, can already cope with discrete categorical data, such as the network packet encoding scheme, utilised in [19]. Additionally, its generator does not replicate network traffic directly, rather, it aggregates metadata variables to simulate traffic flows.

Following this work, [19] prototyped the PAC-GAN traffic generator, which, unlike its predecessor, is capable of generating packets of three distinct traffic kinds, namely ICMP pings, DNS searches, and HTTP requests. This study did not take into account individual attributes, but instead whole traffic packets which were then mapped and fed to the model in an image-based matrix representation. Here, authors claimed to be the first to produce traffic at the individual packet byte level that allowed for much more control and flexibility over the types of traffic that can be synthetically produced.

The overall GAN model implemented in this paper is represented in Figure 2.5, where it is possible to observe that, after performing data encoding into input square matrixes, this PAC-GAN's training process involves two backpropagation feedback loops based on cross-entropy loss values to increase both block's success rates. Authors were then able to achieve up to 99% success rates for individual traffic generations, while 88% success rates were achieved at best for the generation of different traffic mix packets using this model.

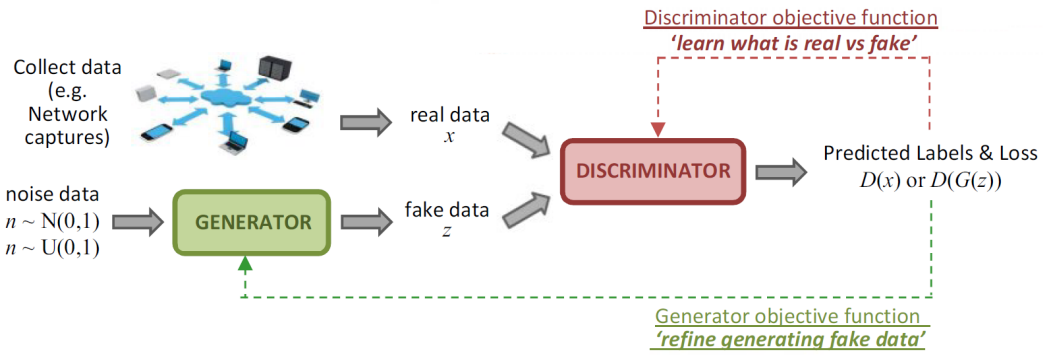

Figure 2.5: GAN architecture in a traffic generation model, modified from [19]

Since then, more modern models have been presented, with an emphasis on IoT. Specifically, [9], where a model for combining an autoencoder with a GAN to create packet sequences corresponding to bidirectional flows is presented. Here, WGANs are used to produce latent vectors that decode into realistic feature vectors, achieving sequences of 99.1% valid generated traffic. Similarly, [16], which follows the same IoT-based objectives, now introduces both semantic and network structural information for various IoT devices via a knowledge graph.

This particular research study [16] is of relevance because its authors consider the generation of Internet of Things traffic to be a special case of time series generation, which is influenced by

the presence of complex background information on the devices. During the construction of the GAN framework, the generator G is comprised of three sub-generators: a category generator Gc, a series length generator Gm, and a traffic series generator Gt. Even though the method used on the discriminator D is rather simple, this is offset by the complexity of the generator G. Because the knowledge graph embeddings of every IoT device are distinct from one another, the background information introduced by these can provide the GAN model with a wide range of situations.

2.2.2.2 Pre-processing Approaches

Although these GAN algorithms represent a faithful framework capable of achieving great results on data generation, a limitation imposed by the model comes as only continuous input attributes are able to be processed. Literature offers some possible input data pre-processing solutions to overcome this obstacle in the context of this work's type of data.

When focused on flow-based network traffic, inevitably data types will contain categorical attributes such as IP addresses or port numbers that must be transformed into continuous attributes. Authors in [18], explore the deployment of a WGAN model with three different pre-processing approaches: 1) Numeric transformation, where all attributes are interpreted as numbers; 2) Binary transformation, where each attribute is converted to a varying number of multiple binary attributes; and 3) Embedding transformation, where attributes become embeddings in a m-dimensional continuous feature space. This third method uses IP2Vec [20], based on a fully connected neural network with a single hidden layer in order to learn meaningful vector representations of categorical packet attributes, as illustrated in Figure 2.6. To IP2Vec, similar attributes will consequently indicate that the devices associated with these IP addresses will establish similar network connections.

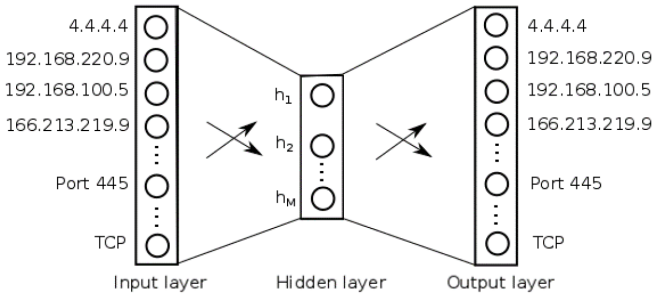

Figure 2.6: Architecture of the neural network used by IP2Vec [18]

Here, authors verified increasingly better results with each transformation, finally achieving values ranging from 92.57% to 99.98% accuracy with the implementation of IP2Vec. Arriving at the conclusion that pre-processing is a crucial factor to take into account for data generation and that these neural networks can be implemented alongside GAN models to improve their performance.

2.2.3 Time Series Generation Algorithms

Time series algorithms are a set of techniques that are used to analyse and model time-dependent data. These algorithms are commonly used to forecast future values of a time series based on its past behaviour, and as such, they can be useful in the generation of synthetic time series data with statistical properties similar to real-world time series data. With this goal of generating synthetic traffic data that accurately reflects the patterns and trends present in the original data, there have been some works that have studied time dependencies in network traffic generation.

The aforementioned research [18] is the one that takes the initial step toward the study of the temporal relations that are present in network traffic. In this study, the timestamps of the network packets are used to construct two attributes labelled weekday and daytime. These properties, along with all the other relevant packet attributes, then serve as input for the GAN algorithm. Here, there are no additional procedures used to obtain the temporal correlations in traffic series as was the case in study [16]. In this second study, a long short-term memory generator equipped with a self-attention mechanism is adopted to capture the temporal correlation in traffic series.

In the field of recurrent neural networks, often known as RNNs, the long short-term memory (LSTM) generators are a type of recurrent neural network that are typically utilised for the modelling and prediction of time series. During the process of making a prediction, self-attention is a method that enables a neural network to attend to many parts of its input simultaneously. It can be used to improve the performance of a variety of neural networks, including RNNs like LSTMs, by allowing the network to weight the value of various input features when generating a prediction, which in turn enables the network to better understand the data it is being fed. This is done thought the use of LSTM cells which are particularly advantageous in this type of tasks that require the model to remember information over an extended period of time.

Each cell has three primary components: an input gate, an output gate, and a forget gate. Each of these gates is a sigmoid neural network layer that outputs a value between 0 and 1 based on the current input data and the previous hidden state of the LSTM cell. The gates are designed to regulate the information flow into and out of the cell, as it can be observed in Figure 2.7.

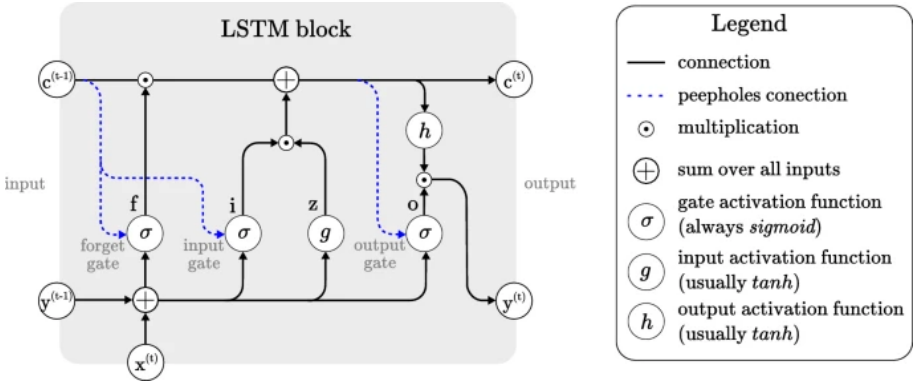

Figure 2.7: LSTM cell architecture [21]

- The input gate controls LSTM cell input. When the input gate is open (sigmoid output is close to 1), fresh information can flow into the cell and be added to its internal state. When the input gate is closed (sigmoid output near 0), new information cannot enter the cell.
- The forget gate controls LSTM cell output. When the forget gate is open, information is deleted from the cell's internal state. When the forget gate is closed, information cannot be forgotten.
- Output gate controls information flow from LSTM cell to output. When the output gate is open, information flows from the cell's internal state to the output. When the output gate is closed, information from the cell's internal state cannot flow to the output.

This way, the input, forget, and output gates allow the LSTM cell to learn and retain patterns in input data over time.

In the context of study [16], by using a series of gates that control the flow of information through the network, the authors claim they are able to effectively capture both the long-term and short-term dependencies in data and, furthermore, able to outperform previous models.

Another example is the GAN based work described in [22]. This study shows that the doppelganger design, which uses GANs to generate synthetic time series data similar to the real data, preserves the privacy of the original data while allowing for data sharing. The doppelganger design was evaluated using networked time series data, and the findings revealed that the synthetic data generated by the generator GAN captured the patterns and trends in the original data while ensuring privacy by not revealing the exact values. The study's results suggest that the doppelganger design is a potential way to protect the privacy of networked time series data while permitting data sharing.

2.3 Algorithm Testing Techniques

There are a few different validation methods that are usually employed in network traffic generation activities to evaluate the quality and realism of the generated synthetic traffic data. Among the most prevalent validation procedures are:

- **Statistical analysis** — This entails comparing the statistical features of the generated synthetic traffic data to those of the real traffic data used for training. Various metrics, including mean and standard deviation of distinct traffic attributes, the distribution of packet sizes and arrival times, and the correlations between different traffic factors, can be compared. In [23], for example, the mean and standard deviation of some traffic parameters, such as packet size, inter-arrival times and bit-rates from and to the connection origin, are analysed between real and synthetic data in order to validate their algorithm.
- **Protocol-level analysis** — This includes analyzing the synthetic traffic data that was generated at the protocol level to confirm that it conforms to the standards and specifications

of the protocol. This can include validating the fields in the packet header, the payloads of the packets, and any other elements that are specific to the protocol to confirm that they are legitimate and accurate.

- **Traffic Classification** — This procedure includes classifying the synthetic traffic data that was generated into several categories, such as network traffic types (for example, HTTP, FTP, and so on), and comparing the classification results with the real traffic data that was used for training. For instance, in the research carried out in [19], the authors categorise the fabricated traffic data into a variety of traffic subtypes, including ICMP, UDP, TCP and DNS, and then evaluate the classification results in light of the actual traffic data.

- **Application-level analysis** — This approach involves assessing the synthetic traffic data that was generated at the application level to confirm that it can be used for the intended applications or scenarios. For instance, if the purpose of generating the synthetic traffic data is to test an intrusion detection system for a network, the data can be assessed to determine whether or not it comprises realistic attack traffic and whether or not it is able to set off the alarms of the detection system, as is done in [17]. Here, the authors examine the synthetic traffic data to determine whether or not it is suitable for the purpose of testing an intrusion detection system for a network. The authors of [19] take it a step further by dispatching the synthetic packets that were generated to the Internet and then monitor the success rate of valid responses received.

Overall, the validation procedures chosen will be dependent on the application's individual requirements and determined by the goals of the traffic generation work.

Chapter 3

Data Analysis and Processing

The development of this work revolves around the implementation of a model that analyzes network traffic to gain insights and generate new data. In order to achieve this, specific input data is essential, which is obtained by observing the flow of traffic passing through a device connected to the target network, for example, an Access Point, an Internet Gateway, or even a specific network client/host.

In order to have a complete understanding of the network's communication and since the uplink and downlink traffic flows are usually asymmetric, it is necessary to then, not only analyse all the different packet's protocol fields, but also to make the distinction between ingress and egress packet flows. However, to protect the confidentiality and privacy of network users, the collected input data then needs to undergo anonymization which includes erasing and encrypting any personally identifying information or sensitive data that is present within the packets. By anonymizing the data on-the-fly, the privacy of network users is protected while retaining the essential information necessary for further analysis.

The subsequent steps involve extracting relevant features and fields from the packets, reformatting the data into a structured CSV format, and transforming the inputs to meet the model's requirements. This comprehensive process enables the model to effectively analyze and comprehend network traffic while prioritizing privacy.

3.1 Capture

Initially, in order to obtain the desired network data, it was necessary to capture packets being sent through a chosen network, which, during the development of this work, was the internal network of the Centre of Telecommunications and Multimedia at INESC TEC.

It is important to note that, while there are several publicly available datasets of network traffic packets online, a significant portion of these datasets is mainly designed for cybersecurity or intrusion detection system assessment. These datasets are purposefully designed to encompass a wide variety of attack scenarios and, as a result, they may not precisely reflect real-world traffic patterns

that occur during normal network operations. By capturing our own data from real network traffic, we ensure that the dataset encompasses the diverse behaviors and patterns characteristic of a INESC TEC user's day-to-day operations.

This was done with the help of the *TcpDump*[24] command which is a widely used and versatile command-line network packet capture tool typically used for monitoring and analysing network traffic. It allows capturing of packets flowing through a network interface, providing valuable insights into the network communication between different hosts, operating at the packet level, capturing individual packets and allowing the examination of their contents.

This command allows for the captured packets to be viewed directly on the terminal, or to be saved as files that can be opened using other packet analysis tools, such as *Wireshark*[25], for more in-depth inspection. This last alternative was specially significant for the development of this work since it allowed for the simultaneous capture of the flow of two distinct ports to separate files, as is demonstrated in Listing 3.1 for ports *eth0* and *eth2*, therefore easily allowing the separation of ingress and egress packets.

3.2 Anonymization

3.2.1 PktAnon

Once the network packets were captured, we proceeded to anonymize them using the *PktAnon*[26] software. *PktAnon* is a tool designed specifically for the purpose of packet anonymization that provides extensive control over the anonymization process and enables the preservation of sensitive data during packet analysis. By applying various anonymization techniques, *PktAnon* transformed the captured packets in a way that preserved their structure and characteristics, all the while removing personally identifiable information or other sensitive details by defining how each specific field was transformed.

One of the key reasons for selecting *PktAnon* is its ability to perform anonymization on the fly, which refers to the process of modifying network packets in real-time as they traverse the network. This is a critical feature relating to data privacy as it removes the need to directly store personal information altogether. In Listing 3.1 the command for on-the-fly anonymization is presented where it can be seen that the outputs of each *Tcpdump* command are being directly writen to the *PktAnon* program: *' -w - | pktanon'*. Due to these reasons and because of its unique features and compatibility with commonly used tools such as *Tcpdump*, *PktAnon* was selected as the best tool for network traffic anonymization for this work.

Listing 3.1: TcpDump and PktAnon commands

```
1 # egress (traffic leaving) captured in port 'eth0'
2 tcpdump -i eth0 -s 0 -w - | pktanon ~/PktAnon/profiles/settings_altered_egr.xml
3 # ingress (traffic coming in) captured in port 'eth2'
4 tcpdump -i eth2 -s 0 -w - | pktanon ~/PktAnon/profiles/settings_altered_ing.xml
```

Another prominent reason is *PktAnon*'s flexibility in allowing users to individually decide which fields inside the collected packets need to be anonymized. This feature allowed for a more effective protection of sensitive information due to the fact that it is given the option of selecting specific fields for anonymization, including addresses and any other sensitive information that may have been included in the packets.

Additionally, *PktAnon* facilitated the anonymization process by offering a variety of anonymization techniques for each selected field. This correlation between specific field and corresponding technique to be applied was done through the creation of an anonymization profile – a concise configuration file where we precisely defined how individual fields within the packet headers should be processed. The anonymization profile then consists of a list of various protocols from the different layers of the TCP/IP protocol model where, for each field in the packet header, it is specified the desired anonymization technique or primitive. This level of customization ensured that we were able to retain precise control over the data that needed to be anonymized while maintaining the general structure and properties of the original packets and, furthermore, allowed properly aligning with the privacy requirements needed.

Althought *PktAnon* cannot operate on the highest layer of the TCP/IP model, the Application layer, it still supports a number of network protocols from the remaining layers such as: TCP and UDP, from the Transport layer; IP, IPv6, ARP and ICMP from the Network layer. During anonymization, the software analyses which protocols each packet corresponds to, as it may be associated with multiple protocols from different layers in the model, and then applies the transformations accordingly.

Table 3.1 demonstrates the protocol attributes that required to be anonymized.

Table 3.1: Protocol attributes requiring anonymization

Protocol	**Attribute**	**Description**
Ethernet Packet	MacSource	the source mac address
	MacDest	the destination mac address
ArpPacket	ArpSourceip	source ip address
	ArpDestip	destination ip address
IpPacket	IpSourceip	source ip address
	IpDestip	destination ip address
Ipv6Packet	Ipv6Sourceaddr	source address
	Ipv6Destaddr	destination address
UdpPacket	UdpSourceport	source port
	UdpDestport	destination port
TcpPacket	TcpSourceport	source port
	TcpDestport	destination port
PayloadPacket	PayloadPacketData	payload data

Figure 3.1 depicts a wireshark examination of the aforementioned anonymization where it is shown the same capture and same packet trace before and after the modification to the sensitive data fields.

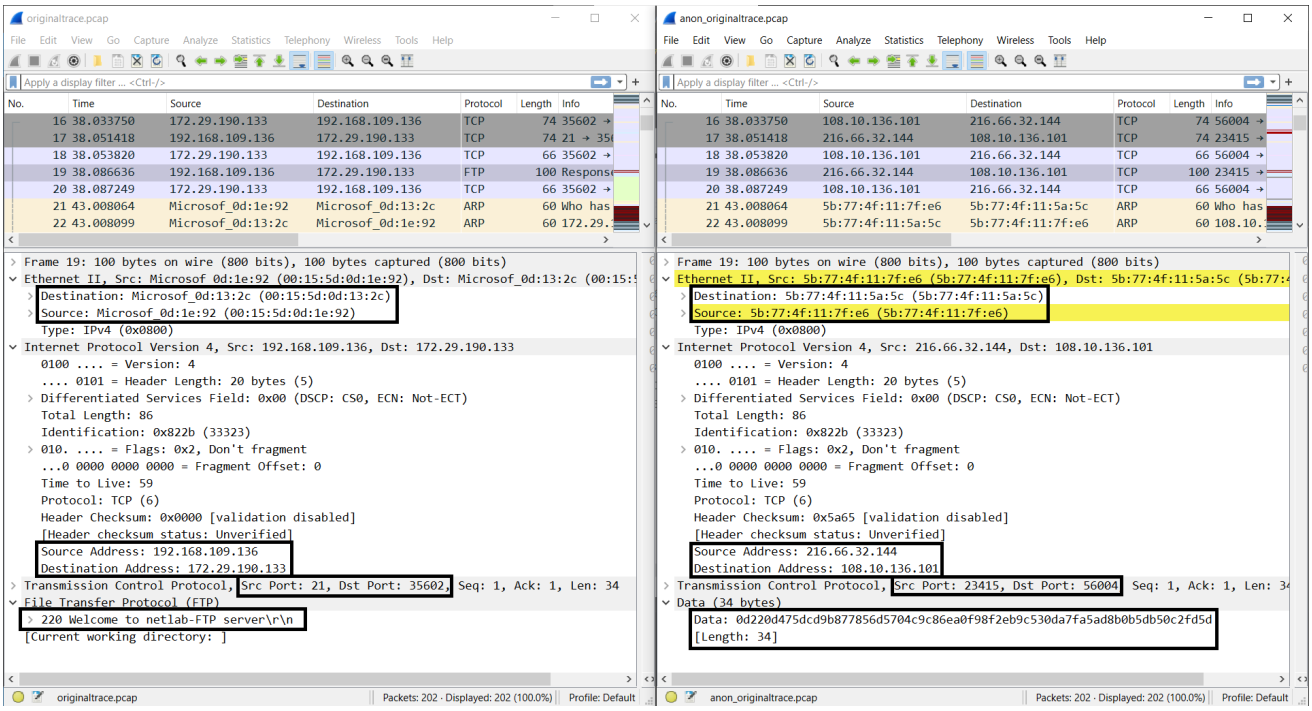

Figure 3.1: Wireshark original trace (left) and anonymized traced (right) comparison

3.2.2 Ports and Addresses

The most sensitive information within network packets often resides in the addresses and ports. To ensure the anonymization of all IP addresses, MAC addresses, and ports in these captured network packets, it was employed the *BytewiseHashSha1* hash algorithm [27] along with a defined key which transformed all ports and addresses into hash values, all the while ensuring consistent mapping of the same addresses to the same anonymized value between captures.

The *BytewiseHashSha1* algorithm, which is based on the SHA-1 cryptographic hash function, processes individual bytes of addresses and ports separately to generate distinct hash values. The key served as a seed value, even allowing the persistent linking between subnetwork masks on multiple addresses. By using a deterministic hash algorithm with a fixed key, this allowed us to maintain the data dependencies and correlations while obfuscating identifiable information that was included inside the captured network packets. Listing 3.2 demonstrates this being done, with a snippet from the configuration file regarding the hashing of UDP ports.

This anonymization technique preserved the structural integrity of captured packets while replacing original identifiers with irreversible hash values, allowing for secure analysis and sharing of anonymized packet data.

Listing 3.2: UDP Port Packet Anonymization

```
6 <submodule name="UdpPacket">
7   <configitem anon="AnonBytewiseHashSha1" key="CLEAN_KEY" name="UdpSourceport"/>
8   <configitem anon="AnonBytewiseHashSha1" key="CLEAN_KEY" name="UdpDestport"/>
9 </submodule>
```

The key plays a critical role in the anonymization process, as it adds an additional layer of protection and randomness to the anonymized data. To generate this random key, whose desired length is set to 10 characters, the code uses a character pool consisting of uppercase letters, lowercase letters, and digits (A-Z, a-z, 0-9), and, at the end of the capture, the code also ensures the secure disposal of the key through a cleanup procedure. The cleanup function is defined so that it performs the necessary actions to remove any trace of the key from the configuration file and terminates relevant processes.

Listing 3.3: Key calculation and cleaning for each run of the program

```
11      # Key Calculation
12
13  character_pool="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
14  key_len=10
15  rand_key=""
16  for i in $(seq 1 $key_len); do
17      rand_index=$(( RANDOM % ${#character_pool} ))
18      rand_char="${character_pool:rand_index:1}"
19      rand_key="${rand_key}${rand_char}"
20  done
21
22      # Cleanup Function
23
24  function cleanup() {
25      python3 output_file_name.py CLEAN_KEY
26      pkill -f tcpdump
27      pkill -f pktanon
28  }
```

The key itself is replaced with the placeholder value "CLEAN_KEY" in the configuration file. This step ensures that the key is not stored in file and eliminates any potential risks associated with its persistence. This key calculation and cleanup process is shown in Listing 3.3.

3.2.3 Payload

In addition to anonymizing IP addresses, MAC addresses, and ports, we took further steps to protect the privacy of the packet payload. This was achieved by substituting the payload data of each individual packet with a random string of the same length and, by doing so, preserving the length field to their original values which was an important factor that needed to be taken into account as it ensured that any subsequent analysis or processing of the data in the length fields remained accurate.

By replacing the original packet payload with a randomized string, it was ensured that the content of the payload was completely obfuscated and, therefore, allowing the safeguarding of any sensitive information or data contained within the packet payloads, such as application-specific data, messages, or file contents.

Listing 3.4: Payload Anonymization

```
30 <submodule name="PayloadPacket">
31     <configitem anon="AnonRandomize" name="PayloadPacketData" />
32 </submodule>
```

PktAnon offers the primitive *AnonRandomize* that was used in this context, as is shown in the configuration file snippet presented in Listing 3.4 regarding the anonymization of a packet with payload data.

3.3 Data Processing

Following the anonymization of the captured network packets, the next step involved extracting specific fields of interest from the anonymized *pcap* files and converting them into a CSV (Comma-Separated Values) format. Both the extraction and conversion process of the desired fields was performed using the *tshark* command which is a command-line tool provided by *Wireshark*. It allowed for the inspection and analysis of all *pcap* files, which contain the recorded network packets.

Once the data was in CSV format, it became more amenable to common data pre-processing techniques. For instance, libraries such as *pandas* in Python provide robust functionality for data manipulation, cleansing, and transformation, which could be readily applied.

3.3.1 Input Fields

In the original captures, both the TCP and UDP packets were present, though it was decided, as a simplification, for these packets to be separated into distinct files.

TCP and UDP are two different transport layer protocols used in computer networks that often coexist within network traffic. While TCP provides reliable, connection-oriented communication; UDP offers a simpler, connectionless communication mechanism. By separating TCP and UDP packets, we can focus on the characteristics and behavior of each protocol individually, allowing for a more targeted analysis of their respective features and, furthermore, build a model that can better understand their different characteristics. This is due to the fact that there are TCP or UDP-only fields, and so, in order for this information to be fed into the model together these fields must be combined.

A challenge arises when these fields are not applicable to a specific packet, to which a zero value has to be assigned. This zero value provides for consistency in the dataset structure and allows for the inclusion of all relevant information; however, it introduces an imbalance in the dataset by creating an unequal distribution between packets with applicable fields and packets without applicable fields. This imbalance posed as a challenge during the training and evaluation of the model since it affected its ability to effectively learn patterns and make accurate predictions, as it was observed from experimentation. Therefore, as initial work, the simplification was made to separate the packets of these protocols.

3.3.1.1 UDP fields

Regarding UDP fields, only three were considered: Time Delta, Frame Length, and Egress Ingress.

The first assessed field, Time Delta, which represents the time difference between packets within the same traffic flow, was chosen due to the fact that, by analyzing the time delta between these packets, we could identify different patterns of regular communication. This value reveals important details regarding the timing and behaviour of network traffic, for example, it may disclose whether a flow is constant or intermittent, if there are bursts of activity or periods of inactivity. This could help in the characterization on the nature and intensity of network traffic.

Listing 3.5: Flow Interval calculation

```
33 for index_aux in range(index-1, 0, -1):
34     row_aux = df.iloc[index_aux, :]
35     if str(row_aux['flow_id']) == str(flow_id) or
36         str(row_aux['flow_id_2']) == str(flow_id):
37           flow_int = row['flow_interval'] - row_aux['flow_interval']
38           break
39     cont_aux =cont_aux +1
```

The Time Delta value was computed as shown in Listing 3.5 through the use of an additional variable *flow_id*. In order to verify that the examined packets belonged to the same flow, it was ensured that the packets had the same endpoints in the communication and the same port in each endpoint. Flow IDs are comprised of these unique combinations of endpoint IP addresses and ports in a *string* form, as shown in Listing 3.6, and are used in the computation of Time Delta through a series of comparisons between packets. When the most recent packet with the same Flow ID is reached, the relative time difference between packets is stored as the Time Delta.

Listing 3.6: Flow ID processing

```
40 data.insert(0, column='flow_id', value = (data['ip.src'].astype(str) + '.'
41                                          + data['ip.dst'].astype(str) + '.'
42                                          + data['udp.srcport'].astype(str) + '.'
43                                          + data['udp.dstport'].astype(str) + '.'
44                                          + data['tcp.srcport'].astype(str) + '.'
45                                          + data['tcp.dstport'].astype(str)))
46
47 data.insert(1, column='flow_id_2', value = (data['ip.dst'].astype(str) + '.'
48                                          + data['ip.src'].astype(str) + '.'
49                                          + data['udp.dstport'].astype(str) + '.'
50                                          + data['udp.srcport'].astype(str) + '.'
51                                          + data['tcp.dstport'].astype(str) + '.'
52                                          + data['tcp.srcport'].astype(str)))
```

The next field, Frame Lenght, denotes the length of each packet in bytes. This field provided valuable information about the size of the packets being transmitted across the network which could help identify potential anomalies or variations in data transfer sizes, which might indicate different types of network activities or protocols in use.

Lastly, the field Egress Ingress was added to determine the direction of the packet regarding that node. In the context of network traffic, egress refers to the outgoing traffic from a network interface or device, while ingress represents the incoming traffic. This field provided with insights into the directionality of the packets, allowing for an analysis of network flows from both perspectives which could prove useful in the identification of anomalies, congestion points or bottlenecks when applied in network simulation performance evaluation.

The Table 3.2 displays the input variables extracted from the UDP packets, along with an example value and the type of value it represents. These input variables are essential for analyzing network traffic dynamics and understanding various aspects of the UDP protocol. It is important to note that these variables can be categorized into three main types: binary, categorical, and continuous.

- Binary variables represent data that can take on one of two possible values, typically denoted as 0 and 1. They indicate the presence or absence of a specific characteristic of the network packet.
- Categorical variables which represent data that falls into specific categories or groups. These variables can have more than two possible values and do not possess a numerical relationship between them.
- Continuous variables, that are numerical variables that can take on a wide range of values within a certain range or interval. In network traffic analysis, continuous variables can provide quantitative information about various packet characteristics.

Table 3.2: UDP Input Field Types and Examples

#	Input Field	Type	Example
1	Time Delta	continuous	14.722 miliseconds
2	Frame Length	continuous	945 bytes
3	Egress or Ingress	binary	Egress

3.3.1.2 TCP fields

For TCP packets, the same fields used for UDP were considered. Although, additional fields were also included, namely TCP flags: *Ack, Push, Rst, Syn* and *Fin*, indicating acknowledgment, pushing, resetting, synchronization, and finalization, respectively.

TCP flags allow for examination on specific characteristics of TCP packets and reveal valuable information about the state and behavior of TCP connections. These are used to control and manage the communication process between TCP endpoints, furthermore revealing patterns of communication, such as the establishment or termination of connections, data transfer operations, and potential network issues such as dropped packets or re-transmissions. Additionally, in the context of network simulation, TCP flags could help identify connection anomalies and diagnose

connectivity or performance problems, for example, throught the use of flags like *Rst* which may occur due to connection errors or security incidents.

Table 3.3 displays all the input variables extracted from the TCP packets, along with an example value and the type of value it represents.

Table 3.3: TCP Input Field Types and Examples

#	Input Field	Type	Example
1	Time Delta	continuous	14.722 miliseconds
2	Frame Length	continuous	945 bytes
3	Egress or Ingress	binary	Ingress
4	Ack Flag	binary	1
5	Push Flag	binary	1
6	Reset Flag	binary	0
7	Syn Flag	binary	1
8	Fin Flag	binary	0

3.3.2 Transformations and Normalization

In the final pre-processing stage of the input data, two additional steps, referred to as distribution transformation and normalization, were carried out.

Distribution transformations involve applying mathematical operations or functions to the data in order to modify its distribution properties, such as logarithmic transformations or power transformations. These techniques help to address issues such as skewness in the data distribution with the aim of enhancing the data and potentially improve the results of the subsequent modelling process.

On the other hand, normalization entails adjusting the data such that they fall within an appropriate range before serving as input to the model. This is done by transforming the continuous variables into values between [0, 1] and the binary values to either be 0 or 1 for each packet. This range is generally preferred for machine learning models as it guarantees that all input features have a constant scale and prevents biases or dominance that may come from differing magnitude ranges of certain features during the model's training process.

Regarding normalization, the considered approach was the *Min-Max* normalization, also known as feature scaling. This normalization technique rescales the continuous variables to the specified range, [0, 1], while preserving the original distribution of the data and ensuring that all values fall within the desired range through the computation of Equation 3.1. However, *Min-Max* normalization is sensitive to outliers. When the data includes extreme values, they may have a disproportionate influence on the scaling process, leading the majority of the values to be compressed towards one end of the range, therefore the need to implement further transformations to deal with this problem.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

This is the case with the continuous variables in the obtained dataset, particularly the Time Delta variable, which demonstrates a distribution with high skewness and thus makes it more difficult for the model to predict on the whole spectrum.

To address this issue, two different types of transformation techniques were explored and evaluated to determine the most appropriate method for the given dataset: a *logarithmic* transformation and *Box-Cox* transformation.

The first of which, *Log* normalization, involves simply obtaining the logarithm of the continuous variables. This transformation is particularly useful when dealing with skewed data, where the majority of the values are concentrated towards one end of the distribution, as the log function compresses the range of values towards the lower end of the scale making the distribution more symmetric. It was applied with the aim of reducing the influence of extreme values and bring out patterns or relationships that may not be apparent in the original data.

On the other hand, the *Box-Cox* transformation is a statistical technique used to transform a non-normal or skewed dataset into a normal distribution. The main objective of the *Box-Cox* transformation is to stabilize the variance of a dataset and make it conform more closely to the assumptions of linear regression and other statistical models by applying a power transformation to the original data. This *Box-Cox* transformation is defined by the following Equation 3.2, where $\lambda_2 = 0.01$ while λ_1 is automatically calculated by the *Box-Cox* function in order to be optimal for the specific distribution.

$$x_{boxcox} = \begin{cases} \frac{(x+\lambda_2)^{\lambda_1}-1}{\lambda_1} & \text{, if } \lambda_1 \neq 0 \\ log(x+\lambda_2) & \text{, if } \lambda_1 = 0 \end{cases} \tag{3.2}$$

The following Figures 3.2 though 3.5 demonstrate these transformations when compared with the original distributions.

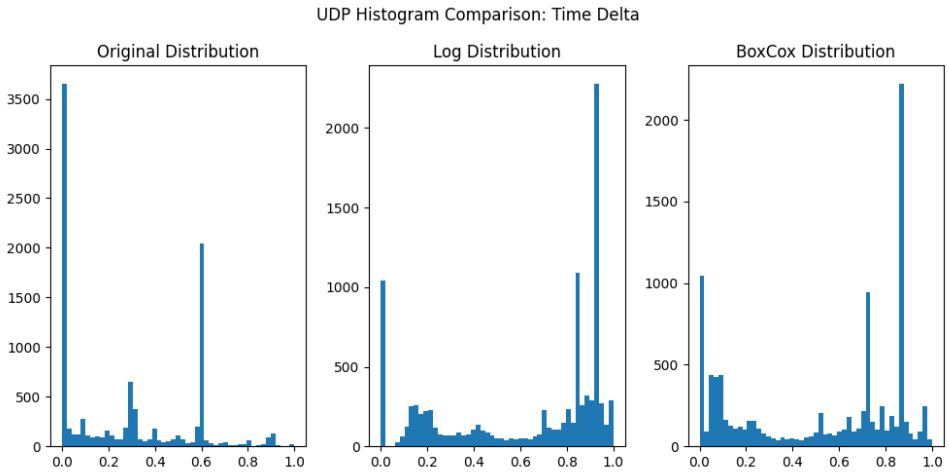

Figure 3.2: Transformation Comparison of Time Delta in UDP packets

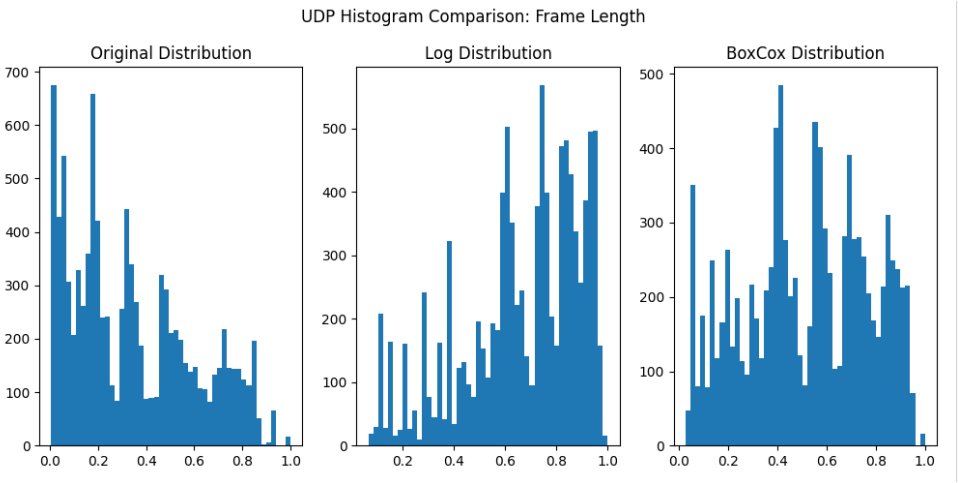

Figure 3.3: Transformation Comparison of Frame Length in UDP packets

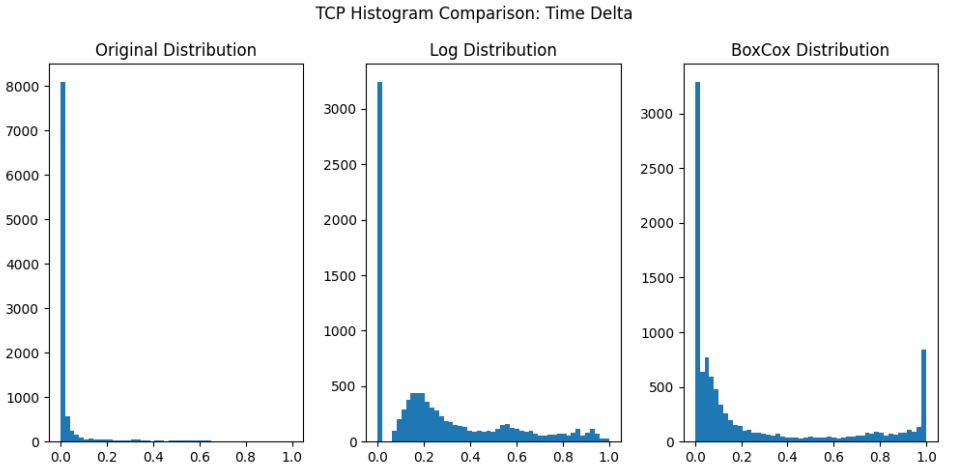

Figure 3.4: Transformation Comparison of Time Delta in TCP packets

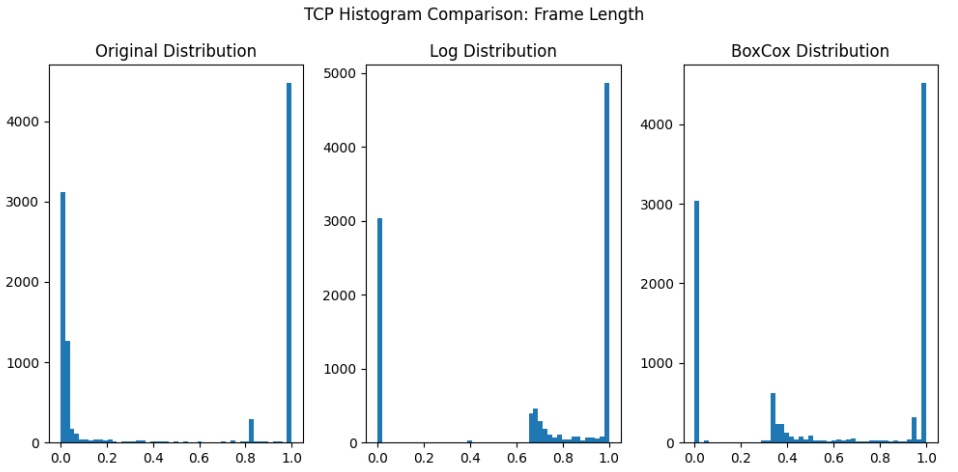

Figure 3.5: Transformation Comparison of Frame Length in TCP packets

To address the model's challenges in generating *Min-Max* distributions during training, an alternative approach using the *Box-Cox* distribution was tested for comparison. By comparing the performance of the model trained with both techniques, we hope to get a better understanding of the impact of different normalization methods and achieve better results regarding the model's efficacy.

3.4 Dataset Imbalances

Another challenge presented itself regarding discrete variables, namely TCP flags, in the form of data imbalances. Upon examination of the dataset it was observed that there were notable data imbalances caused by the varying frequency of appearance of certain flags in TCP packets, such as ACK and FIN whose distributions are shown in Figures 3.6 and 3.7.

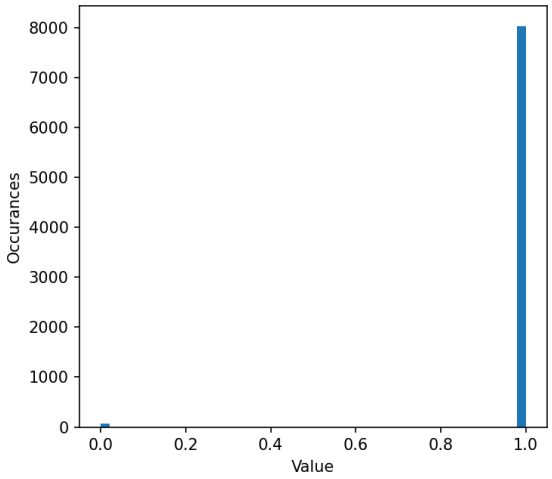

Figure 3.6: Data Imbalance in ACK flag

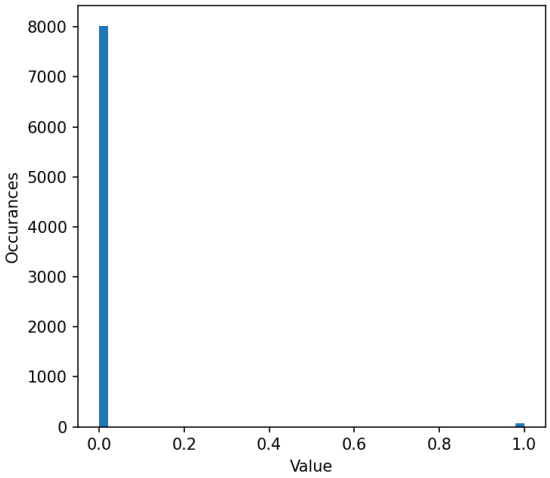

Figure 3.7: Data Imbalance in FIN flag

These flags play a crucial role in maintaining and terminating connections between network devices, however, their occurrences are not distributed evenly leading to these imbalances which will further hinder on the model's learning. Nevertheless, we believe that leveling out the dataset would be counterproductive, as these imbalances hold valuable information that offers unique insights into the flow of network traffic. As such, typical data augmentation tasks are not put in place, though another method is put in place in Section 4.6 as a countermeasure for this challenge with the addition of a "noise floor" to the original data.

3.5 Summary

For the development of this work, the model requires specific input data which was obtained by observing the traffic flow passing through a device connected to the desired network. In order to have a complete understanding of the network's communication, it was necessary to then, not only

analyse all the different packet's protocol fields, but also to make the distinction between ingress and egress packet flows.

The next step involved the data anonymization process which, by being done on-the-fly, allowed the protection of network user privacy while preserving the information that is necessary for further analysis.

After this step, the next stage was to extract the relevant features and fields from the packets. In this particular case, the focus was on capturing the protocol components that could prove useful for the model such as packet timestamps, packet length, protocol types and flags.

Following the last step of the feature extraction process, the data was reformatted using the CSV format, which made it possible to express data in a structured manner, with each feature or field occupying its own column in the table. In addition, in order for the model to be able to process the data, these inputs needed to be normalised and even transformed through some log functions in order to produce better results concerning the model. This transformed data could then be fed into the model, where the pertinent characteristics extracted from anonymized packets provide the necessary data for the learning task and predictions based on observed network traffic patterns.

Altogether, the process of acquiring, anonymizing, extracting pertinent features, transforming the input data into CSV format, normalizing and applying log functions is essential for enabling the model to analyse and comprehend network traffic effectively. It permits the utilisation of real-world data while prioritising privacy and ensuring the model's requirements are met.

Chapter 4

Traffic Generator Model

The primary objective of this dissertation was to develop an algorithm that was able to generate accurate estimates when it came to the generation of synthetic network packets. In doing so, we aimed to overcome the limitations that are associated with relying solely on real-world data for network traffic analysis and experimentation. Our goal was to provide a reliable and scalable solution that could generate synthetic packets with a high degree of fidelity to real traffic patterns.

To achieve our objective, we drew inspiration from the state-of-the-art techniques in machine learning, specifically Generative Adversarial Networks. GANs have demonstrated remarkable success in generating realistic synthetic data across various domains, including images, text, audio and, as explored in 2.2.2.1, network traffic data. In this context, we recognized the potential of GANs and leveraged their capabilities to develop our model.

An essential aspect of network traffic is its time-dependent nature, with patterns and characteristics varying over different time intervals. To capture these temporal dynamics accurately, we incorporated time-dependent algorithms within our GAN model and, in doing so, we hoped for these algorithms to enable our model to generate synthetic network packets that exhibit realistic timing patterns, such as burstiness, periodicity, or fluctuations, as observed in actual network traffic.

4.1 Technologies Used

For the development of this work, an *Anaconda* environment [28] was initially set up which facilitated an organized workflow, ensuring that all the necessary libraries and dependencies were readily available without conflicts and, furthermore, allowing for a consistent and reproducible setting for our experiments and analyses.

In the *Anaconda* environment, we relied on *Python*[29] as the primary programming language due to its versatility, ease of use, and extensive libraries that support various data science and machine learning tasks. *Python* is also a flexible and scalable language capable of handling large datasets and complex models and, additionally, has interoperability with other languages and tools[30], enabling them to complement one another.

Regarding the machine learning framework, we opted to use *Keras*, which is integrated within the *TensorFlow* ecosystem. While *Keras* provides a high-level API that allows us to easily design and build deep learning models, *TensorFlow* acts as the backbone, providing the underlying computational capabilities for training and deploying these models effectively.

4.2 Architecture

As it was previously explored in Chapter 2.2.2.1, GANs are a type of deep learning model that consists of two main components: a generator and a discriminator. The generator network learns to produce synthetic network packet data, while the discriminator network evaluates the authenticity of this data by trying to distinguish them from real packet data.

The discriminator's role is then to classify whether a given input is real or fake. It is intended to learn to differentiate between the real data samples and the ones generated by the generator with the goal of becoming more accurate as the training progresses. To achieve this, the discriminator architecture depicted in Figure 4.1 was chosen as follows:

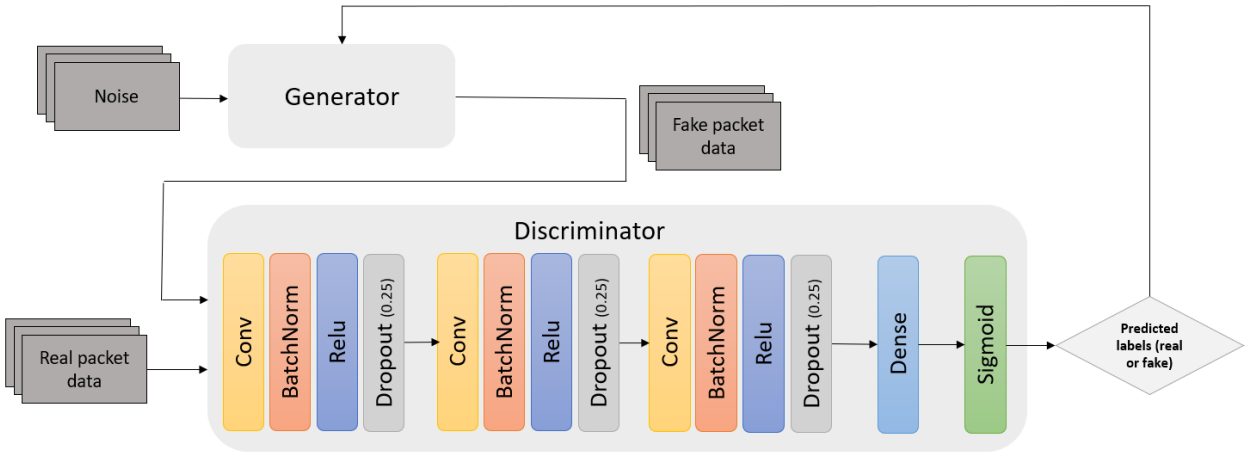

Figure 4.1: GAN's structure and training

- Convolutional Layer (Conv): The convolutional layer is the fundamental building block of a CNN. It applies a set of learnable filters (also known as kernels) to the input data where each filter slides across the input, performing element-wise multiplication and summing the results to produce a feature map. This way, allowing the network to capture local patterns and extract meaningful features.

- Batch Normalization (BatchNorm): Batch normalization is a technique that normalizes the intermediate activations within a batch. It helps address the internal covariate shift problem, where the distribution of inputs to a layer changes during training, causing the network to

constantly adapt to these changes. Additionally, this layer reduces the sensitivity of the network to the initialization of weights and gradients, therefore improving its overall stability and convergence speed.

- Rectified Linear Unit (ReLU): ReLU is an activation function that introduces non-linearity into the network, which map any negative input to zero and keeps positive values unchanged. ReLU activations are computationally efficient and help the network model complex relationships between features. By introducing non-linearity, ReLU enables CNNs to learn more expressive representations and capture highly nonlinear patterns in the data.

- Dropout: Dropout is a regularization technique that helps prevent overfitting of the model by randomly setting a fraction of the input units to zero during training, forcing the network to learn redundant representations and making it more robust. Dropout can be particularly beneficial in CNNs as it helps prevent the network from relying too heavily on specific features or activations, promoting the learning of more generalized features.

- Dense layer: This is a fully connected layer, where each neuron is connected to every neuron in the previous layer. In GANs, the discriminator's output value is often interpreted as the probability that the input sample is real (i.e., from the real data distribution) or fake (i.e., generated by the generator). Here, the purpose of adding this Dense layer with a sigmoid activation function is to produce a single output value that represents the discriminator's prediction, in which the sigmoid activation function squashes the output value to a range between 0 and 1, making it interpretable as a probability. Mathematically, the sigmoid activation function is defined as:

$$\sigma(x) = \frac{1}{1+e^{x}} \tag{4.1}$$

When it comes to the generator in our task, there are multiple algorithms that can be considered. To ensure a comprehensive evaluation, we began by defining a set of baseline algorithms that can serve as a reference for comparison. These baseline algorithms provide a solid starting point and allow us to establish a performance benchmark.

In addition to the baseline algorithms, we also explore a proposed algorithm for the generator. This proposed algorithm introduces techniques and modifications to improve upon the baseline's performance. To thoroughly assess its effectiveness, we investigate two variations of the proposed algorithm, each with its unique approach and adjustments.

By exploring these variations of the proposed algorithm alongside the baseline algorithms, we aim to gain a deeper understanding of their respective strengths and weaknesses. We hope that this comparative analysis will provide valuable insights into the advancements offered by the proposed algorithm and its potential for enhancing the generator's capabilities.

4.2.1 Baseline Generator Models

Starting with the baseline models, these can serve as a benchmark, enabling a more informative evaluation of a trained model. By later comparing our model with the baselines we hope to better understand the flaws of the baseline and display a stronger overall performance with our approach. If our trained model does not perform better than the baseline model, then we must conclude that the model's added complexity does not provide enough benefit.

Here, two models were considered: an LSTM generator, and a TCN generator.

4.2.1.1 LSTM generator

The first of these baselines consists on a simple LSTM generator, whose model is inspired by what was done in [22]. This architecture encompasses a single LSTM layer with multiple cells, followed by a softmax layer and a final Dense layer as depicted in Figure 4.2. Here, softmax is applied to convert the previous layer's output into a probability distribution over the different categories, where each category represents a possible outcome, and the softmax function ensures that the probabilities sum up to 1. While the Dense demonstrates a similar purpose as in the discriminator's network by outputting a single value for each packet field between 0 and 1.

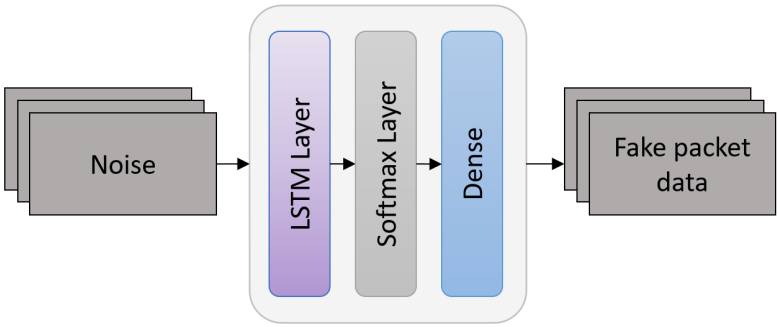

Figure 4.2: LSTM Generator's structure

4.2.1.2 TCN generator

Convolutional neural networks, more popularly known as CNNs, are often used for image classification tasks; nevertheless, these networks have been shown to be useful tools for sequence modelling and forecasting when the appropriate adjustments are made.

The concept of sequence modelling in the context of deep learning has, up until fairly recently, been primarily linked with recurrent neural networks, such as LSTM. Thought, according to [31], convolutional networks should be taken into account as a possible choice for modelling sequential data as they were able to demonstrate that these networks may achieve greater performance than RNNs in many tasks while avoiding frequent limitations of recurrent models, such as the issue of exploding or vanishing gradients or a lack of memory retention.

TCNs (Temporal Convolutional Networks) are then a possible approach, which were determined to be reviewed in the framework of this study as they have not been explored previously in the state of the art in the context of network traffic generation. This generator model [32] was implemented though the use of unit blocks which contain the dilated, causal 1D convolutional layers with the same input and output lengths, with additional batch normalization layers and ReLU (Rectified Linear Unit) activation functions as depicted in Figure 4.3.

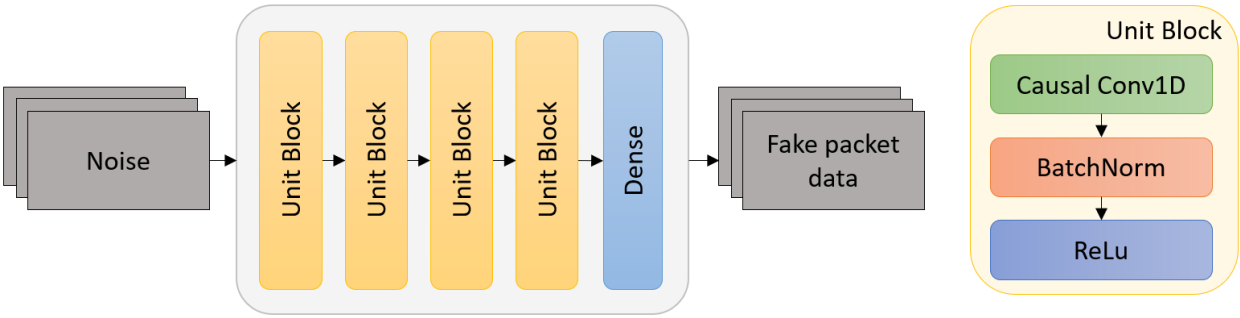

Figure 4.3: TCN Generator's structure

In order for these convolutional layers to be causal, each element in the output sequence can only depend on the elements in the input sequence that come before it [32], that is, each element in the output sequence, designated as the i-th element for i belonging to the set {0,..., input_length - 1}, may only rely on the elements in the input sequence with indices {0,..., i}. In other words, these causal convolutional layers guarantee that the information flow is constrained to the past and that no future components in the input sequence are taken into account while calculating the output, therefore allowing for forecasting in the generation process.

4.2.2 Proposed Generator Model

With this work, we set out to propose a model that incorporates the two previous baseline network architectures and combines their strengths to create a novel approach inspired by the works of [16] and [31]. This was achieved with a unique Temporal Convolutional Network for each variable, and further providing the individual outputs to a shared LSTM network common to all variables as is depicted in Figure 4.4.

TCNs are especially effective at extracting local patterns in sequential data; by using these networks for each individual variable, we hope that the model is able to capture and learn variable-specific temporal patterns and dependencies. These are implemented as before, with two unit blocks per variable, with each comprising a causal convolutional layer, a batch normalization layer and a ReLU activation function.

After extracting the temporal patterns associated with each individual variable using these TCNs, the model employs shared LSTM layers across all variables. The potential of LSTM layers to represent long-term relationships and capture global patterns in sequential data is well established by the literature as explored in Chapter 2. By sharing these layers, we then hope that the

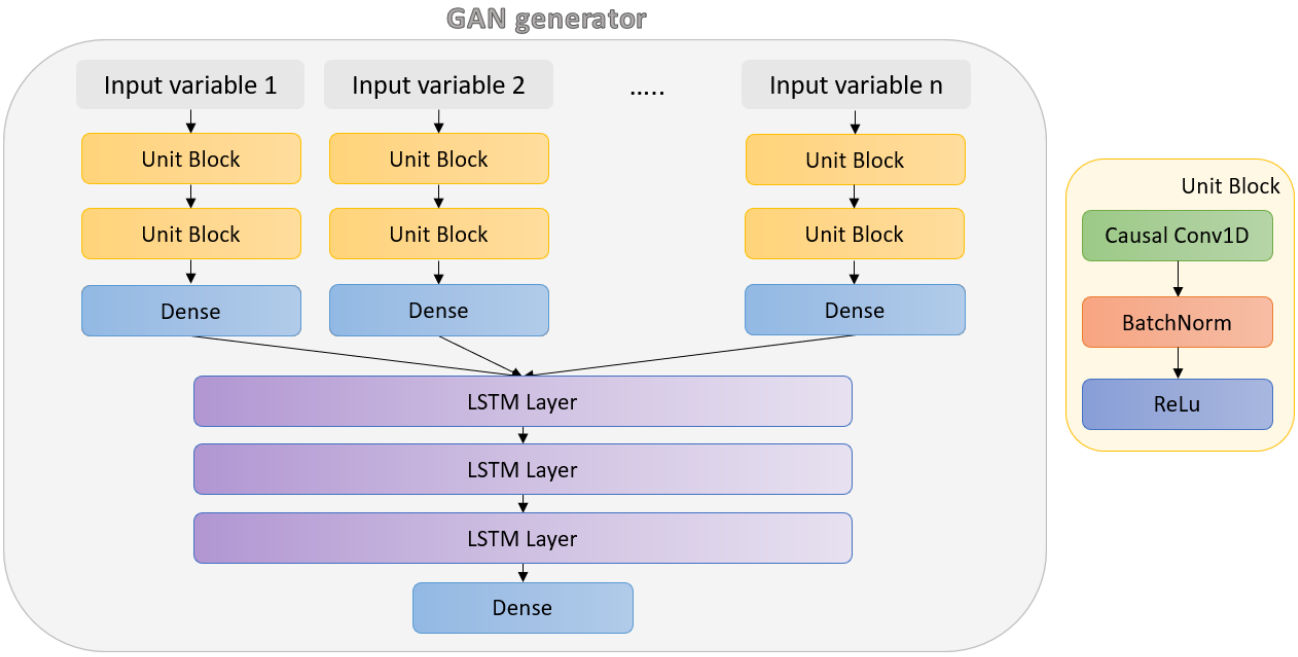

Figure 4.4: Proposed Generator's structure

model may leverage the inter-dependencies between variables and capture the complex relationships that may exist across the entire input sequence. Which, in turn, might enable the model to express and predict cross-variable interdependence and global temporal trends.

An additional simplified model was tested, in which the LSTM layers were substituted as, presented in Figure 4.5, by a single Dense layer.

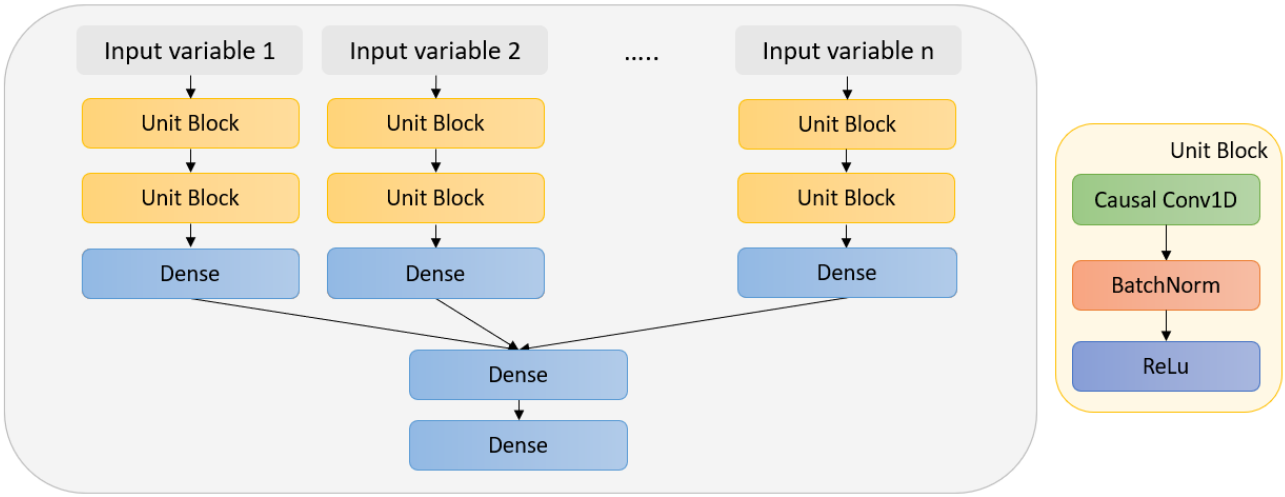

Figure 4.5: Simplified Generator's structure

4.3 Training

As previously stated, the model was trained iteratively, with the generator network learning to generate more accurate and realistic packet data while the discriminator network, in an adversarial manner, improved its ability to differentiate between real and synthetic packet data. Typically, the GAN training process is divided into separate steps for the discriminator and the generator. This strategy is required due to the adversarial nature of GANs as, training both the generator and the discriminator at the same time, might result in instabilities and oscillations in the learning process. By dividing the training processes, it enables each network to update separately and progressively learn, eventually reaching an equilibrium where the generator provides samples realistic enough to fool the discriminator. As such, the training process involved the following steps:

- **Initialization**: The input data is divided into training set (80%) and validation set (remaining 20%). The GAN's generator and discriminator are initialized with random weights.
- **Adversarial Training Loop**: The training process alternates between updating the generator and the discriminator in a series of data batches. During each iteration of the loop:
  - **Discriminator Training**:
    * Real Data: A batch of real data samples from the training dataset is passed to the discriminator. Here, the samples are labeled with '1' in order to be associated as real samples by the discriminator.
    * Generated Data: The generator takes a random noise as input with the same format as a batch of real samples and generates a corresponding batch of synthetic samples. In turn, these synthetic samples are then labeled with '0' in order to be associated as fake samples by the discriminator.
    * Discriminator Loss: The discriminator is trained to minimize its loss by correctly classifying the real and generated samples.
    * Gradients and Backpropagation: The gradients of the discriminator's loss with respect to its parameters are computed, and backpropagation is used to update the discriminator's weights.
  - **Generator Training**:
    * Generated Data: The generator takes a random noise as input with the same format as a batch of real samples and generates a corresponding batch of synthetic samples.
    * Discriminator Feedback: The synthetic samples generated by the generator are passed through the discriminator. The discriminator evaluates these synthetic samples and produces an output for each one. This output is later taken into account by the generator.
    * Generator Loss: The generator is trained to maximize the GAN's loss, aiming to generate samples that the discriminator incorrectly classifies as real. During this

step, the discriminator's training is frozen due to the fact that, if the discriminator were updated concurrently with the generator, it would give a changing target for the generator. This would make it more difficult to stabilise the generator's training process and, as a consequence, to optimise the generator's objective function. Therefore, by keeping the discriminator constant, we provide a more consistent and stable environment for the generator to learn.

    * Gradients and Backpropagation: The gradients of the generator's loss with respect to its parameters are computed, and backpropagation is used to update the generator's weights.

- Iteration: Previous steps are repeated for a certain number of iterations which depends on the batch size chosen and the number of samples in the input training data. This iterative training process allows the generator and discriminator to improve their performance iteratively through adversarial competition and feedback.

4.4 Network Parameters

One of the challenges of this work is to identify the optimal combination of parameters that yield the best performance for the specific model architecture that is being explored, while avoiding failure modes and mode collapse.

GANs are notoriously hard to train because of the adversarial nature of the learning process. The generator and discriminator networks compete against each other which can lead to instability in the training process, causing issues like mode collapse (where the generator produces limited variations of the data) or vanishing gradients (where the gradients become too small for effective learning), therefore hindering the reach of a point of equilibrium.

Several key hyperparameters, such as the choice of loss function, learning rate, optimizer, and number of epochs, directly impact the stability and convergence of GAN training, furthermore determining the success or failure of the learning process.

4.4.1 Learning Rates

The learning rate is the first vital hyperparameter that influences the training dynamics of GANs. A learning rate that is too high may lead to unstable training, causing the networks to oscillate rapidly between different states, making it difficult for the networks to reach equilibrium and, furthermore, preventing convergence. This phenomenon is known as "exploding gradients" and can hinder the convergence of the GAN, causing the training process to fail.

Conversely, if the learning rate is too low, the GAN's training can become excessively slow. The updates to the network's parameters become tiny, and it takes a long time for the model to learn meaningful patterns from the data. Moreover, an extremely low learning rate might cause the GAN to get stuck in local minima, which are suboptimal points in the loss landscape, preventing it from finding the global optimum.

Fine-tuning the learning rate is then a crucial step to finding the right balance between rapid convergence and stability during GAN training. Throughout the development of this work various values were tested, from which, the following were chosen:

- Discriminator's Learning Rate = 0.00003
- Generator's Learning Rate = 0.001

4.4.2 Training Epochs

The number of epochs is another crucial hyperparameter that profoundly influences the training process of machine learning models. An epoch refers to a single pass of the entire dataset during training and responsible for determining how much the model learns from the data and how well it generalizes to unseen examples.

When the number of epochs is set too low, the model might not have enough opportunities to learn from the data fully. In such cases, the model's performance may be suboptimal and might not capture the underlying patterns and complexities in the data, therefore, generating new data that may lack accuracy and fail to generalize well on new data.

On the other hand, an excessively high number of epochs can lead to overfitting, which occurs when the model becomes too specialized in the training data, capturing noise and irrelevant details instead of learning the general patterns. As a result, the model's performance on the training data might be excellent, but it will also perform poorly on new, unseen data.

Through a series of tests, the number of epochs was chosen to be 25 as by this epoch the models we observed to have reached a balance.

4.4.3 Optimizer

The optimizer is the algorithm which adjusts the network's weights and biases to minimise the loss function by calculating and iteratively updating the gradients of the loss function based on the parameters. The optimizer's purpose is to then discover the most suitable parameter values that minimise the loss and so improve the network's predictions. One such popular and effective optimizer is the Adam optimizer which has an adaptive nature, as it dynamically adjusts the learning rates for each parameter during training.

This adaptiveness is achieved through an random initialization of the network's weights and biases, and as the training progresses, the computation of the loss function's gradients with respect to each parameter using backpropagation. Then, the Adam optimizer updates the parameters iteratively which ensures that the network converges towards an optimal solution efficiently.

4.4.4 Loss Functions

Loss functions are one of the most significant parts of neural networks since they are directly responsible for adapting the model to the available training data (when combined with optimisation functions). Regarding the training process of the generator, the Mean Squared Error was employed

as the loss function as a way to we encourage the algorithm to produce synthetic data that closely resembles the real data, thereby enhancing the quality of the generated samples.

Regarding the discriminator network's training, the loss is computed using the algorithm's output and a binary cross-entropy loss function, though a focal loss function was also implemented for comparison regarding the model's performance. Both of these loss functions aim to measure the difference between predicted probabilities and ground truth labels, but they have different characteristics which will be explored.

4.4.4.1 Mean Squared Error

Mean Squared Error (MSE) is a widely implemented loss function in numerous machine learning tasks that may be used as a measure of dissimilarity between generated outputs and target (real) data in the context of a generator network.

The formula for MSE is straightforward: it calculates the average of the squared differences between the generated samples (Y_i) and the real data (Y_i) as given in Equation 4.2, where N is the number of data points.

$$MSE_{Loss} = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{4.2}$$

By squaring the differences, larger errors are penalized more heavily, emphasizing the importance of reducing significant discrepancies between the generated and real data points. This characteristic of MSE makes it particularly suitable for applications where we want to minimize large errors, such as image generation or time series prediction.

This aligns with the primary objective of producing synthetic data samples that closely resemble the real data, as MSE facilitates the learning task by quantifying the discrepancy between the two sets of samples.

4.4.4.2 Binary Cross Entropy

Binary Cross Entropy (BCE), also known as Binary Log Loss, is a loss function that is extensively employed in machine learning, especially in binary classification issues as is the problem in the GAN's discriminator. This loss is given by Equation 4.3, which calculates the cross-entropy between the predicted and true binary labels, where:

- y represents the true binary label (0 or 1)
- p represents the predicted probability of the positive class (between 0 and 1)

$$BCE_{Loss} = -[y * log(p) + (1-y) * log(1-p)] \tag{4.3}$$

When the resulting loss value is low, it indicates that the model's predictions are accurate. Conversely, when the model's loss value is bigger, it will be more harshly penalised, which then encourages the model to alter its parameters to minimise the gap between the predicted and real labels.

4.4.4.3 Focal Loss

Focal loss is an extension of the binary cross-entropy loss, though, designed specifically with the aim of addressing the issue of class imbalance in binary classification tasks. This is the case of some binary variables in our dataset, in which one class has considerably more data than the other resulting in class imbalance and biassed model training. Focal loss introduces a modulating factor that reduces the importance of well-classified instances in favour of hard or misclassified cases, therefore enabling the model to concentrate more heavily on troublesome samples which should enhance overall performance of the model [33].

The Focal Loss function is defined in Equation 4.4, where:

- γ is the modulating factor that controls the rate at which the loss is reduced for well-classified examples. In this work it was set to $\gamma = 2$

- α is a weighting parameter that can be used to adjust the rate of loss reduction for the negative class. In this work it was set to $\alpha = 0.25$

$$Focal_{Loss} = \begin{cases} -\alpha(1-p)^{\gamma} log(p) & \text{, for y = 1 (positive class)} \\ -(1-\alpha) * p^{\gamma} * log(1-p) & \text{, for y = 0 (negative class)} \end{cases} \tag{4.4}$$

4.5 Validation

When analysing machine learning models, many strategies may be applied to gain insights into their performance and behaviour.

During the development of this work, and in order to evaluate the accuracy of our algorithm, we compared the statistical properties and performance metrics of the generated synthetic packets with those of real network traffic. These metrics included variable distribution comparison, correlations between variables and, additionally, the following metrics:

- **Mean Absolute Error (MAE)** measures the average magnitude of the errors in a set of predictions, while maintaining its units in the same as the predicted values, making it easier to interpret the results. Equation 4.5 depicts its mathematical definition, where $\hat{x}_i$ is the predicted data and x_i is the real data, with N being the number of samples:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |x_i - \hat{x}_i| \tag{4.5}$$

- **Mean Absolute Percentage Error (MAPE)**, similarly to MAE, provides a measure of the accuracy of the model's predictions as a percentage of the actual values given by Equation 4.6.

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{|x_i - \hat{x}_i|}{x_i} \tag{4.6}$$

- **Root Mean Squared Error (RMSE)** that measures the deviation of the predictions from the actual values, taking into account the magnitude of the error and given mathematically by the Equation 4.7.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2} \tag{4.7}$$

4.6 Data Augmentation

Data augmentation plays a crucial role in enhancing the performance and generalization ability of machine learning models. By augmenting the available training data, we can effectively increase its diversity, thereby enabling the model to learn more robust and representative features.

In the context of machine learning, data augmentation techniques are employed to artificially create new training samples that resemble the original data distribution. These augmented samples are generated by applying various transformations, distortions, or perturbations to the existing training data, with the objective of introducing variations that mimic real-world scenarios and enrich the training set.

In some cases, certain machine learning models may struggle to accurately predict across the entire spectrum of input data. This limitation can arise due to various factors, such as model architecture, training data characteristics, or the complexity of the problem at hand. To address this challenge, one potential solution that has been explored is data augmentation by setting a "noise floor" on the input training data as can be seen implemented in Figure 4.6 and Figure 4.7 on both continuous and discrete variables, respectively.

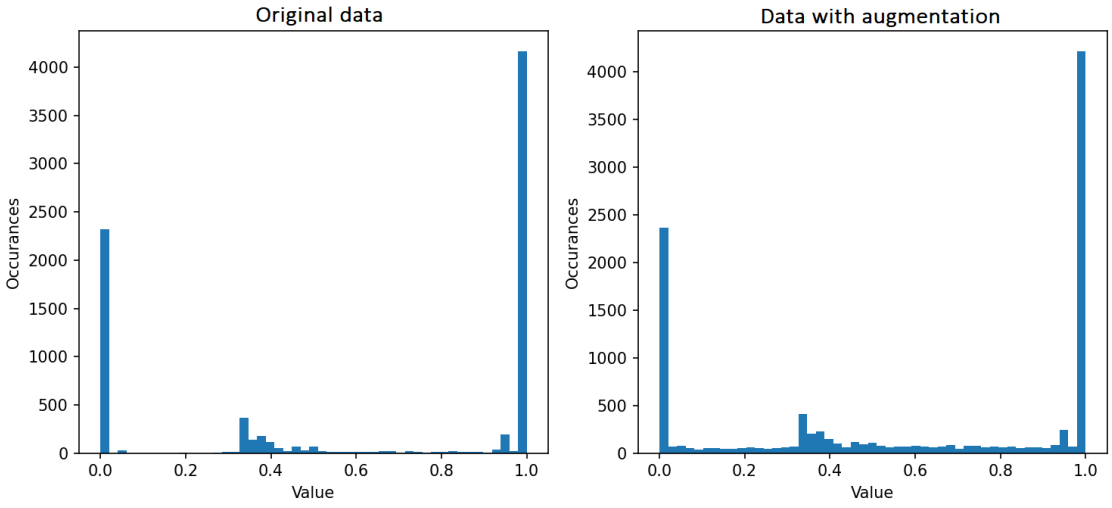

Figure 4.6: Variable "Packet Length" with Noise Floor

It's worth noting that this measure was implemented in light of experimentation, where some models were not able to predict on the whole output spectrum. With the arise of this scenario, it was thought that these models could benefit of data augmentation where data is spread out throughout the whole spectrum in an attempt to obtain better model predictions.

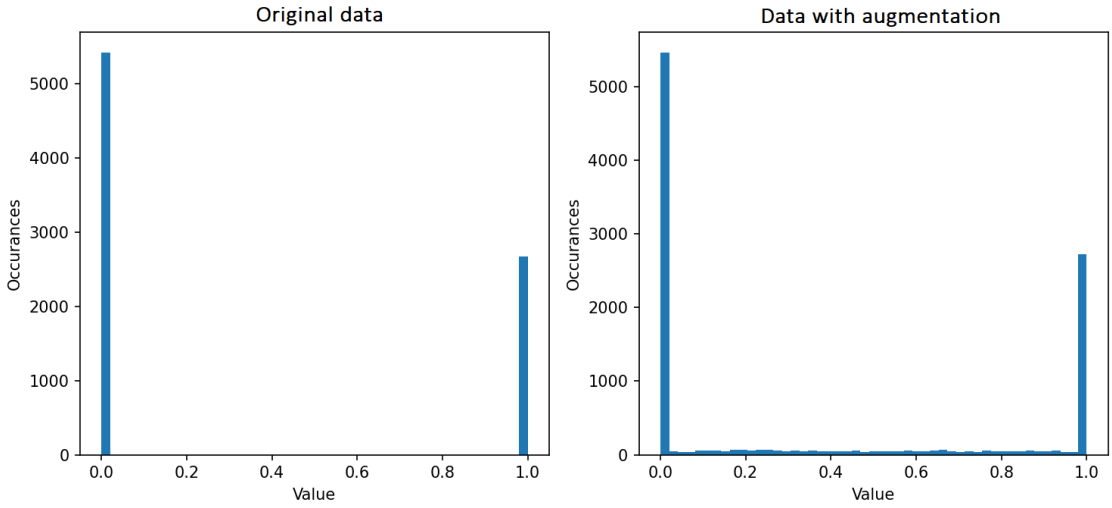

Figure 4.7: Variable "Flag Push" with Noise Floor

Moreover, this noise floor should be removed from the generated output as it isn't representative of real data.

4.7 Summary

This chapter provided a comprehensive overview of the model developed in the context of this work to generate accurate synthetic network packets, with the primary objective of overcoming the limitations of the state of the art solutions. To achieve this, the model was built using Generative Adversarial Networks, a powerful technique known for generating realistic synthetic data and incorporated time-dependent algorithms to capture temporal dynamics in network traffic accurately.

For this purpose, various technologies were used, including the *Anaconda* environment with *Python* as the primary programming language, and *Keras* integrated with *TensorFlow* as the machine learning framework.

To capture inter-variable dependencies, the suggested generator architecture integrated LSTM and TCN networks, alongside the deployment of baseline LSTM and TCN generator models for model comparison. The evaluation methods included model's accuracy, statistical properties and performance metrics of the generated synthetic packets when compared with real network traffic. Data augmentation with the addition of a noise floor was also utilized in order to increase generated spectrum and, furthermore, enhance the model's generalization ability.

Chapter 5

Evaluation and Result Analysis

In this next chapter, the results of our experimental phase are explored. Here, an in-depth analysis of the obtained empirical results using our proposed methodology are presented, shedding light on the effects of various parameters. By meticulously fine-tuning and optimizing these essential elements, we seek to understand their influence on the overall's model behavior and identify ideal configurations for the best results.

To ensure an adequate assessment, we use a variety of evaluation metrics explored in Section 4.5 that are tailored to capture specific aspects of the model's behaviour and, furthermore, analyse the output disparities that arise from subjecting each model to different input transformations and data augmentation techniques. Additionally, we also delve into the differences observed when we applied both the Binary Cross Entropy and the Focal Loss functions in the discriminator's learning, in hopes to identify the mode that proves to be the most suitable match for the unique characteristics of each model.

5.1 Generated Data Binarization

In the context of machine learning models, it is essential to recognize that, while they can handle binary classification tasks and provide probabilities that seem binary-like, they inherently produce continuous outputs rather than binary ones.

Given that binary variables can only take two distinct values within the range of 0 to 1, to make binary decisions based on these continuous outputs, a binarization process was employed with a threshold of 0.5. Under this transformation, any values equal to or below 0.5 were categorized as 0s, while values exceeding 0.5 were considered as 1s.

By implementing this binarization process, we ensured that the binary variables were converted into a format that was compatible with the normalized inputs, allowing for consistent and meaningful comparisons.

5.2 Results

Concerning the obtained results, there are the following generator models to consider: LSTMs, TCNs, the original proposed model and, finally, the simplified version of this proposed model.

To achieve these results, we employed the methods explored earlier in Section 4.5, such as distribution plots to visualise the frequency distribution of the data and identify any potential outliers or skewed distributions. Furthermore, autocorrelation plots were used to examine the presence of any patterns or correlations in the data, and mathematical metrics were used to quantify various aspects of the dataset.

While our subsequent result discussion and analysis done in Section 5.4 will take into account all variables, the focus will now be on a concise set of only four variables for a more concise result presentation: two with continuous values, Time Delta and Length, and two discrete variables, Flag ACK and Flag PUSH. These have been chosen as representative of the entire dataset, and include all of the continuous variables, as well as a highly imbalanced discrete variable, Flag ACK, and a more well-balanced discrete variable, Flag PUSH, allowing us to draw more meaningful insights from a broader perspective.

5.2.1 LSTM generator

Beginning with the exposition of the first baseline generator, the distributions and auto-correlations plots are represented below, where only the two simplest scenarios are presented: no data augmentation is added to the input and the discriminator's loss function is *binary cross-entropy*, with both transformations: *Min-Max* and *BoxCox*. Additionally, the evaluation metrics of this generation network are exposed in Table 5.1.

Table 5.1: LSTM network metrics

Augmented data	Input transformation	Loss Function	Evaluation Metrics		
			MAE	***MAPE***	***RSME***
Without Augmentation	*Min-Max*	*BCE Loss*	0.20742	9.122	0.43957
		Focal Loss	**0.20354**	**8.951**	0.43802
	Box-Cox	*BCE Loss*	0.23864	10.495	0.45321
		Focal Loss	0.23232	10.216	0.45674
With Augmentation	*Min-Max*	*BCE Loss*	0.25246	11.203	0.47493
		Focal Loss	0.25860	13.543	**0.43403**
	Box-Cox	*BCE Loss*	0.23765	10.452	0.45355
		Focal Loss	0.24614	12.593	0.44529

As such, Figures 5.1 through 5.5 represent the results obtained for the following scenario parameters:

- *Min-Max* normalization
- Without data augmentation
- *Binary Cross-Entropy* Loss

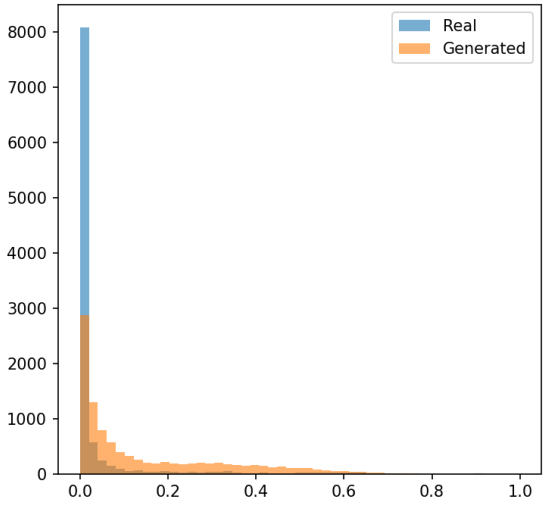

Figure 5.1: Time Delta Distribution with LSTM generator

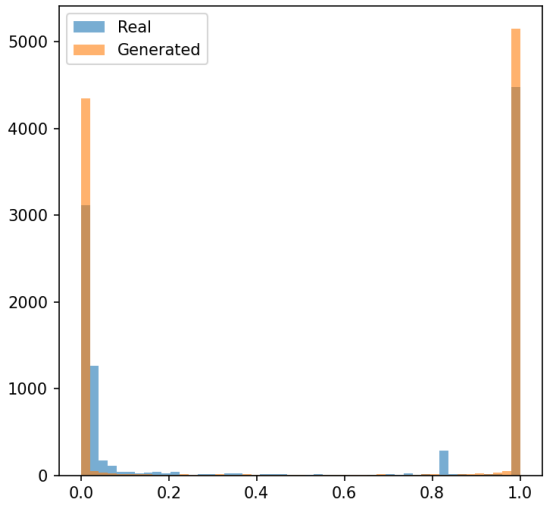

Figure 5.2: Length Distribution with LSTM generator

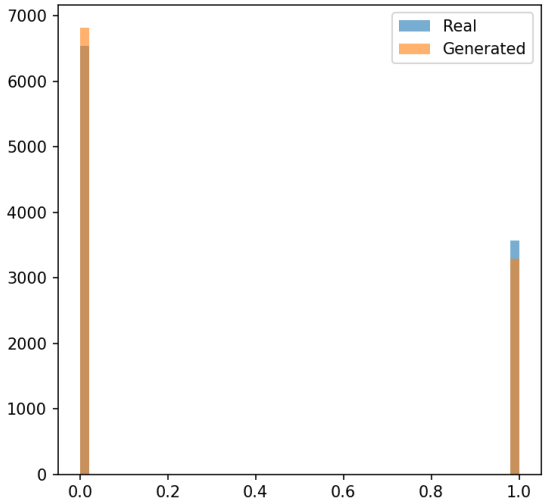

Figure 5.3: Flag PUSH Distribution with LSTM generator

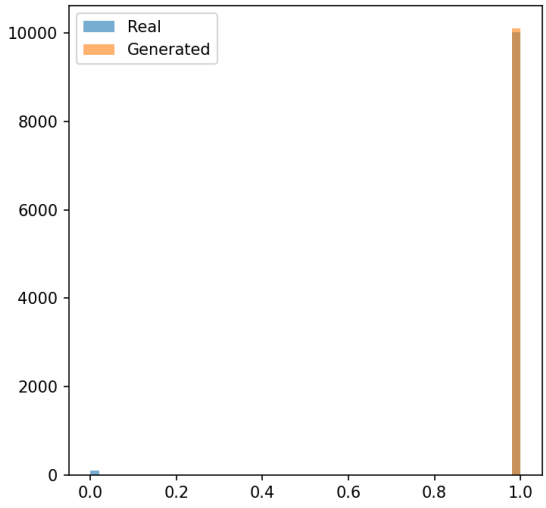

Figure 5.4: Flag ACK Distribution with LSTM generator

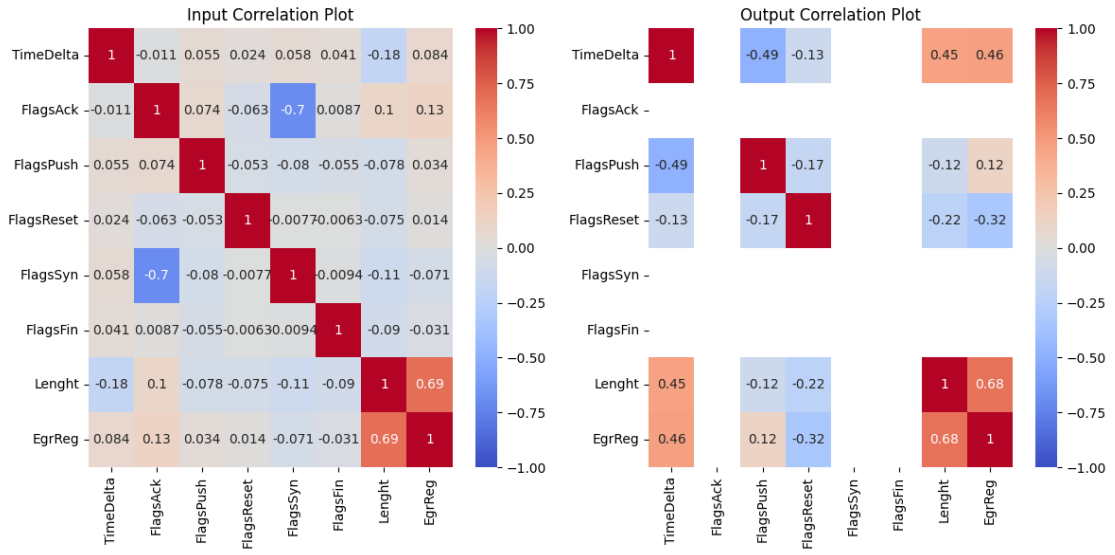

Figure 5.5: Variable Correlation Plot with LSTM generator

The results of the next scenario are depicted below in Figures 5.6 through 5.10, where the following parameters were considered:

- *Box-Cox* transformation
- Without data augmentation
- *Binary Cross-Entropy* Loss

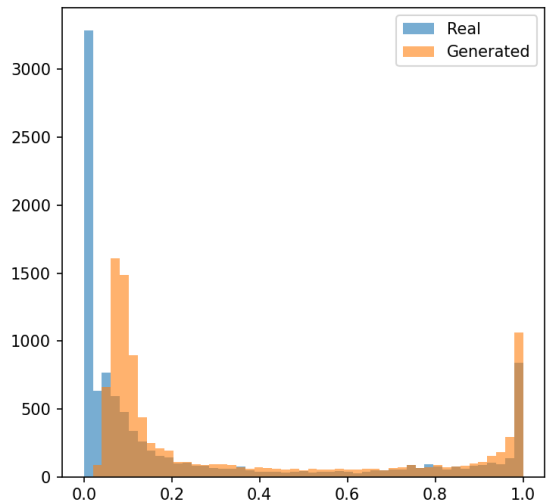

Figure 5.6: Time Delta Distribution with LSTM generator

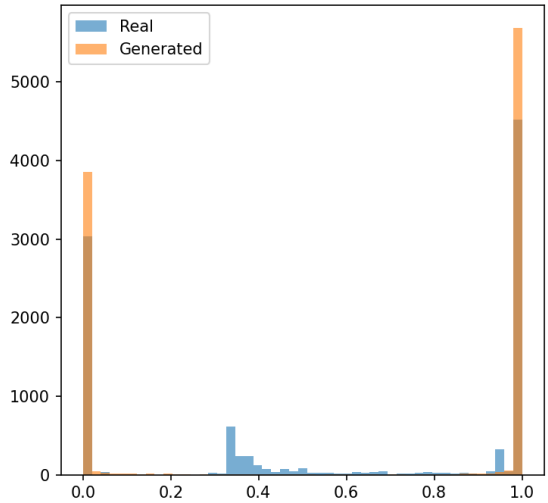

Figure 5.7: Length Distribution with LSTM generator

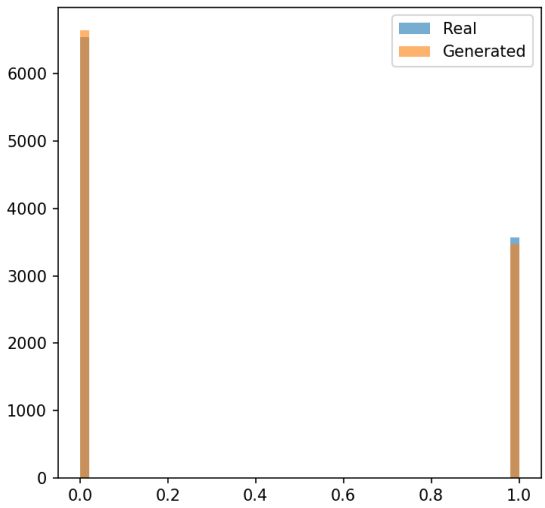

Figure 5.8: Flag PUSH Distribution with LSTM generator

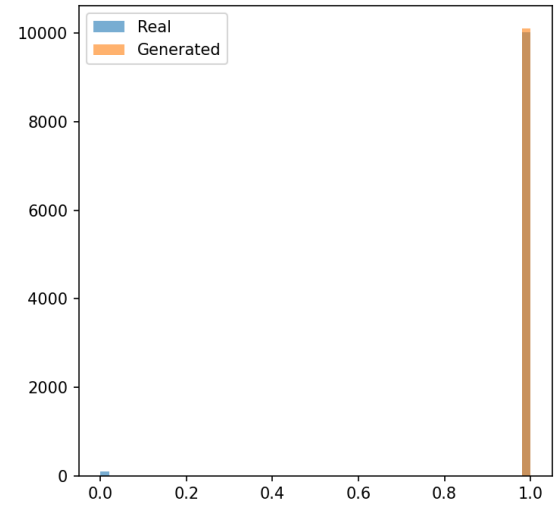

Figure 5.9: Flag ACK Distribution with LSTM generator

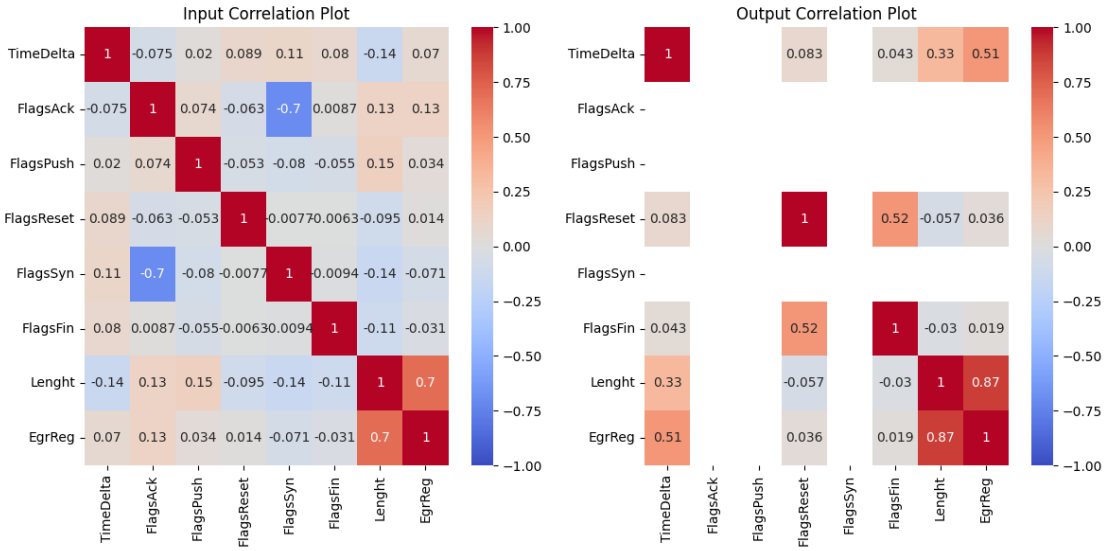

Figure 5.10: Variable Correlation Plot with LSTM generator

5.2.2 TCN Generator

Next, the remaining baseline generator's results are presented, where the distributions and auto-correlations plots for the generated data in the same two cases are showcased. By comparing the results of the second generator with those of the first, valuable insights are obtained regarding their respective strengths and weaknesses, later discussed in Sector 5.4.

Table 5.2 presents the evaluation metric values obtained for this TCN generation network.

Table 5.2: TCN network metrics

Augmented data	Input transformation	Loss Function	Evaluation Metrics		
			MAE	***MAPE***	***RSME***
Without Augmentation	*Min-Max*	*BCE Loss*	0.22617	9.152	0.46472
		Focal Loss	0.22969	9.294	0.45467
	Box-Cox	*BCE Loss*	0.23947	9.689	0.46098
		Focal Loss	0.24299	9.832	0.46228
With Augmentation	*Min-Max*	*BCE Loss*	**0.18726**	**7.577**	**0.41440**
		Focal Loss	0.28532	8.593	0.44944
	Box-Cox	*BCE Loss*	0.29792	12.055	0.46831
		Focal Loss	0.30525	12.201	0.47712

Regarding the first scenario, its results are represented in Figures 5.11 through 5.15, where the following parameters are considered:

- *MinMax* normalization
- Without data augmentation
- *Binary Cross-Entropy* Loss

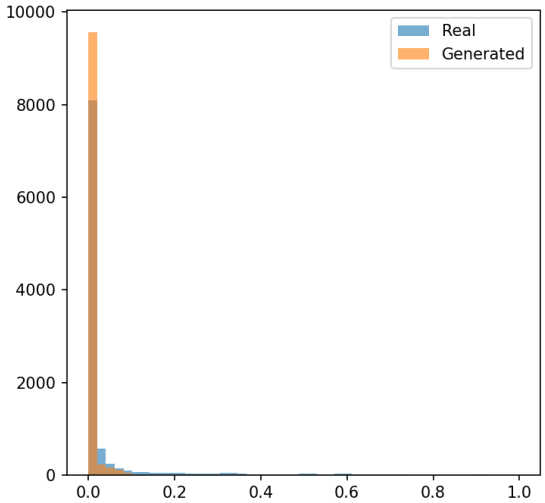

Figure 5.11: Time Delta Distribution with TCN generator

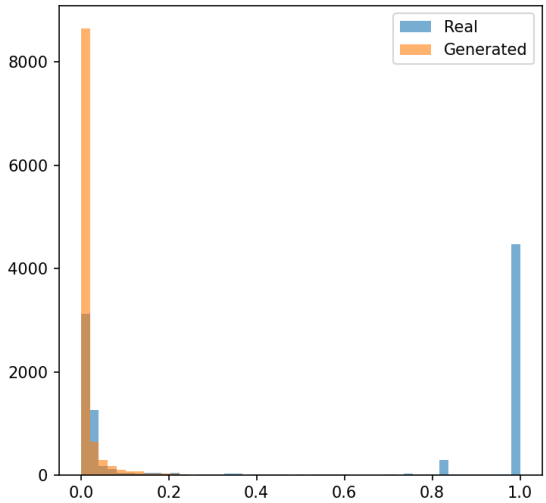

Figure 5.12: Length Distribution with TCN generator

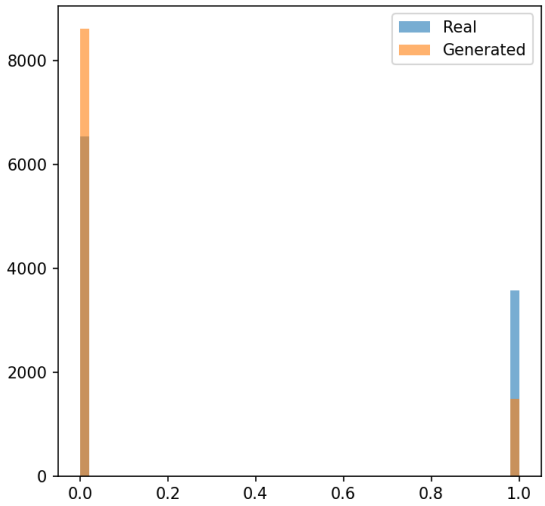

Figure 5.13: Flag PUSH Distribution with TCN generator

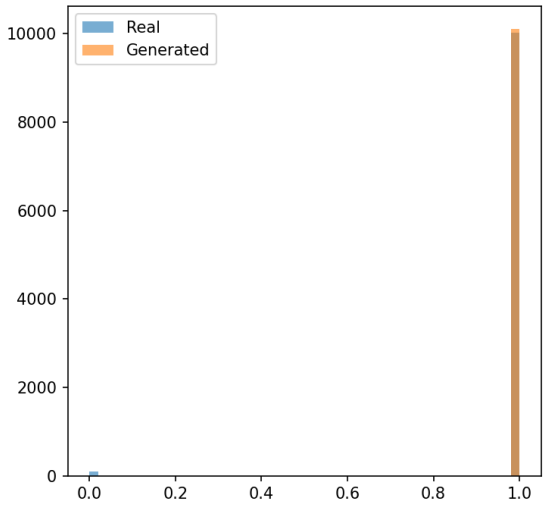

Figure 5.14: Flag ACK Distribution with TCN generator

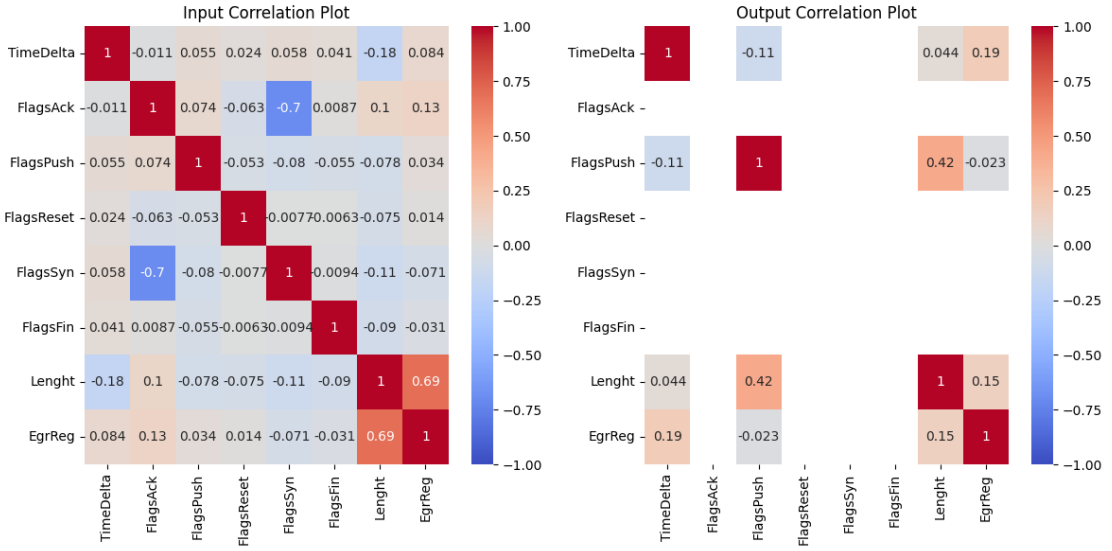

Figure 5.15: Variable Correlation Plot with TCN generator

The results of the next scenario are depicted below in Figures 5.16 through 5.20, where the following parameters were considered:

- *Box-Cox* transformation
- Without data augmentation
- *Binary Cross-Entropy* Loss

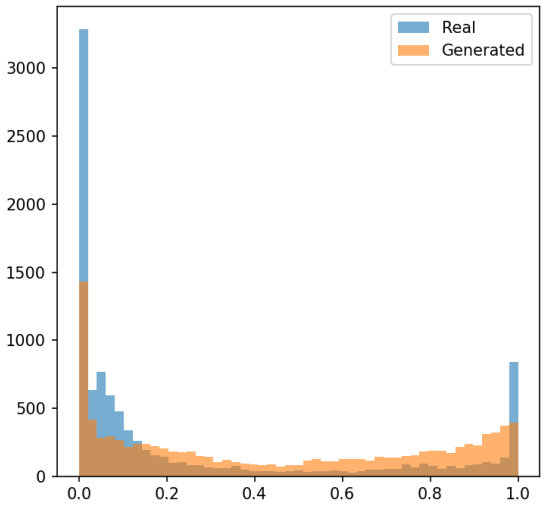

Figure 5.16: Time Delta Distribution with TCN generator

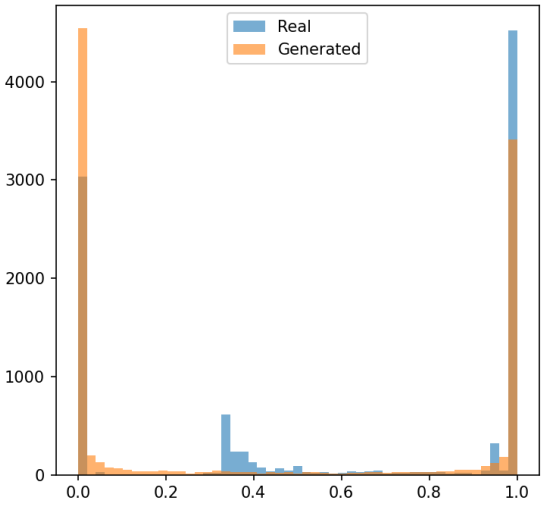

Figure 5.17: Length Distribution with TCN generator

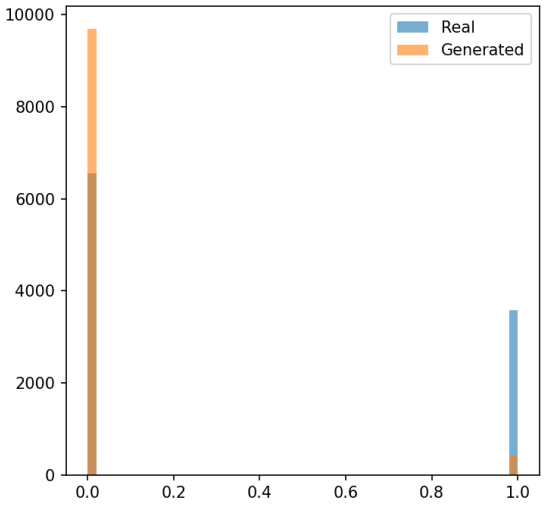

Figure 5.18: Flag PUSH Distribution with TCN generator

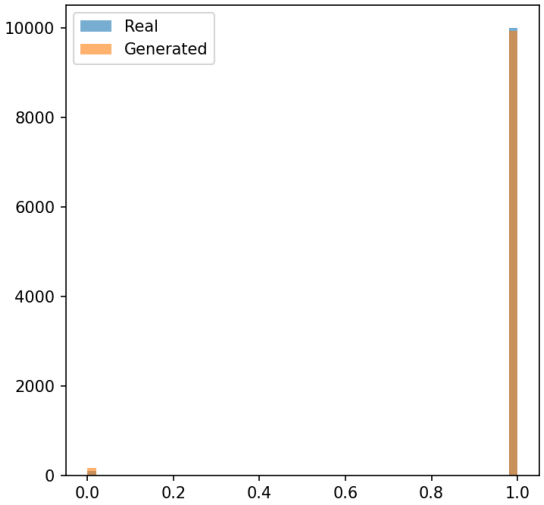

Figure 5.19: Flag ACK Distribution with TCN generator

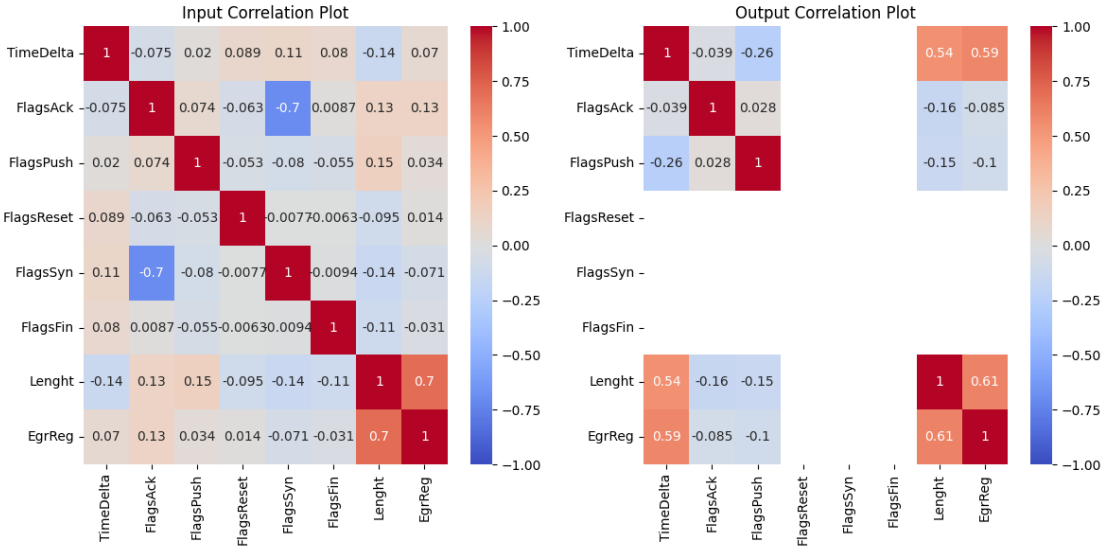

Figure 5.20: Variable Correlation Plot with TCN generator

5.2.3 Proposed Generator

The proposed generator's results are subsequently introduced, and its resulting evaluation metric values exposed in Table 5.3, followed by its distributions and autocorrelations plots for the same scenarious.

Table 5.3: Proposed network metrics

Augmented data	Input transformation	Loss Function	Evaluation Metrics		
			MAE	***MAPE***	***RSME***
Without Augmentation	*Min-Max*	*BCE Loss*	**0.213303**	6.415	0.450415
		Focal Loss	0.213518	6.422	**0.447168**
	Box-Cox	*BCE Loss*	0.22719	**4.724**	0.45060
		Focal Loss	0.22922	5.564	0.45233
With Augmentation	*Min-Max*	*BCE Loss*	0.64256	19.326	0.79157
		Focal Loss	0.64194	19.149	0.80013
	Box-Cox	*BCE Loss*	0.60973	18.337	0.72211
		Focal Loss	0.60994	18.433	0.72032

Regarding the first scenario, its results are represented in Figures 5.21 through 5.25, where the following parameters are considered:

- *MinMax* normalization
- Without data augmentation
- *Binary Cross-Entropy* Loss

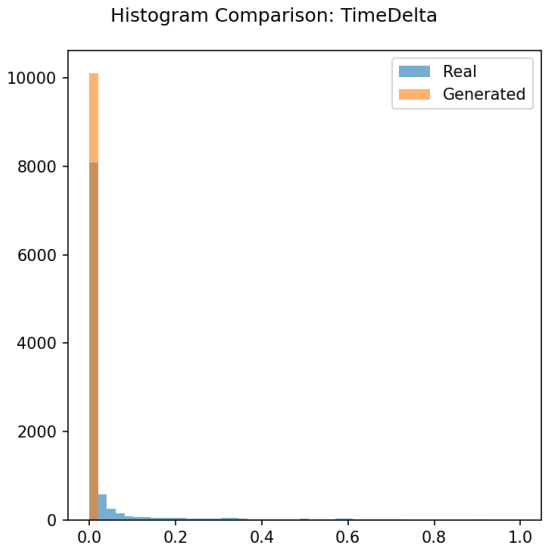

Figure 5.21: Time Delta Distribution with Proposed generator

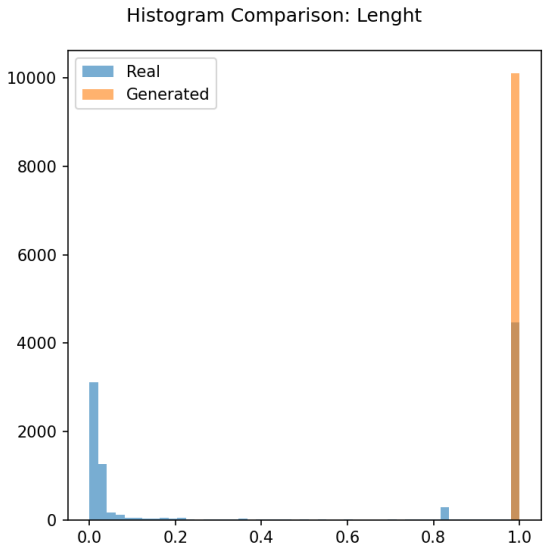

Figure 5.22: Length Distribution with Proposed generator

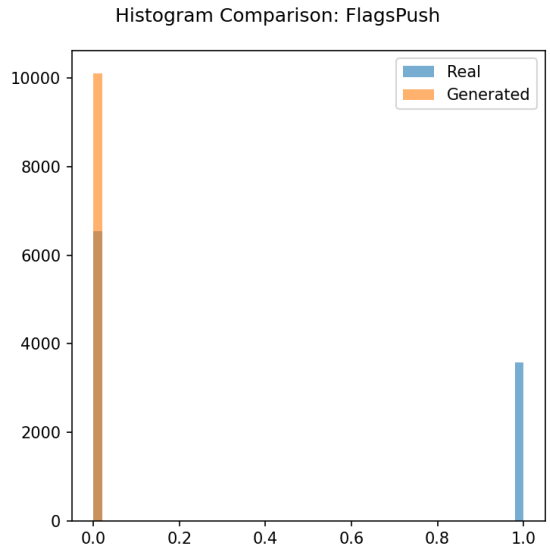

Figure 5.23: Flag PUSH Distribution with Proposed generator

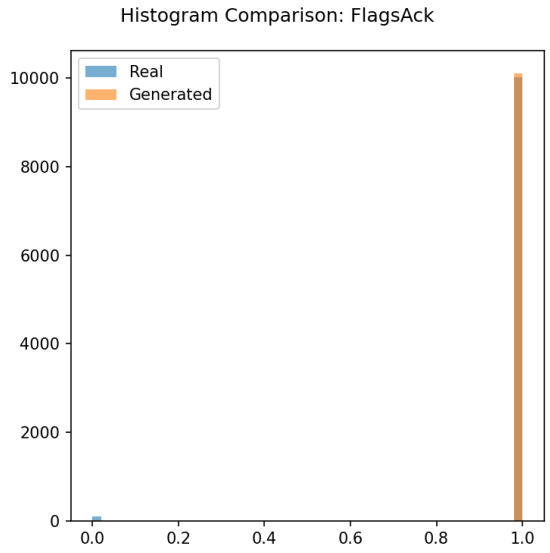

Figure 5.24: Flag ACK Distribution with Proposed generator

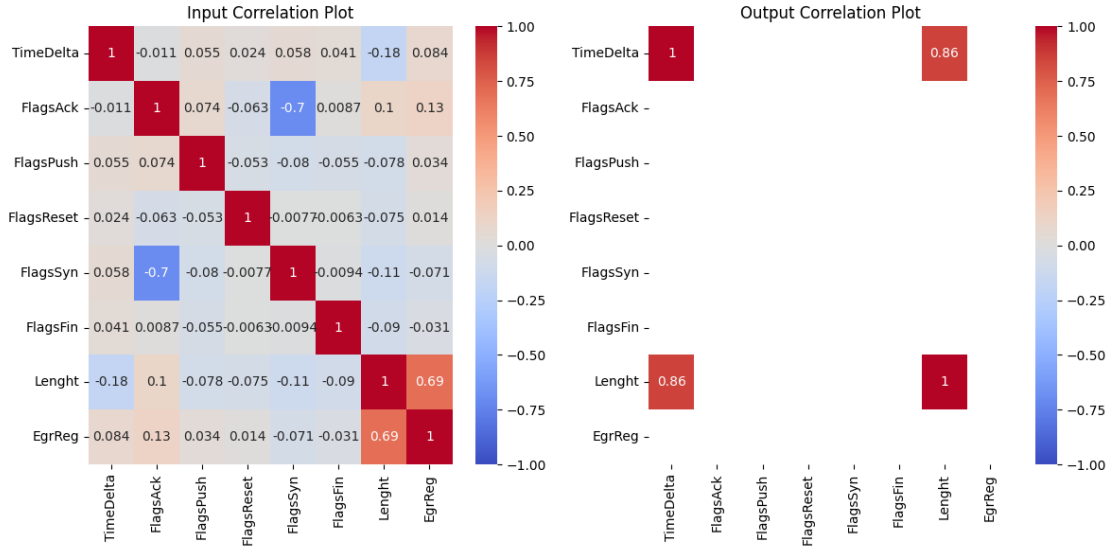

Figure 5.25: Variable Correlation Plot with Proposed generator

The results of the next scenario are depicted below in Figures 5.26 through 5.30, where the following parameters were considered:

- *Box-Cox* transformation
- Without data augmentation
- *Binary Cross-Entropy* Loss

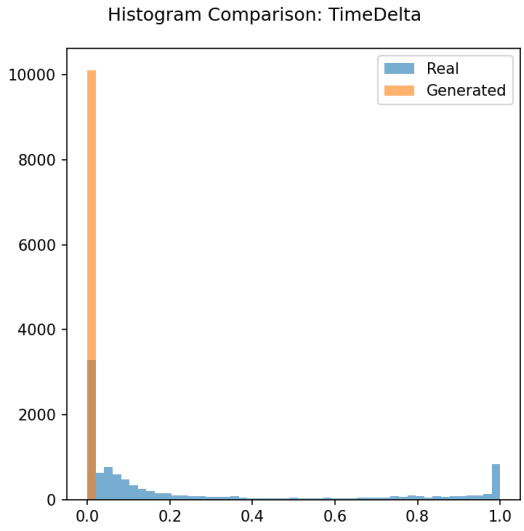

Figure 5.26: Time Delta Distribution with Proposed generator

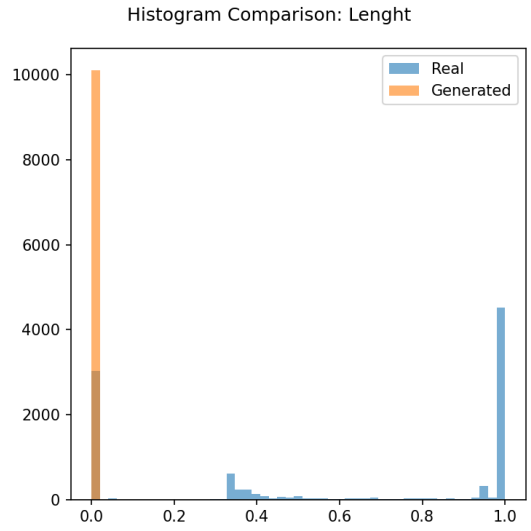

Figure 5.27: Length Distribution with Proposed generator

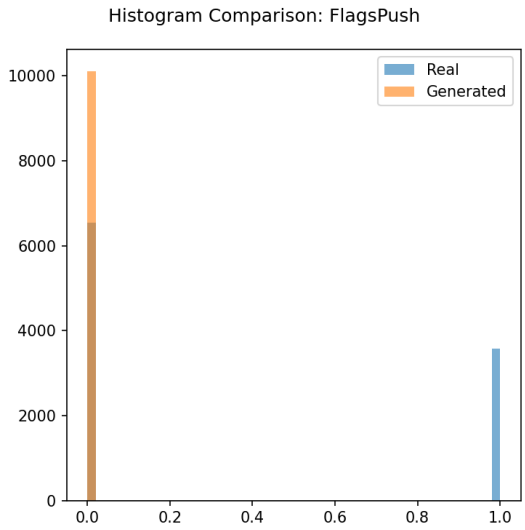

Figure 5.28: Flag PUSH Distribution with Proposed generator

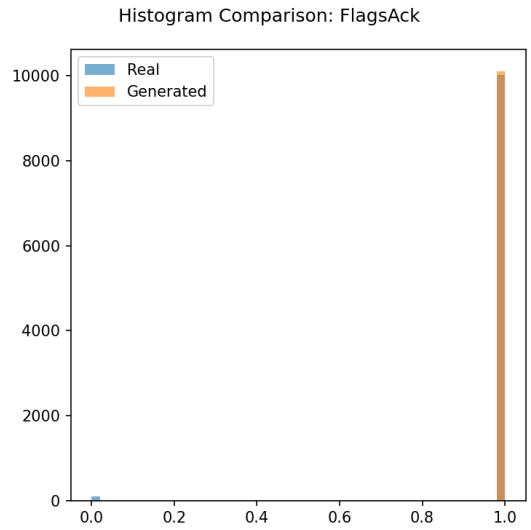

Figure 5.29: Flag ACK Distribution with Proposed generator

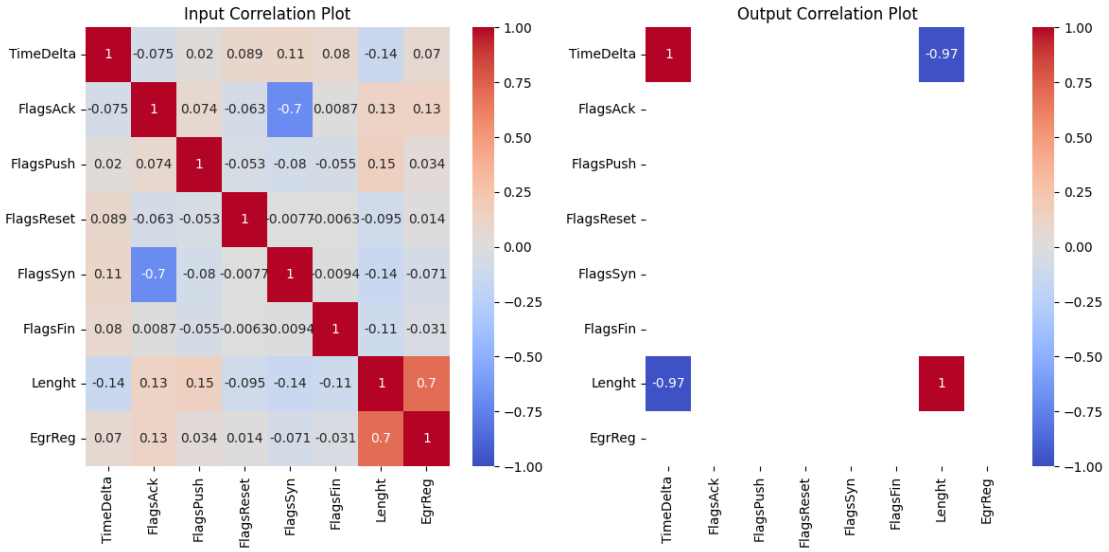

Figure 5.30: Variable Correlation Plot with Proposed generator

5.2.4 Simplified Proposed Generator

For the final generator, additional scenarios are presented since, of the proposed generators, this was the one who demonstrated the best results with the following Table 5.4 demonstrating it's obtained validation metric values.

Table 5.4: Simplified Proposed Network metrics

Augmented data	Input transformation	Loss Function	Evaluation Metrics		
			MAE	***MAPE***	***RSME***
Without Augmentation	*Min-Max*	*BCE Loss*	0.20927	4.896	0.44426
		Focal Loss	0.21761	5.002	0.45535
	Box-Cox	*BCE Loss*	0.23491	**3.638**	0.45830
		Focal Loss	0.22711	4.641	0.44437
With Augmentation	*Min-Max*	*BCE Loss*	0.66992	7.953	0.79204
		Focal Loss	0.20848	4.263	**0.42459**
	Box-Cox	*BCE Loss*	0.23214	4.018	0.44415
		Focal Loss	**0.18992**	4.747	0.43022

Regarding the results of the first scenario, these are depicted below in Figures 5.31 through 5.35, where the following parameters were considered:

- *MinMax* normalization
- Without data augmentation
- *Binary Cross-Entropy* Loss

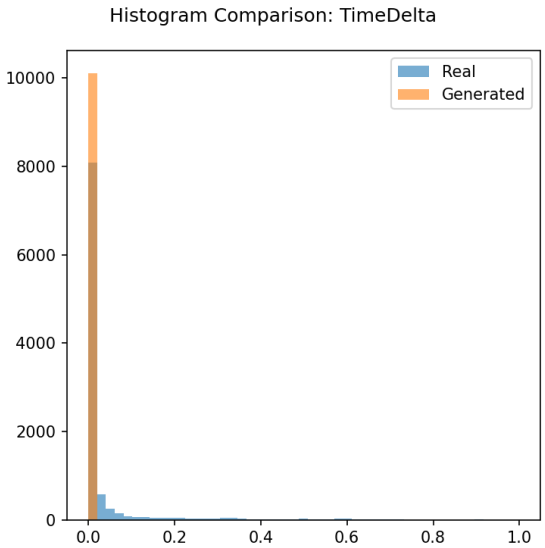

Figure 5.31: Time Delta Distribution with Simplified Proposed generator

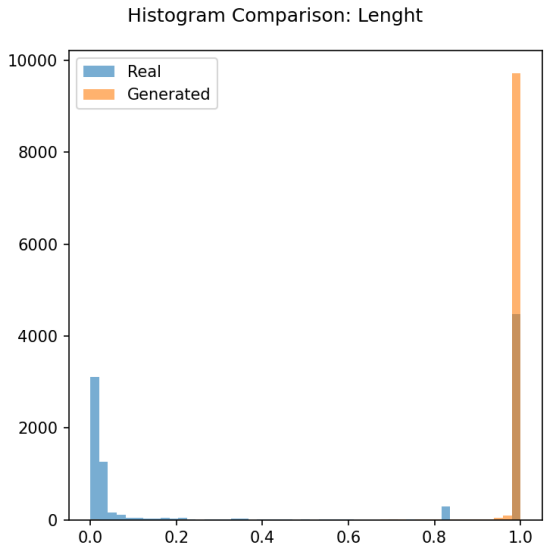

Figure 5.32: Length Distribution with Simplified Proposed generator

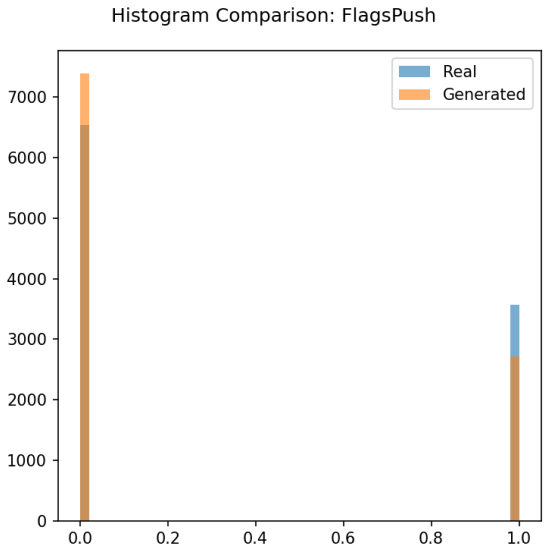

Figure 5.33: Flag PUSH Distribution with Simplified Proposed generator

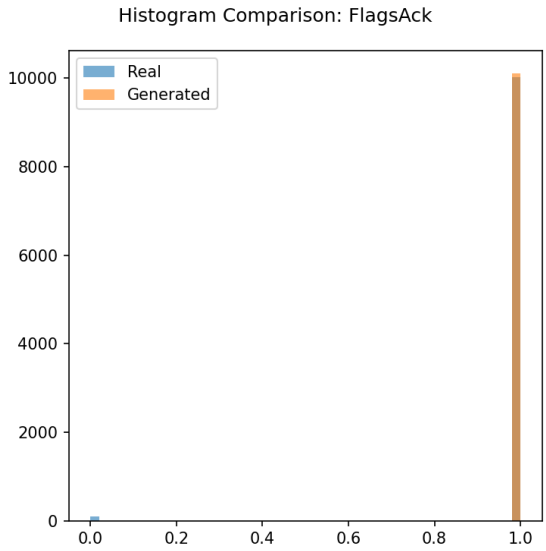

Figure 5.34: Flag ACK Distribution with Simplified Proposed generator

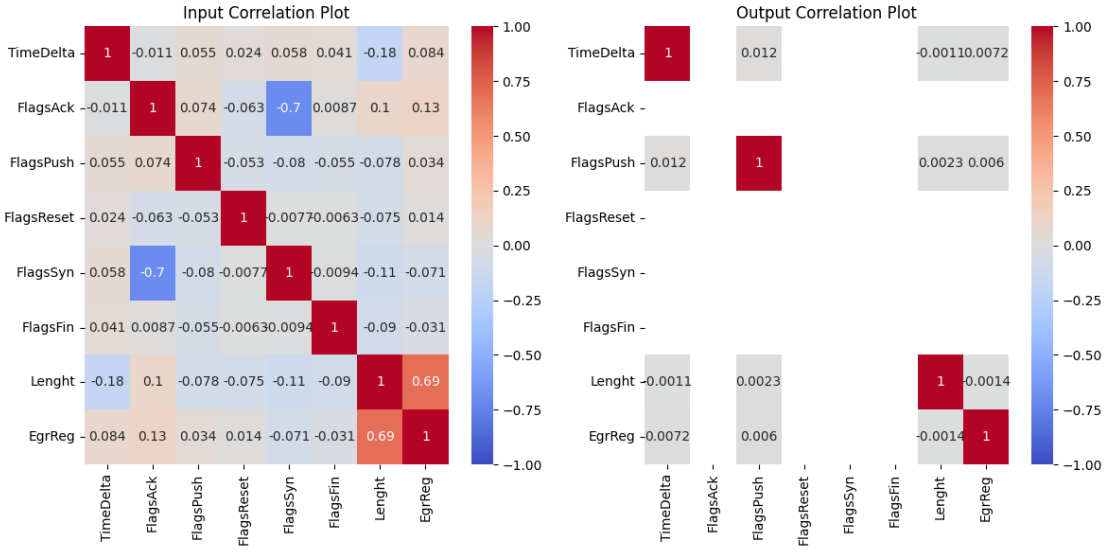

Figure 5.35: Variable Correlation Plot with Simplified Proposed generator

Following with the results of the second scenario, which are depicted below in Figures 5.36 through 5.40, where the following parameters were considered:

- *Box-Cox* transformation
- Without data augmentation
- *Binary Cross-Entropy* Loss

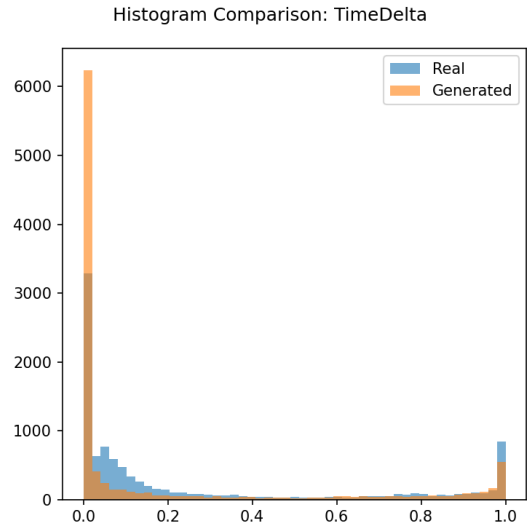

Figure 5.36: Time Delta Distribution with Simplified Proposed generator

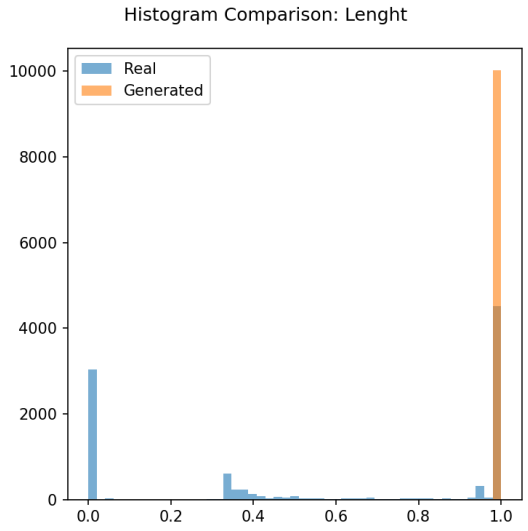

Figure 5.37: Length Distribution with Simplified Proposed generator

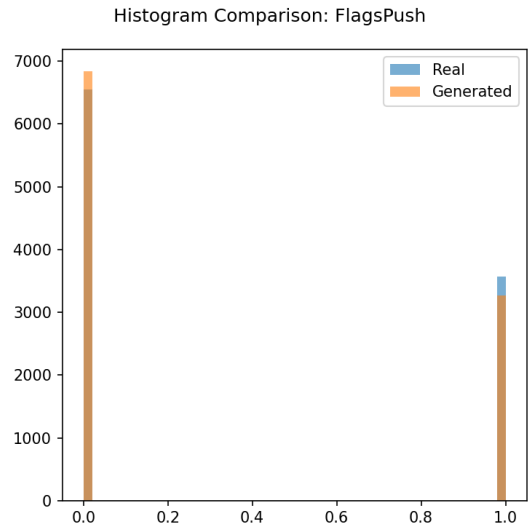

Figure 5.38: Flag PUSH Distribution with Simplified Proposed generator

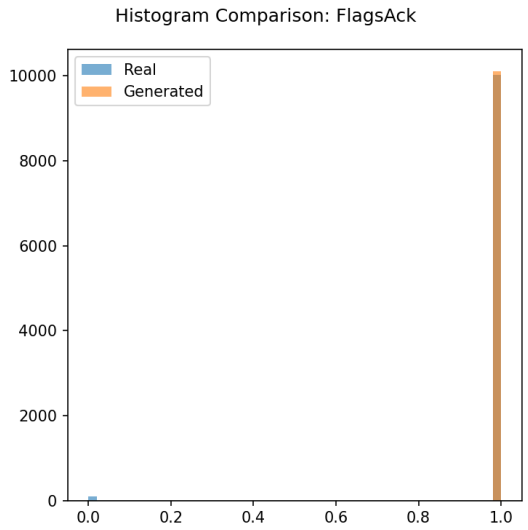

Figure 5.39: Flag ACK Distribution with Simplified Proposed generator

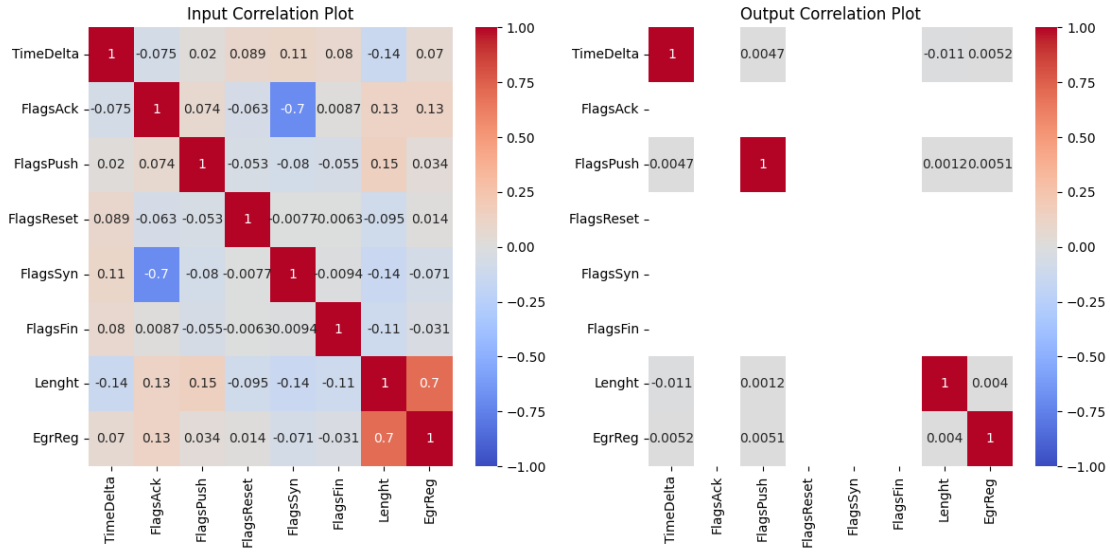

Figure 5.40: Variable Correlation Plot with Simplified Proposed generator

Additionally, other scenarios considered presented the results depicted below in Figures 5.41 through 5.45, where the following parameters were considered:

- *MinMax* normalization
- With data augmentation
- *Binary Cross-Entropy* Loss

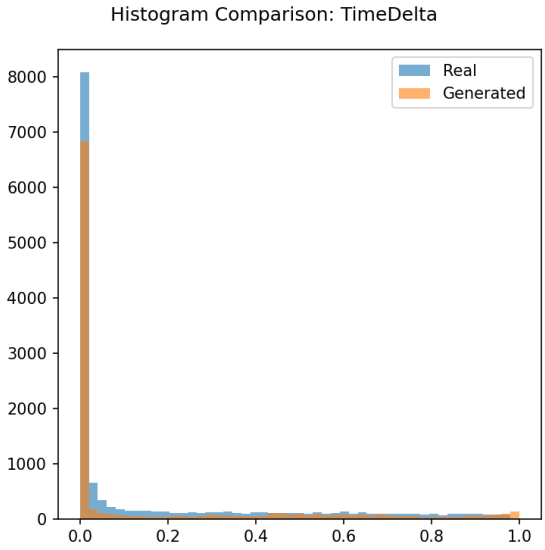

Figure 5.41: Time Delta Distribution with Simplified Proposed generator

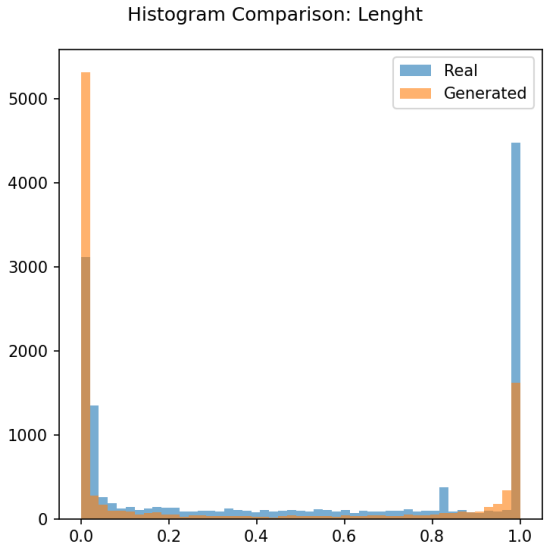

Figure 5.42: Length Distribution with Simplified Proposed generator

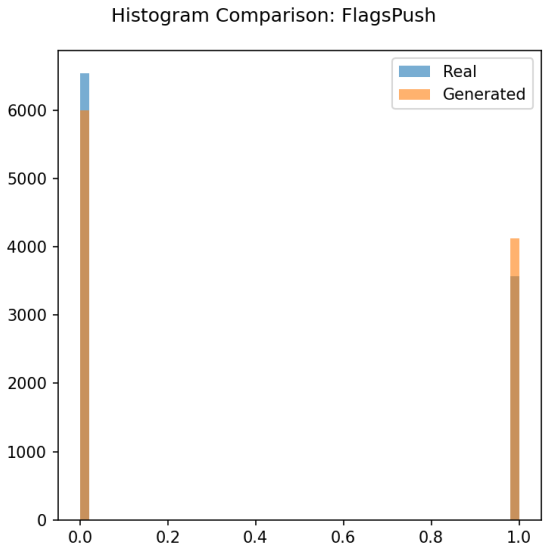

Figure 5.43: Flag PUSH Distribution with Simplified Proposed generator

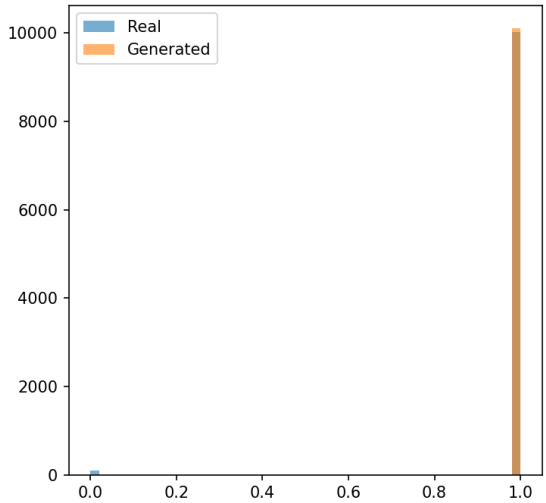

Figure 5.44: Flag ACK Distribution with Simplified Proposed generator

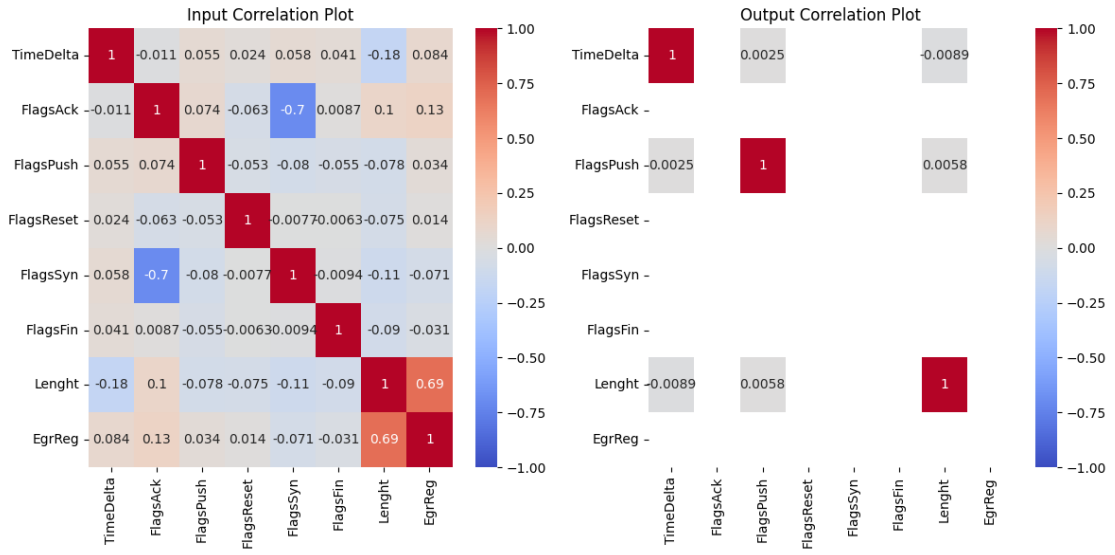

Figure 5.45: Variable Correlation Plot with Simplified Proposed generator

Finally, the results depicted below in Figures 5.46 through 5.50 represent the scenario where the following parameters were considered:

- *Box-Cox* transformation
- With data augmentation
- *Binary Cross-Entropy* Loss

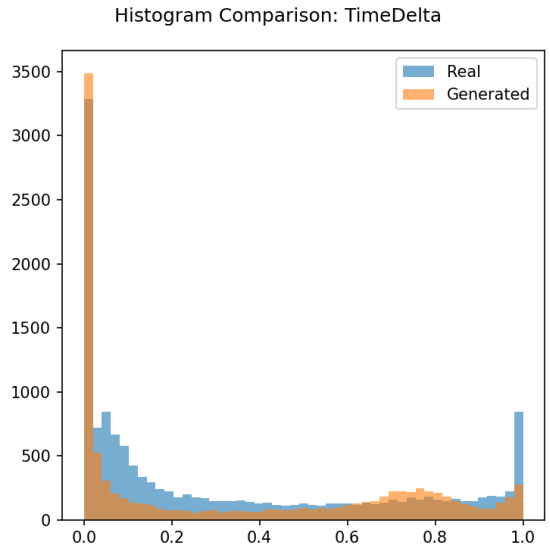

Figure 5.46: Time Delta Distribution with Simplified Proposed generator

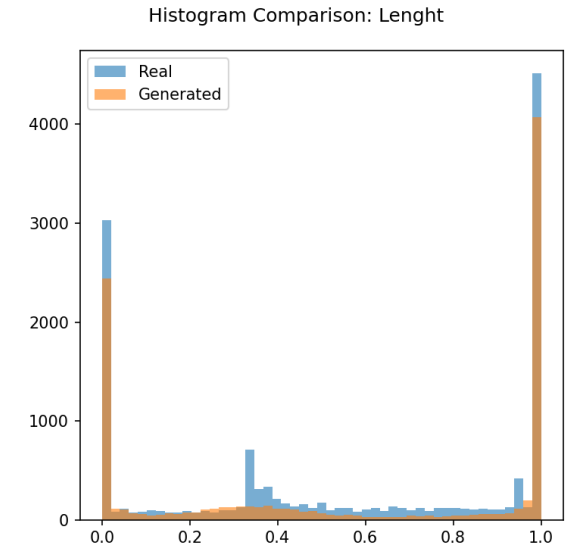

Figure 5.47: Length Distribution with Simplified Proposed generator

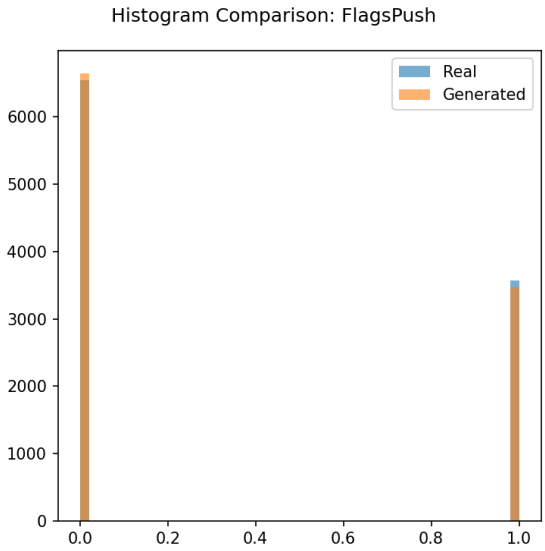

Figure 5.48: Flag PUSH Distribution with Simplified Proposed generator

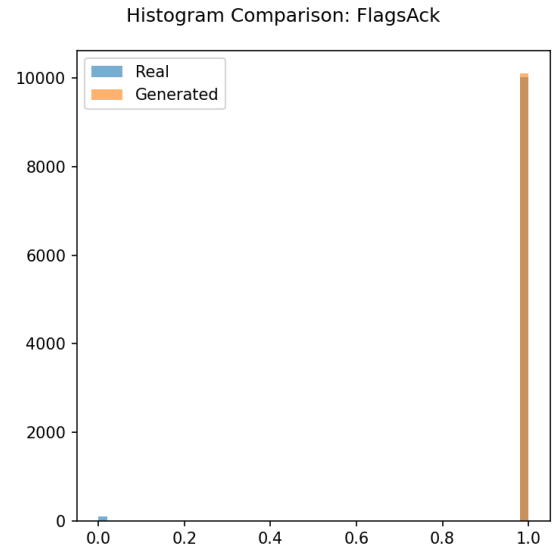

Figure 5.49: Flag ACK Distribution with Simplified Proposed generator

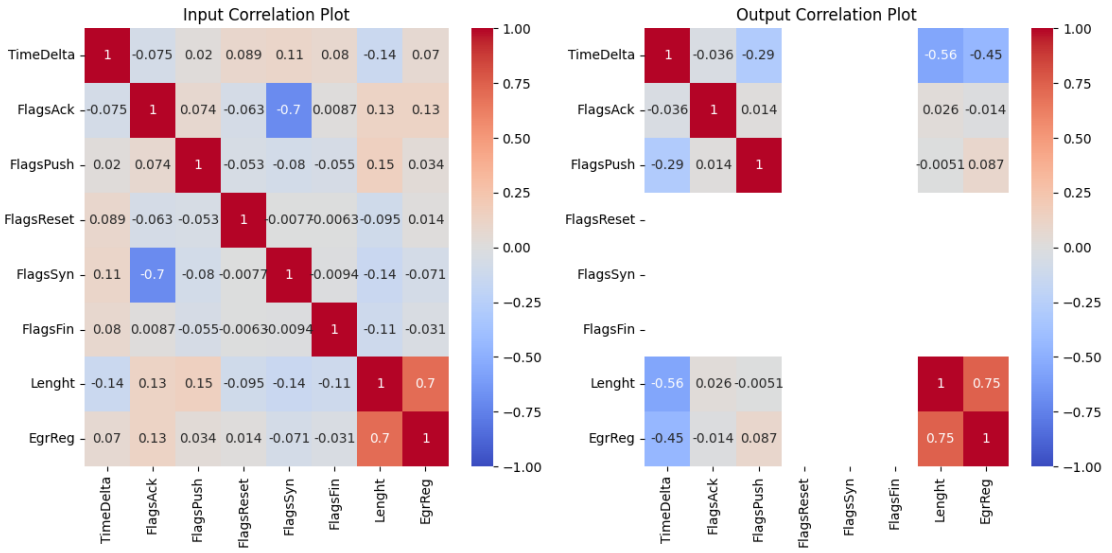

Figure 5.50: Variable Correlation Plot with Simplified Proposed generator

5.3 Effect of Hyperparameters

As aforementioned in Section 4.4, an extensive analysis of hyperparameter combinations was done to optimise the model's performance, and through this process the previously described set of parameters were achieved. During this optimization process one significant observation was that simpler models tended to outperform more complex ones in terms of results achieved. Models with fewer layers and lower complexity per layer demonstrated better convergence and avoided

overfitting issues. This finding highlighted the importance of avoiding unnecessary complexity when designing these models, as it can be observed by the achieved results.

Moreover, the learning rate discrepancy between the generator and discriminator had a significant impact on the training process. At first both networks were attributed the same learning rate value, thought it was observed that, in this case, the discriminator's loss had a tendency to approach zero within a few epochs. This is due to the fact that, when the discriminator's learning rate was set too high compared to the generator's, it tended to outlearn the generator rapidly and making it challenging for the generator to keep up and improve its performance. Adjusting the learning rates of the generator and discriminator to be more balanced resulted in a more stable and successful GAN training process, as the generator could better learn to produce realistic samples without being overwhelmed by the discriminator.

5.4 Discussion

Moving on to the result analysis, it becomes apparent from the preliminary observations that all models encountered considerable difficulty when predicting highly imbalanced binary variables. In such cases, where one class significantly dominates the other in the target variable, the models tended to lean towards predicting the most prevalent value. This observation was consistent across all model variations and hyperparameter combinations tested, despite the implementations put in place to handle these imbalanced variables.

- **LSTM generator** Starting with the LSTM generator, this model managed to predict well both discrete and continuous variables by generating accurate predictions that closely reproduced the real data distribution. Furthermore, managed to recognise underlying relationships between variables, though, in some variables, overestimating the correlations between them when also compared with real data correlation values.

  However, when the the Box-Cox was applied as a transformation to the input data, it further enabled the LSTM generator to identify previously unseen patterns and correlations in the raw data. Because of the model's improved capacity to capture non-linear correlations and complicated interconnections, the generator reached a greater degree of prediction when it came to correlation between variables even though, it did not manage to enhance it's performance regarding distribution comparison or even error metrics.

- **TCN generator** This generator model performed overall nicely in predicting continuous variables, similarly to the results of the LSTM generator. Because of its capacity to capture long-term relationships via convolutional processes, it was useful in dealing with continuous data, while allowing for quicker training times when compared to recurrent models such as LSTM due to the TCN's parallel processing architecture; yet, when it pertains to tracking discrete variables, the TCN experienced greater difficulties. One of the possible reasons behind this struggle with discrete variables could be that this model lacks the explicit memory mechanism found in recurrent models. While LSTM and other recurrent models use

hidden states to store and update information from past time steps, allowing them to retain information over longer sequences, the TCN, on the other hand, typically has a fixed-size receptive field, limiting its ability to capture long-term dependencies in sequences, especially in scenarios with sparse or complex temporal patterns making it harder to comprehend the sequential structure of the input in tasks requiring discrete data.

When the Box-Cox was applied to the input data, similarly to the the previous LSTM model, it further helped to identify previously unseen patterns and correlations though now leading to the overestimation of some correlation values within the data.

Despite these challenges, the TCN generator remains a powerful tool for data prediction due to its ability to handle continuous time series and numerical data effectively, coupled with its faster computation speed, making it a compelling choice for this scenario application.

- **Proposed generator** The proposed model, while showing promise in terms of low MAE, MAPE, and RMSE error values, encountered a critical limitation that hindered its performance in the generation task. The model consistently predicted only the most prevalent value for all tested scenarios, regardless of the input data, which severely impacted the model's ability to generate diverse and accurate predictions and ultimately rendering it ineffective for the intended task.

  One possible reason for this limitation is that the proposed model may be too complex for the task at hand and, as such, a simplified version of this model was proposed and also tested.

  Despite achieving low error metrics when augmentation was not used, it is crucial to recognize that these metrics do not necessarily indicate the model's success. Low error values might be a result of the model's tendency to predict the dominant value, effectively reducing the variance between the predicted and true values, however, this does not imply that the model is generating meaningful or accurate data predictions.

- **Simplified proposed generator** The simplified version of the proposed model exhibited promising results, particularly initially when dealing with discrete variables, surpassing its performance with continuous variables. This initially suggested that the model's architecture could be better suited for capturing the inherent patterns in discrete data sequences compared to continuous ones; however, the real breakthrough came when data augmentation was applied to the training process.

  With data augmentation, the simplified model showcased significantly improved results across all scenarios, regardless of the data transformations. This enhancement is indicative of the model's ability to learn from a more comprehensive and varied dataset, resulting in better predictions and increased robustness. Moreover, in this last case where data augmentation was employed, the simplified model achieved comparable results to the baseline models, namely the LSTM generator, regarding distribution comparison where this simplified model was able to generate across the entire spectrum as desired.

In the last results demonstrated, with both the Box-Cox transformation and augmentation, the model even displayed a similar correlation plot to the ones associated with the baseline models.

Furthermore, when the focal loss was used instead of the *binary cross-entropy*, certain working models demonstrated a tendency to achieve even lower error values. In particular, both the LSTM and the simplified generator algorithm exhibited noticeable improvements in predictive performance with the focal loss which can be attributed to the focal loss's ability to handle imbalanced datasets and emphasize hard-to-predict samples, making it a suitable choice for tasks with skewed or imbalanced class distributions

Nevertheless, it is essential to note that the same level of improvement was not observed with the TCN generator. While the focal loss proved beneficial for LSTM and the simplified generator, the TCN did not exhibit significant gains in predictive performance. This discrepancy suggests that the effectiveness of the focal loss might vary depending on the model architecture and the nature of the data being processed.

It's also worth nothing that, ideally, the training process would have been conducted using the entire dataset, as this has the potential to improve the models' predictions. However, practical constraints in computing resources prevented this due to the large size of the dataset and the computational demands of training complex models. Instead, only a limited portion of the data was utilized during the training phase and, although the dataset subset was carefully chosen to be representative of the whole distribution, it is important to recognise that this sample may have introduced some bias and may not completely reflect the intricacies contained in the entire dataset.

Despite the limits and obstacles encountered during training, the models produced promising results on select well-balanced variables and demonstrated the ability to generalize to some extent. However, when dealing with unbalanced variables, it is critical to interpret the findings beyond accuracy and deeply analyse distribution, as it was done throughout this work, due to the fact that a model can achieve high accuracy by merely predicting the majority class without actually representing the complete dataset. More research and improvements in addressing data imbalances will be necessary to enhance the model's capacity to manage such difficult scenarios.

Chapter 6

Conclusions and Future Work

This work addressed the challenges faced in generating realistic network traffic data, not only in network simulation, but also highlighted its potential usefulness in real-world applications. Beyond simulations, this generated data has broader implications for practical scenarios, such as network testing and evaluation, where accurate and diverse traffic data is essential for developing next-generation network solutions to which we aimed to contribute with the development of this work.

In the introduction, it was contextualized the importance of accurate network simulations for developing next-generation network solutions, while also highlighting the limitations of trace-based simulation methods and the need for more realistic digital twin environments. Also exploring the limitations of existing network traffic generation algorithms, which often produce simplistic and generic traffic scenarios. It emphasized the potential of GANs as a powerful tool for generating synthetic traffic data but acknowledged the need for further improvements to create realistic testing environments. Here, the main objectives were drawn: to develop a machine learning model capable of accurately generating network traffic data and, furthermore, to explore the differences in performance when various preprocessing techniques, input transformations, and data augmentation methods were applied. Moreover, the contributions of this research included the development of these preprocessing techniques to address imbalanced datasets and the identification of optimal data augmentation methods to enhance model performance with an additional study involving the impact of different hyperparameters to achieve best possible results.

The subsequent chapters delved into the relevance of synthetic network traffic data and its applications, the process of data acquisition and anonymization, and the development of the GAN-based model for generating realistic network traffic. The model incorporated time-dependent algorithms, LSTM, and TCN networks to capture temporal dynamics in network traffic accurately, alongside baseline models based on these algorithms.

In Chapter 5 the results of the GAN-based traffic generators were presented and discussed for all models with different metrics, shining a light on each model's advantages and disadvantages. While the simplified proposed model achieved some improvements compared to state-of-the-art solutions, it also highlighted the challenges of predicting highly imbalanced binary variables and

the need for further improvements in handling class imbalances. The study emphasized the importance of finding the right balance between model complexity and generalization capability, favoring simpler architectures for better performance.

In conclusion, this work has made strides towards enhancing network simulations by introducing a new GAN generator model capable of generating more realistic synthetic network traffic data. While challenges remain in handling class imbalances and replicating volatile network environments, the development of the model opens up possibilities for creating more accurate and representative digital twin environments. Furthermore, simplifications on the dataset were done which excluded some important parameters regarding network traffic and more complex generation structures were not implemented, leaving room for future research which should focus on addressing these challenges and further refining the model to achieve even more realistic network traffic generation for the advancement of next-generation networking solutions.

References

[1] Eduardo Nuno Almeida, Mohammed Rushad, Sumanth Reddy Kota, Akshat Nambiar, Hardik L. Harti, Chinmay Gupta, Danish Waseem, Gonçalo Santos, Helder Fontes, Rui Campos, and Mohit P. Tahiliani. Machine learning based propagation loss module for enabling digital twins of wireless networks in ns-3. In *Proceedings of the WNS3 2022*. ACM, jun 2022. URL: https://doi.org/10.1145%2F3532577.3532607, doi:10.1145/3532577.3532607.

[2] Adrien Becue, Eva Maia, Linda Feeken, Philipp Borchers, and Isabel Praça. A new concept of digital twin supporting optimization and resilience of factories of the future. *Applied Sciences*, 10:4482, 2020. doi:10.3390/app10134482.

[3] Huan X. Nguyen, Ramona Trestian, Duc To, and Mallik Tatipamula. Digital twin for 5g and beyond. *IEEE Communications Magazine*, 59(2):10–15, 2021. doi:10.1109/MCOM.001.2000343.

[4] Paul Almasan, Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Diego Perino, Diego López, Antonio Agustin Pastor Perales, Paul Harvey, Laurent Ciavaglia, Leon Wong, Vishnu Ram, Shihan Xiao, Xiang Shi, Xiangle Cheng, Albert Cabellos-Aparicio, and Pere Barlet-Ros. Digital twin network: Opportunities and challenges, 2022. URL: https://arxiv.org/abs/2201.01144, doi:10.48550/ARXIV.2201.01144.

[5] Miquel Ferriol-Galmés, José Suárez-Varela, Jordi Paillissé, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Building a digital twin for network optimization using graph neural networks. *Computer Networks*, 217:109329, 2022. URL: https://www.sciencedirect.com/science/article/pii/S1389128622003681, doi:https://doi.org/10.1016/j.comnet.2022.109329.

[6] Vitor Lamela, Helder Fontes, Tiago Oliveira, Jose Ruela, Manuel Ricardo, and Rui Campos. Repeatable and reproducible wireless networking experimentation through trace-based simulation. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 53–58, 2019. doi:10.1109/WiMOB.2019.8923120.

[7] Hongwei Yang, Yang Li, Kehan Yao, Tao Sun, and Cheng Zhou. A systematic network traffic emulation framework for digital twin network. In *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, pages 94–97, 2021. doi:10.1109/DTPI52967.2021.9540090.

[8] Krzysztof Rusek, Jose Suarez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, oct 2020. URL: https://doi.org/10.1109%2Fjsac.2020.3000405, doi:10.1109/jsac.2020.3000405.

[9] Mustafizur R. Shahid, Gregory Blanc, Houda Jmila, Zonghua Zhang, and Hervé Debar. Generative deep learning for internet of things network traffic generation. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 70–79, 2020. doi:10.1109/PRDC50213.2020.00018.

[10] Oluwamayowa Ade Adeleke, Nicholas Bastin, and Deniz Gurkan. Network traffic generation: A survey and methodology. *ACM Comput. Surv.*, 55(2), jan 2022. URL: https://doi.org/10.1145/3488375, doi:10.1145/3488375.

[11] Michael L. Wilson. URL: https://www.cs.wustl.edu/~jain/cse567-06/ftp/traffic_models2/#sec4.3.

[12] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009. doi:10.1109/TNET.2009.2020830.

[13] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Comput. Netw.*, 56(15):3531–3547, oct 2012. URL: https://doi.org/10.1016/j.comnet.2012.02.019, doi:10.1016/j.comnet.2012.02.019.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL: https://arxiv.org/abs/1406.2661, doi:10.48550/ARXIV.1406.2661.

[15] J. Brownlee. A gentle introduction to generative adversarial networks (gans), 19 Jul 2019. URL: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/#:~:text=GANs%20as%20a%20Two%20Player,as%20a%20supervised%20learning%20problem.

[16] Shuodi Hui, Huandong Wang, Zhenhua Wang, Xinghao Yang, Zhongjin Liu, Depeng Jin, and Yong Li. Knowledge enhanced gan for iot traffic generation. In *Proceedings of the ACM Web Conference 2022*, page 3336–3346, New York, NY, USA, 2022. Association for Computing Machinery. URL: https://doi.org/10.1145/3485447.3511976, doi:10.1145/3485447.3511976.

[17] Qiumei Cheng, Shiying Zhou, Yi Shen, Dezhang Kong, and Chunming Wu. Packet-level adversarial network traffic crafting using sequence generative adversarial networks, 2021. URL: https://arxiv.org/abs/2103.04794, doi:10.48550/ARXIV.2103.04794.

[18] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers &amp Security*, 82:156–172, may 2019. URL: https://doi.org/10.1016%2Fj.cose.2018.12.012, doi:10.1016/j.cose.2018.12.012.

[19] Adriel 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)*, pages 0728–0734, 2019. doi:10.1109/IEMCON.2019.8936224.

[20] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. Ip2vec: Learning similarities between ip addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 657–666, 2017. doi:10.1109/ICDMW.2017.93.

[21] Nápoles G. Van Houdt G., Mosquera C. A review on the long short-term memory model. *Artificial Intelligence Review*, page 5929–5955, May 2020. URL: https://doi.org/10.1007/s10462-020-09838-1, doi:10.1007/s10462-020-09838-1.

[22] 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*, IMC '20, page 464–483, New York, NY, USA, 2020. Association for Computing Machinery. URL: https://doi.org/10.1145/3419394.3423643, doi:10.1145/3419394.3423643.

[23] R. F. Bikmukhamedov and A. F. Nadeev. Multi-class network traffic generators and classifiers based on neural networks. In *2021 Systems of Signals Generating and Processing in the Field of on Board Communications*, pages 1–7, 2021. doi:10.1109/IEEECONF51389.2021.9416067.

[24] The Tcpdump Group. Tcpdump - man page, 2023. URL: https://www.tcpdump.org/manpages/tcpdump.1.html.

[25] Wireshark Foundation. Wireshark, 2023. URL: https://www.wireshark.org.

[26] Karlsruhe Institute of Technology Institute of Telematics. Pktanon - packet trace anonymization. URL: http://www.tm.kit.edu/software/pktanon/download/index.html.

[27] D.I. Management Services Pty Limited ABN. Sha-1 cryptography software, 2002. URL: https://www.cryptosys.net/sha1.html.

[28] Anaconda Inc. Anaconda | the world's most popular data science platform, 2023. URL: https://www.anaconda.com.

[29] Python Software Foundation. Welcome to python.org, 2023. URL: https://www.python.org.

[30] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *CoRR*, abs/2002.04803, 2020. URL: https://arxiv.org/abs/2002.04803.

[31] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evatualion of generic convolutional and recurrent networks for sequence modeling. 2018. arXiv:1803.01271.

[32] Zhenhao Gan, Chaoshun Li, Jianzhong Zhou, Geng Tang, and Geng Tang. Temporal convolutional networks interval prediction model for wind speed forecasting. *Electric Power Systems Research*, 2021. doi:10.1016/j.epsr.2020.106865.

[33] Lavanya Gupta. Focal loss — what, why, and how?, 2021. URL: https://medium.com/swlh/focal-loss-what-why-and-how-df6735f26616.