TimeGAN as a Simulator for Reinforcement Learning Training in Programmable Data Planes

Thiago C. Tavares*†, Leandro C. de Almeida †‡, Washington R. D. Silva†, Marco Chiesa§, and Fábio L. Verdi†
*IFSULDEMINAS Federal Institute of South of Minas Gerais, Poços de Caldas, MG, Brazil
†Department of Computer Science, Federal University of São Carlos, Sorocaba, SP, Brazil
‡Academic Unit of Informatics, Federal Institute of Paraíba, João Pessoa, PB, Brazil
§KTH Royal Institute of Technology, Stockholm, Sweden
*thiago.caproni@ifsuldeminas.edu.br, ‡leandro.almeida@ifpb.edu.br, †washingtonrds@estudante.ufscar.br,
§mchiesa@kth.se, †verdi@ufscar.br

***Abstract*—This study explores the application of Time Series GAN in a Programmable Data Plane (PDP) for enhancing Reinforcement Learning within the context of computer networks, particularly in video applications. We address various challenges, including dataset augmentation, balancing, and extended RL training times in real setups. By leveraging synthetic data generated by TimeGAN, we accelerate experimentation, enhance dataset diversity, and simplify RL model training, ultimately evaluating TimeGAN's performance against real setups in resource optimization for PDPs using an RL agent. This research contributes by directly comparing GAN usage and real setups, bridging a gap in computer network literature, and highlighting a 99% similarity in Quality of Service achieved by an RL model trained with synthetic data, affirming TimeGAN's potential as a valuable simulator without compromising RL training efficacy.**

***Index Terms*—Machine Learning, Generative Adversarial Networks, Autonomous Management**

I. Introduction

Generative AI, including ChatGPT, produces various data types like images, text, and media. It's gained popularity outside academia, emphasizing language understanding and generation, evolving from traditional models with enhanced learning capabilities due to increased parameters.

Within generative models, Generative Adversarial Networks (GANs) stand out. Initially introduced for image synthesis, GANs have gained prominence for capturing high-dimensional data distributions. They rely on two neural networks: a generator and a discriminator, engaged in a game interplay as outlined in [1]. The generator creates synthetic data to deceive the discriminator, acting as a judge to distinguish real from synthetic data. The goal is to reduce the discriminator's ability to differentiate between real and synthetic data.

Against this backdrop, GAN has seen an extension of its application into the domain of computer networks over recent years. Navidan et al. [2] have undertaken a classification of GAN based on their specific application objectives. To illustrate, GAN is capable of generating synthetic data to serve as inputs for distinct learning models. Alternatively, GANs may allocate one of their constituent neural networks—the generator or the discriminator—toward tasks of data prediction or classification, respectively

The utilization of GANs in computer networks depends on the specific application's needs. GANs serve as synthetic data generators, acting as simulators that replicate the original dataset's distribution, preserving privacy. They are also valuable for tasks like dataset augmentation and balancing, enhancing representational capacity. Consequently, the resulting models enable the sharing of complex real-setup dynamics while reducing complexities and maintaining data quality.

In computer networks, Machine Learning (ML) finds application in real setups, simulated environments, and through GANs emulating specific environment attributes. GANs excel at embodying system characteristics and facilitating resource orchestration in experimental frameworks for industry and research. The significance of generative AI in Computer Networks is underscored by the efforts of a group formed by ATIS Alliance in October 2023, as documented in [3].

This work has two primary objectives: firstly, it involves the generation of synthetic data using TimeGAN from a real dataset obtained in a PDP scenario, where input metrics stem from In-band Network Telemetry (INT) and DASH video to train models. Secondly, the study explores the use of synthetic data generated by TimeGAN as input for a Reinforcement Learning (RL) agent aimed at optimizing switch buffer sizes. The key advantage of employing synthetic data is the significantly reduced time required for offline RL training, typically taking just a few minutes, in contrast to time-intensive online training. Our work is inspired by PDP scenarios and draws from our prior research [4]. However, it can be applied to non-programmable data planes as discussed in Section V. While the utilization of an RL raises concerns about potential Service Level Agreement (SLA) impacts during online phase training, offline simulation-based training introduces simulation-to-real discrepancies affecting real-world fidelity, as noted in [5]. In this regard, TimeGAN offers a simpler approach to address these complexities compared to traditional simulation-based methods.

Our work also addresses concerns related to dataset augmentation, balancing, and the extended training time of RL in real setups. Utilizing synthetic data generated by TimeGAN accelerates experimentation, as it eliminates the need to run the entire application every time a change is made to the RL. TimeGAN also enables the creation of diverse datasets and enhances statistical variation for RL model training. We aim to evaluate TimeGAN's performance compared to a real setup

in the context of resource optimization within PDPs using an RL agent, with a specific focus on a video DASH application. This study contributes to filling a gap in the literature by directly comparing GAN usage and real setups, particularly in computer networks, where existing literature covers various GAN types and applications [2], [6], but lacks insights into the discrepancy between GANs and real setups, especially in the context of complex Time Series data inherent to computer networks.

Our main contributions can be summarized as follows.

We assess TimeGAN's capacity as a data generator for training RL models with real datasets. Our systematic methodology gauges TimeGAN's data generation quality, aiming to demonstrate its viability as a simulator to reduce simulation-to-real discrepancies.

We designed a metric for the optimization of trained GAN models to minimize differences from real data. The quality of TimeGAN-trained distributions for each dataset feature depends on hyperparameter settings. To address this, we introduce a metric to systematically identify optimal models for each feature. This metric helps generate a synthetic dataset that closely resembles the original data's characteristics, yielding a more representative approximation.

We conduct a comparative analysis of RL training in a real setup versus using TimeGAN as a simulator. Our primary objective is to investigate performance disparities between the two approaches, specifically assessing differences in the training efficacy of the RL algorithm when utilizing real data versus synthetic data generated by TimeGAN. Our experimental results highlight a remarkable 99% similarity in the Quality of Service (QoS) achieved by an RL model trained with synthetic data as opposed to one trained in a real setup.

The paper is structured as follows: Section II covers GAN fundamentals and the related works. Section III explains our approach and methodology. Section IV discusses experiment results. Section V shares lessons learned, and Section VI presents the paper's conclusions.

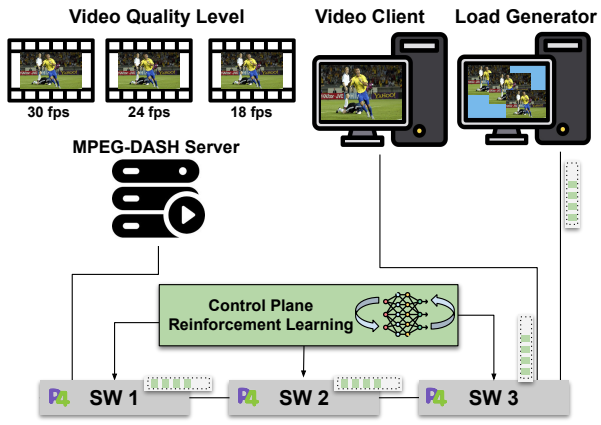


Fig. 1. Real setup used for the evaluation composed by P4 BMv2 switches, a DASH server and video clients.

II. Related Work

Machine learning encompasses three main paradigms: supervised, unsupervised, and reinforcement learning. GAN, for instance, falls under unsupervised learning, with the generator replicating data distributions and the discriminator playing a role akin to supervised learning by comparing generated and real data. GANs have diverse applications in Computer Networks, including data generation, system optimization, and data classification, as documented in existing literature [2], [7]. The choice of GAN variant depends on specific objectives, and recent adaptations extend GANs to time-series network data. There are a lot of GAN models, such as Vanilla GAN, BIGAN, CGAN [8], InfoGAN [9], CycleGAN [10], EBGAN [11], and LSGAN [12]. However, there is an inclination towards encompassing time series data [7]. Specific GANs, like CTGAN [13], DoppelGANger [14], and TimeGAN [15], are tailored for learning the intricacies of time-series data [16].

GANs have versatile utility in data generation, addressing issues like data imbalance and privacy preservation. They can rectify skewed datasets, impute missing values, and obfuscate sensitive information, fostering secure data exchange. These data generators are important in creating new datasets, often combined with complementary machine learning methods, such as RL models, as demonstrated in [17]. RL is a machine learning paradigm where an autonomous agent optimizes decision-making in an unknown environment. It involves the agent interacting with the environment, taking actions, and receiving rewards or penalties based on the outcomes. Rewards follow positive outcomes, and penalties follow harmful ones, driving the agent to optimize state-action pairs through iterative trial-and-error exploration.

Zou et al. [18] and Gupta et al. [19] discuss RL's applications in computer networks, enhancing resource optimization and network efficiency. However, the integration of GAN and RL remains underexplored, lacking a comprehensive exploration of their combined benefits. RL excels at decision optimization, while GAN simulates complex data distributions, aiding in training RL agents to optimize network configurations in an offline way. However, when using RL in real setups versus simulations, an important challenge arises: sim-to-real discrepancy, observed across domains like communication, computer vision, natural language processing, robotics, and autonomous driving [20]. This discrepancy reflects the gap between simulated and real environments, making it difficult to bridge this divide in the context of computer network research, where simulating real network configurations within complex and layered network structures remains a significant challenge.

In [21], the authors discussed the Atlas system, an automatic online service configuration for network slicing aimed at reducing Sim-to-Real discrepancy through a three-stage methodology. Complementarily, in our study, we explore the potential of reducing Sim-to-Real discrepancy using TimeGAN instead of Bayesian optimization by training a Reinforcement Learning model and comparing its performance with real and synthetic data. On the other hand, [17] introduces a strategy combining deep reinforcement learning and distributional modeling to optimize resource allocation in network slices, improving efficiency and service quality by addressing

resource management complexities. While the authors utilize a GAN to approximate state-action value distributions in an RL model, our primary focus is on generating simulated environmental data to facilitate RL training.

In [22], a CGAN-based approach addresses network slicing in heterogeneous vehicular networks by partitioning the network into virtual networks to meet diverse service requirements. The authors propose using GANs to generate customized network slices tailored to specific vehicular communication needs, to optimize resource allocation, improve communication quality, and facilitate efficient coexistence of vehicular applications. In this case, the CGAN model is trained using simulated data due to the absence of real-world datasets.

In [23], the integration of GAN and RL techniques is explored for estimating channel coefficients in wireless communication. The approach involves generating synthetic data with GAN and using it to train RL algorithms, resulting in improved accuracy for channel coefficient estimation. This highlights the potential of machine learning in wireless communication. It's worth noting that the study relies on synthetic data generated with Gaussian distributions for GAN training, without utilizing a real experimental setup or real data.

Table I compares various works in the literature focused on resource optimization in computer networks. Four key metrics are analyzed: the use of (1) GANs and (2) RL for resource optimization, (3) utilization of real data or real setup, and (4) integration of simulation methods in the proposed solutions.

TABLE I
WORK COMPARISON

Metric	Ours	[21]	[22]	[23]	[17]
Year	2023	2022	2021	2021	2019
GAN	✓	X	X	✓	✓
RL	✓	X	✓	✓	✓
Real	✓	✓	X	X	X
Simulated	✓	✓	✓	✓	✓

We bridge a literature gap by using TimeGAN for simulating real setups. Our methodology tackles network dataset complexities and dynamics, emphasizing statistical training and model selection. We evaluate the practicality of synthetic data by deploying an RL agent in a real setup after offline training with GAN-generated data, as detailed in Section III.

III. TIMEGAN AS A SIMULATOR

In this section, we delineate our proposal, structured into distinct subsections comprising the problem definition, the methodology employed to address the problem, and the tasks associated with the construction of TimeGAN. This model serves a dual purpose as both a generator of synthetic data and a simulator, faithfully replicating the behavior of a real setup.

A. Problem definition

As mentioned in Section II, there is a research gap in the current literature regarding the comparative analysis between GANs trained on original datasets and their application in RL model training. This gap arises from the complexities of executing certain ML models in operational scenarios. Operators typically adopt a monitoring-centric approach, monitoring application performance, tracing data, and selecting datasets. However, publicly accessible datasets are limited, and even when real data acquisition is possible, replicating experiments for comparisons under different conditions, such as meeting distinct user requirements and SLAs, presents challenges.

As depicted in Figure 1, our study operates within a real video application environment. This environment comprises an MPEG DASH Server for network load generation, a network with three programmable switches that provide INT telemetry, and a Video Client for collecting video-related metrics. The integration of INT, in conjunction with application metrics, enables comprehensive monitoring of application behavior on the network. Overseeing this setup is an RL agent, tasked with enhancing user experience by optimizing switch queue sizes, primarily aimed at resource allocation and network efficiency improvement. Adjusting queue sizes is an important technique for accommodating network traffic and influencing the QoS of applications. For this reason, we applied this action to validate our RL agent.

So, the RL model serves as a data plane optimizer, managing queue sizes in three switches to enhance user experience. Obtaining cooperation from network operators for such experiments is often challenging. To address this, a mechanism is devised to train the RL agent without a real setup, using a GAN trained on real data as a simulator. Our approach combines real setup training for the RL agent with dataset storage for the GAN model. In contrast to online, offline training of the RL model is conducted using synthetic data generated by TimeGAN. This allows us to compare the RL agent's generalization capabilities in both scenarios: real setup and its simulated counterpart using the GAN generator.

However, a pertinent question arises: given the availability of datasets sourced from operators, the utilization of GAN to train Machine Learning models, including Reinforcement Learning methodologies, is necessary? Why not exclusively rely upon the original dataset for the training of such models?

Collecting a dataset for GAN model training typically involves obtaining data from operators or real setups, which may suffer from issues like imbalance, inadequacy, missing or erroneous values, and static characteristics. This limits the feasibility of repeated experiments and statistical validation, as seen in [24]. In contrast, a trained GAN can generate balanced data without inconsistencies, enabling experimentation with diverse data volumes and distributions for various scenarios. Additionally, GAN accelerates RL model training compared to real scenarios, enhancing the agent's adaptability to broader conditions in RL applications.

Consequently, this paper delineates a methodology wherein the efficacy of an RL model is evaluated through its deployment in conjunction with data generated by a TimeGAN model. The TimeGAN is trained to use telemetry data obtained from video applications situated within the PDPs. The following exploration thereby offers insights into the viability and performance of RL when trained with synthetic data generated by TimeGAN.

B. Methodology

Broadly, the methodology adopted in this proposal is encapsulated within Fig. 2, and its implementation can be defined through a delineation into four distinct steps as follows.

1) Creation of a Real Setup and Data Collection: In this preliminary phase, the establishment of a tangible real setup is undertaken, as presented in detail in Section III-A.

2) Dataset Recording and TimeGAN Model Training: Within this stage, an ensemble of pertinent features is stored. Among these features are switch telemetry metrics, notably encompassing switch queue sizes and packet arrival times. Simultaneously, the capture of video metrics, such as frames per second (FPS) and resolution, is also undertaken. The ascertained dataset comprises two distinct scenarios, characterized by dissimilar queue sizes—32 and 64 packets—affording coverage of two operational settings.

3) Reinforcement Learning Training Utilizing Synthetic Data: After training the TimeGAN model, the next step involves training the RL algorithm with synthetic data. Two synthetic datasets were generated, corresponding to two queue sizes: 32 and 64 packets. The RL algorithm's input depends on the agent's actions, which can transition the queue size between these two values. When the agent takes an action to change the queue size to 32 packets, the RL model receives synthetic data from that dataset (scenario), and vice versa for a queue size change to 64 packets.

4) Performance Evaluation of RL Models: This phase conducts a comprehensive evaluation of two RL training methods: one in a real setup and the other using synthetic data from TimeGAN. The primary objective is not to optimize RL model parameters but to assess the similarity of outcomes in both scenarios. The evaluation aims to determine if TimeGAN can effectively replicate RL training results, validating its potential as a surrogate simulator without compromising RL training efficacy.

The continuous line in Fig. 2 illustrates an online component wherein the RL model undergoes real-time training (depicted in step 1). Conversely, the GAN model training and data generation undergo in an offline manner, as depicted in steps 2 and 3. Concluding the sequence, step 4 embodies the comparative evaluation conducted between the RL model, trained dynamically in the real setup, and synthetic data engendered by the TimeGAN algorithm.

C. Dataset characterization

Our dataset, centered on the Video Application within the PDP context, comprises two distinct categories of metrics: video metrics and network metrics, each residing in separate datasets. The video metrics include frames per second (FPS), bitrate, and buffer size. In contrast, the network metrics, obtained from INT, consist of queue depth at packet queuing (`enq_qdepth1`), packet queuing duration in microseconds (`timedelta1`), and queue depth at packet dequeuing (`deq_qdepth1`). The number after the metrics determines the switches where they were collected on the network.

Our network configuration employed BMv2 software switches with P4 code. Specific INT probes are sent from the DASH server to the Video Client. This approach eliminates the need to modify data packets for telemetry metadata. Telemetry metadata (32 bytes) was collected at each network node. Two experiments were conducted to gather data from our real setup, as discussed in Section III-A. The first experiment used switches configured with 32-packet buffer size, while the second utilized a 64-packet buffer size. Consequently, we obtained two datasets, each representing the real setup under differing buffer size conditions. These datasets were merged based on timestamps, with higher `timedelta1` samples filtered out to capture network behavior under high-load conditions.

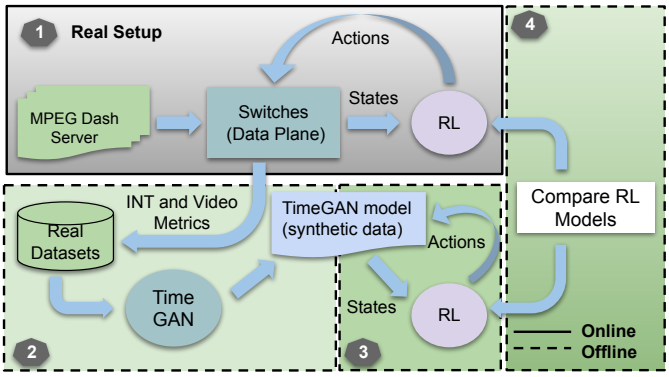


Fig. 2. Proposed Methodology.

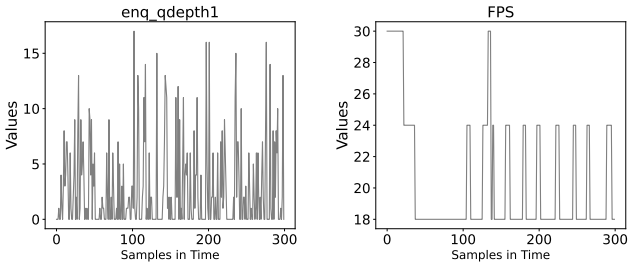


Fig. 3. Real data plots for enq_qdepht1 and FPS features.

An essential characteristic of the telemetry values within our PDP application is the non-stationary nature of all our features, as visually demonstrated in Fig. 3. This data distribution poses significant challenges in the endeavor to compare real data with synthetic data generated by GANs. To address this complexity, we introduce a metric in Section III-E. This metric serves as a tool in our model selection process, aiding us in the identification of the most suitable model capable of providing the closest feature representation to the real dataset.

D. TimeGAN Training

TimeGAN is a generative model crafted to generate synthetic time series data, intending to replicate the statistical characteristics inherent to real-world time series datasets. The training of a TimeGAN model encompasses a series of steps.

Firstly, we initiated the process with data preprocessing, addressing concerns such as missing data imputation, outlier removal, and data normalization. The configuration of hyperparameter values was also crucial since it plays a pivotal

role in TimeGAN's training regimen. The determination of sequence sizes or windows was a significant consideration. These sizes are instrumental in the training, as they subdivide the dataset into multiple 'snapshots,' each designed to capture the temporal behaviors within the time series.

Furthermore, other hyperparameters are variated, including sequence length, the number of hidden dimensions, and batch size. Hyperparameter tuning is a recognized challenge in machine learning training, with various strategies available, such as Grid Search. In our case, we adopted an empirical approach, wherein we iteratively adjusted hyperparameter values based on prior training experiences and our insights into the parameters that exerted the most significant influence on the training process. Detailed hyperparameter values are outlined in Table II. We executed these experiments using the YData Synthetic API[1] to facilitate our research endeavors.

Following the completion of the training processes, it came to our attention that the loss values across all 54 training iterations (27 for each buffer size setting for switches) exhibited a degree of similarity. Consequently, we recognized the need to develop a model selection methodology, the details of which are expounded upon in Section III-E.

TABLE II
TRAINING HYPERPARAMETERS

Hyperparameters	Values		
Sequence size	50	100	150
Hidden Dimension	20	40	60
Batch Size	100	128	156

E. Models Selection

Assessing synthetic data generated by GANs is a challenge. The lack of consensus on evaluation methods is evident in studies like [25] and [26]. This challenge is even more pronounced in the context of time series data, where recent publications on the topic are scarce compared to more conventional GAN models. Adding to the complexity, the dataset we've collected is non-stationary, meaning it exhibits trends, seasonality, and other systematic changes that cause statistical properties like mean, variance, and autocovariance to vary with time. This non-stationarity makes it difficult to apply traditional tests like Kullback–Leibler (KL) divergence. For instance, some trained GAN models may produce synthetic data with constant values for certain features, whereas real data often has high variability. If we were to use traditional tests, the KL test results might favor distributions with constant values, which is not what we want in practice.

In our experimentation, we noted through empirical observations that certain GAN models, which we have trained have superior synthetic data for distinct features compared to others. Consequently, we proposed a mechanism aimed at selecting the most suitable model by analyzing each specific feature.

To address this objective, we introduce a metric constructed through straightforward statistical measures, encompassing quartiles and medians. The central concept revolves around contrasting the distribution disparities exhibited by real and generated synthetic data. Consequently, a more favorable model for a given characteristic is indicated by a diminished dissimilarity in data distribution between these two sources. The calculation of this metric is depicted by Equation 1. To initiate this calculation, it is imperative to determine the discrepancy magnitude between the third and first quartiles for both datasets: real (X) and synthetic (y). This calculation furnishes the size of data dispersion, facilitating a comparison.

Subsequently, another calculation involves assessing the variation between the sizes of these data dispersion, thereby gauging the difference between real and synthetic data. However, determining the disparity in dispersion magnitude alone might be insufficient, as the dispersion might have similar sizes while maintaining a positional offset. To address this, we calculate the difference in medians between the real and synthetic datasets and incorporate this disparity alongside the variance in dispersion sizes. Thus, a summation of these differences between quartiles and median for each feature is achieved and stored in metric (M) for all of the models. So, a superior model is characterized by the minimal value of M, indicative of the least dispersion in data distribution between real and synthetic data sources.

$$\mathbf{M} = \sum_{n=1}^{n\_feats} |[Q_3(X_n) - Q_1(X_n)] - [Q_3(y_n) - Q_1(y_n)]| + |[med(X_n) - med(y_n)]| \tag{1}$$

The algorithm presented in Algorithm 1 elucidates the procedure of selecting the optimal model for individual features. The algorithm takes as its inputs arrays of models about the two distinct scenarios exposed in Section III-D, specifically those delineated by switch queue sizes of 32 and 64 packets.

The algorithm's initial operation involves the computation of the aforementioned metrics for each feature and all of the trained models, as detailed earlier. After this metric computation, an iteration wherein the algorithm determines the summation of metrics for each model is achieved. So, the minimum value of the sum of all features determines the best model ***considering the models that were trained in our study***.

The algorithm culminates by providing two arrays, each comprising the features of the best model. These arrays distinctly encapsulate the finest model under the respective queue size scenarios, addressing the dual scenario of queue sizes - 32 and 64 packets.

An evaluation of the proposal outlined in this section is expounded upon in Section IV. The primary objective is to provide an analysis of the synthetic data generated by TimeGAN and its applicability in training an RL agent through an offline approach, contrasting it with an agent trained in an online format.

IV. EVALUATION

We evaluated our proposal after training as described in Section III-D. We built 27 models with varying hyperparameters (see Table II). In Section III-E, we calculated an evaluation

[1]https://github.com/ydataai/ydata-synthetic.

Algorithm 1 Optimal Model Selection

Input: $models32, models64$
Output: $bestModel32, bestModel64$
Initialisation :
$metrics32 \leftarrow calcMetricsFeatures(models32)$
$metrics64 \leftarrow calcMetricsFeatures(models64)$
for each $mod32$, $mod64$ in $metrics32$ and $metrics64$ **do**
  $sums32[mod32] \leftarrow sumFeatsMetrics(metrics32)$
  $sums64[mod64] \leftarrow sumFeatsMetrics(metrics64)$
end for
$bestModel32 \leftarrow min(sums32)$
$bestModel64 \leftarrow min(sums64)$
return $bestModel32, bestModel64$

metric for each model. This allowed us to rank the models based on performance, with the best model having the lowest metric value. Conversely, the worst model had the highest metric value, indicating greater disparities compared to real data.

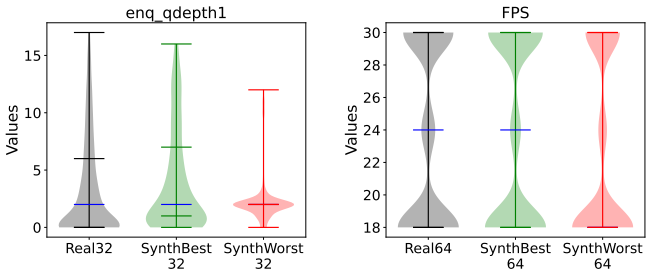


Fig. 4. Data distribution of enq_qdepth1 and FPS features for 32-buffer and 64-buffer, respectively.

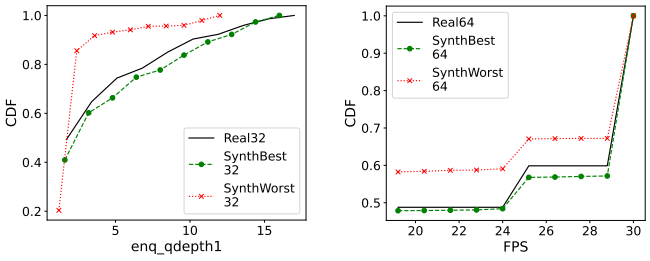


Fig. 5. CDF of enq_qdepth1 and FPS features for 32-buffer and 64-buffer, respectively.

Given the extensive array of features under analysis, it is unfeasible to present the complete set of examined features in this paper[2]. Consequently, we have chosen to showcase plots for only two specific features. Fig. 4 depicts two violin plots: the first delineates the distribution of the `enq_qdepth1` (an INT metadata) feature within the context of a 32-buffer size, while the second illustrates the `FPS` feature in the context of a 64-buffer size.

[2]A repository containing the codes and analyses can be accessed at: https://github.com/thiagocaproni/noms2024.

For comparison, we explain the differences in distribution between real data, synthetic data generated by the best model, and synthetic data produced by the worst model (for 32-buffer size). This examination reveals that the distributions generated by the best model bear a closer resemblance to those observed in real data in comparison to the inferior model. In the case of the `enq_qdepth1` feature of the worst model, disparities exist when juxtaposed with real data. Contrarily, the distribution generated by the inferior model in the context of the `FPS` feature aligns more closely with the real data, particularly when considering interquartile ranges, albeit exhibiting differences in median values. It is imperative to note that while the inferior model may be good in replicating the distribution of certain features, it may fall short in replicating others. Hence, the critical criterion for model selection hinges upon its ability to emulate favorable distributions across a majority of features while maintaining fidelity to real data.

An alternative method for assessing the data distribution of each feature involves the use of cumulative distribution function (CDF) plots, as exemplified in Fig. 5. The figure comprises two separate plots illustrating the CDFs for the `enq_qdepth1` and `FPS` features, respectively. In both instances, it is evident that the best model exhibits a more favorable data fit than real data, while the worst model falls short in this regard.

Fig. 6 illustrates the correlation matrices for real data, the best model, and the worst model. Certain features in real data display noticeable intercorrelations, consistent with the anticipated behavior of network and video-related attributes. These expected correlations also extend to the synthetic data generated by the models. However, it is worth noting that some of the trained models cannot accurately capture these correlations, resulting in less well-defined patterns. Despite these variations, our approach of utilizing a metric for model selection has proven effective for our use case. This effectiveness is evident in the correlation matrix of the best model, as shown in Fig. 6, where the expected correlation patterns are better reproduced when compared to the worst model.

In [15], various methods for evaluating the quality of data generated by a TimeGAN model are presented. These evaluation mechanisms encompass visualization techniques utilizing t-SNE and PCA, as well as the application of regression models designed to predict samples generated by TimeGAN models. However, the visualization approach requires human intervention for manual assessment, thereby necessitating subjective analysis to identify the most suitable model. In contrast, employing a regression-based evaluation methodology proved to be less suitable in our specific case. Our data exhibits non-stationary characteristics, rendering the creation of an effective regressor challenging.

Nevertheless, we implemented a simple Recurrent Neural Network (RNN) using the Keras framework for regression purposes, enabling a comparative analysis of real and synthetic data. We employed the Adam optimizer and Mean Absolute Error (MAE) as the loss function for this task. Our RNN was primarily designed to predict the last sample within a data window based on preceding samples. However, as expected, the results for both real and synthetic data did not meet our

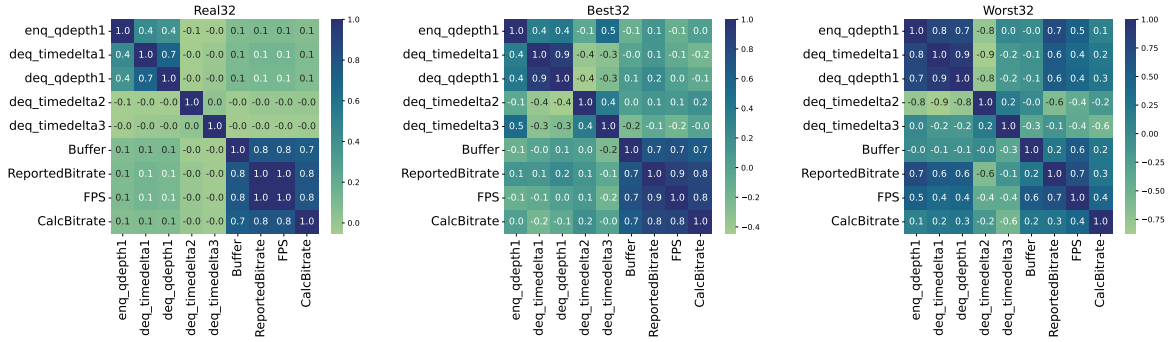


Fig. 6. Correlation matrix to show the relationship among features for real, best, and worst models for 32-buffer size.

expectations. The code for this regression model is available in our Git repository for reference.

As a result of the aforementioned challenges, our model selection methodology relied exclusively on statistical analyses. Furthermore, we adopted an alternative approach for the evaluation of our synthetic data. We devised an RL agent, trained on both real and synthetic data, and subsequently assessed its performance under both conditions. This evaluation is provided in Section IV-A.

A. Reinforcement Learning

To evaluate the feasibility of employing TimeGAN as a network traffic simulator for video applications, we modeled a Deep Q-Network (DQN) [27] agent using the Multi-layer Perceptron (MLP) architecture, comprising an input layer with units for each INT feature, 2 hidden layers with 24 Rectified Linear Units (ReLU) each, and an output layer with 2 units - one for each action the agent could take (increase queue size to 64 or decrease it to 32).

We elaborated the agent-environment workflow considering the DASH Server video streaming chunk size as time steps, thus, the agent should take an action to increase or decrease the queue size every 4 seconds to maximize the video client's Local Buffer Occupancy (LBO) and FPS. These metrics are fundamentally correlated, such that as higher the LBO, the higher the FPS will be. Hence, considering that the LBO and FPS variations are related to the queue size, the reward for the agent action was calculated only on the environment's next state observation. After each action, the INT metrics were accordingly saved in their respective datasets to train the TimeGAN in the later stage, as depicted in Fig. 2.

In this sense, the first agent was trained on the real setup throughout the video transmission, afterwards, we trained a second agent using only the synthetic data generated by the TimeGAN. For such, we followed the same methodology aforementioned, but instead of using the real setup, we fed the agent with the synthetic INT metrics by the action taken, thus simulating the real network behavior. In order to verify whether such a model is suited for a real-world scenario, we applied both pre-trained agents on the real setup again and compared their results. For such, the DQN weights were initialized from the respective models saved on the previous training stage instead of randomly.

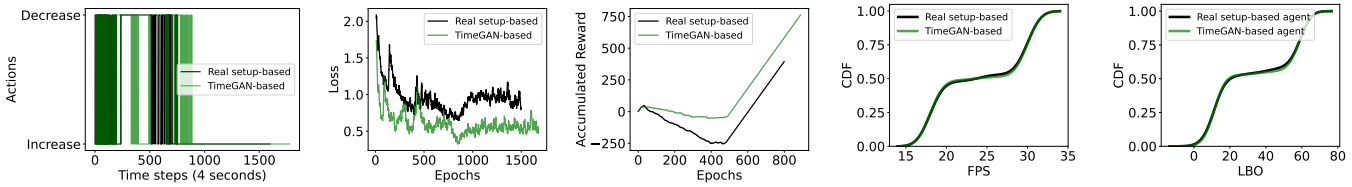
As can be observed in Fig. 7, the behavior of both agents regarding the actions taken, the rewards received, and the training loss were similar, indicating that the TimeGAN can be used as a simulator to train an agent that is intended for a real setup. Such premise is corroborated by the results shown in Figs. 7(d) and 7(e), in which it is possible to notice that the FPS and LBO obtained with each agent are very close. Table III shows the quantitative assessments of the information depicted in these figures, which describes the similarity between the video client QoS metrics in accordance with the real setup-based and TimeGAN-based agents, respectively.

Moreover, another important contribution of the proposed method refers to the time needed to train each agent. While the real setup-based agent trained on the fly throughout the video streaming took 1 hour, the synthetic data generated by TimeGan enabled us to reduce the training time to $\approx$3 minutes using the same number of samples that we usually would obtain by using a real setup. Such finding, in addition to the results previously showed, contributes to the elaboration of more precise RL agents in similar contexts, since the reduced training time facilitates the hyperparameter search for the deep learning models comprising them, and for the agent-environment interaction settings.

V. Lessons Learned

In this section, we share insights gleaned from our study, which can benefit the community engaged in synthetic data generation in the context of computer networks. While our focus is on a video application within the context of PDPs, the principles, and methodology we present can hold applicability across various domains that exhibit similar behaviors and metrics.

1) Dataset Characterization of Telemetry Metrics in a Video Application Over PDPs: The data acquired through our conducted experiments exhibited dynamic attributes. Consequently, the time series data for each feature collected proved to be highly intricate, posing a challenge for GAN model learning. Furthermore, given that we collected two distinct datasets—video and network metrics—the initial hurdle was to determine the most rational approach for merging them effectively. To address this, an analysis was imperative, necessitating the selection of specific behaviors for filtration. This choice held significant sway over the subsequent GAN training



(a) Actions (b) Loss (c) Reward (d) FPS (e) LBO

Fig. 7. Performance analysis graphs for RL trained in real setup vs. synthetic data.

TABLE III
COMPARISSON OF QOS FOR RL TRAINED IN REAL SETUP VS. SYNTHETIC

Agent type	Proportion of frames played at 18 FPS	Proportion of frames played at 24 FPS	Proportion of frames played at 30 FPS	LBO $>$ 30 seconds
Real setup-based	0.48	0.05	0.47	0.46
TimeGan-based	0.49	0.02	0.48	0.47
QoS similarity	99%	97%	99%	99%

process. Hence, it underscores the criticality of conducting a comprehensive dataset analysis and pre-processing assessment before embarking on GAN training.

2) Hyperparameters Definition on Trainings: The optimization of GANs, as a general practice, presents inherent challenges that necessitate careful consideration in training tasks. The determination of hyperparameter values is contingent upon the specific dataset characteristics, involving key choices such as batch size, the number of hidden dimensions, and the length of windows employed as input to the model. It is essential to underscore that there is no universally optimal approach for setting hyperparameters, as the selection of values is typically derived empirically. This process entails a fine examination of the real dataset's intrinsic attributes to arrive at reasonable choices.

3) Mechanism to Select the Best Models: Evaluating and selecting optimal models post-TimeGAN training poses challenges. Traditional approaches like regression analysis are unsuitable due to the dynamic nature of our real setup dataset. Visual analyses lack automation for model selection. To address this, we introduced a straightforward statistical metric for ranking models trained with varied hyperparameters. Our evaluation demonstrates its effectiveness in aligning data distributions with the real dataset, simplifying the comparison of synthetic data without the need for additional time-consuming machine learning models.

4) Application of synthetic data to train an RL model: The results derived from the utilization of synthetic data for training our RL model have yielded utility, underscoring the effectiveness of synthetic data generated by TimeGAN within the domain of computer networks. The similarity between the RL agents is high. Also, the time required for RL model training using synthetic data stands in stark contrast to that of a real setup, with the former taking a sheer three minutes as opposed to the latter's one-hour duration for experimentation. This methodology showcases a potential for broader applicability across diverse contexts and machine learning methodologies. These findings hold implications for the field, emphasizing the feasibility of leveraging synthetic data instead of real setups, thus facilitating accelerated experimentation and model training while retaining efficacy

Our experience with TimeGAN-generated synthetic data is significant for our future work. It boosts our confidence in using synthetic data for various studies in PDPs. Additionally, synthetic data can be a valuable resource for sharing our research environments with other researchers and groups, promoting collaboration and knowledge dissemination, in contrast to traditional sharing of real setups. Section VI summarizes our key findings and conclusions, providing insights into our future research plans.

VI. CONCLUSIONS AND FUTURE WORK

This study presents the adoption of TimeGAN as a surrogate simulator, substituting a real setup for a video application in PDPs. We introduce a statistical metric for optimal model selection during training and conduct a comprehensive evaluation of the synthetic data generated by TimeGAN, comparing its distribution with that of real features. Two RL agents are trained, one with online training using a real setup and the other with offline training utilizing synthetic data. Our findings demonstrate the effectiveness of our statistical metric in selecting superior models, even across models with differing hyperparameters.

Moreover, we successfully showcase the feasibility of offline RL training with synthetic data, producing results that are comparable to their online-trained counterparts. The versatility of TimeGAN is underscored, given its ability to create diverse, balanced, and privacy-preserving datasets, thereby accelerating model training. While the focus of our study revolves around a video application, the model selection methodology we propose, along with the insights we have unveiled, holds applicability in other contexts and applications within the computer network domain. This assertion is underpinned by the data characteristics typical of this domain, which extend the utility of our contributions beyond the specific scope of our investigation. In our future research, we intend to delve into the exploration of DoppelGANger as a framework for generating synthetic time series data derived from our real setup.

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014.

[2] H. Navidan, P. F. Moshiri, M. Nabati, R. Shahbazian, S. A. Ghorashi, V. Shah-Mansouri, and D. Windridge, "Generative adversarial networks (gans) in networking: A comprehensive survey & evaluation," *Computer Networks*, vol. 194, p. 108149, 2021.

[3] C. Gray-Preston, "Ai network applications," Sep 2023. [Online]. Available: https://www.atis.org/tops-council/ai-network-applications/

[4] L. C. de Almeida, R. Pasquini, and F. L. Verdi, "Using machine learning and in-band network telemetry for service metrics estimation," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, 2021, pp. 33–39.

[5] J. Shi, M. Sha, and X. Peng, "Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 887–901. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/shi

[6] J. Liu, M. Nogueira, J. Fernandes, and B. Kantarci, "Adversarial machine learning: A multilayer review of the state-of-the-art and challenges for wireless and mobile systems," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 123–159, 2022.

[7] C. Zou, F. Yang, J. Song, and Z. Han, "Generative adversarial network for wireless communication: Principle, application, and trends," *IEEE Communications Magazine*, pp. 1–7, 2023.

[8] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[9] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016.

[10] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020.

[11] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," 2017.

[12] C.-K. Lee, Y.-J. Cheon, and W.-Y. Hwang, "Least squares generative adversarial networks-based anomaly detection," *IEEE Access*, vol. 10, pp. 26 920–26 930, 2022.

[13] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[14] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar, "Using gans for sharing networked time series data: Challenges, initial promise, and open questions," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 464–483. [Online]. Available: https://doi.org/10.1145/3419394.3423643

[15] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[16] M. H. Naveed, U. S. Hashmi, N. Tajved, N. Sultan, and A. Imran, "Assessing deep generative models on time series network data," *IEEE Access*, vol. 10, pp. 64 601–64 617, 2022.

[17] Y. Hua, R. Li, Z. Zhao, H. Zhang, and X. Chen, "Gan-based deep distributional reinforcement learning for resource management in network slicing," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[18] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L.-C. Wang, "Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and challenges," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 44–52, 2019.

[19] R. K. Gupta, S. Mahajan, and R. Misra, "Resource orchestration in network slicing using gan-based distributional deep q-network for industrial applications," *The Journal of Supercomputing*, vol. 79, no. 5, pp. 5109–5138, 2023.

[20] Q. Miao, Y. Lv, M. Huang, X. Wang, and F.-Y. Wang, "Parallel learning: Overview and perspective for computational learning across syn2real and sim2real," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 3, pp. 603–631, 2023.

[21] Q. Liu, N. Choi, and T. Han, "Atlas: Automate online service configuration in network slicing," in *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 140–155. [Online]. Available: https://doi.org/10.1145/3555050.3569115

[22] F. Falahatraftar, S. Pierre, and S. Chamberland, "A conditional generative adversarial network based approach for network slicing in heterogeneous vehicular networks," *Telecom*, vol. 2, no. 1, pp. 141–154, 2021. [Online]. Available: https://www.mdpi.com/2673-4001/2/1/9

[23] P. Mani, E. S. Gopi, H. Shekhar, and S. Chandra, "Generative adversarial network and reinforcement learning to estimate channel coefficients," in *Machine Learning, Deep Learning and Computational Intelligence for Wireless Communication*, E. S. Gopi, Ed. Singapore: Springer Singapore, 2021, pp. 49–58.

[24] C. Qian, W. Yu, C. Lu, D. Griffith, and N. Golmie, "Toward generative adversarial networks for the industrial internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 19 147–19 159, 2022.

[25] E. Brophy, Z. Wang, Q. She, and T. Ward, "Generative adversarial networks in time series: A systematic literature review," *ACM Comput. Surv.*, vol. 55, no. 10, feb 2023. [Online]. Available: https://doi.org/10.1145/3559540

[26] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.