IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Generating Synthetic Vehicle Data Using Decentralized Generative Adversarial Networks

BASEM SHAKER[1], GASTONE PIETRO ROSATI PAPINI[1], MATTEO SAVERIANO[1], and KUO-YUN LIANG[2]

[1]Department of Industrial Engineering, University of Trento, Trento, Italy.
[2]Scania CV AB, Södertälje, Sweden.

Corresponding author: Gastone Pietro Rosati Papini (e-mail: gastone.rosatipapini@unitn.it).

ABSTRACT

Ensuring the privacy of personal data is crucial in the era of big data, especially in the transportation industry where sensitive data needs to be processed to develop intelligent vehicle technologies. In particular, collecting and analyzing anomalous data is essential for improving vehicle safety and performance, but accessing such data is often difficult and costly. To address this problem, we propose a novel approach to generating synthetic anomalous data using Generative Adversarial Networks (GANs) and Federated Learning (FL). The proposed learning strategy is decentralized, utilizing data generated by each vehicle to locally train individual discriminators. These discriminators then share only the loss values and weights with a centralized generator. Consequently, the GAN model is trained without exchanging raw data, thereby ensuring the privacy of personal information. Our approach involves a Convolutional Neural Network (CNN)-based architecture for both the generator and discriminator, with the generator residing on the server and a separate discriminator at each vehicle. This design reduces the computational demand on edge devices and enables us to train the GANs using FL. We experiment with different FL strategies and find that the best performer favored the least forgiving discriminator considering data from a pool of vehicles. Our results demonstrate the feasibility of using FL with CNN-based GANs to generate synthetic time-series data for training models in a privacy-preserving manner. This approach has potential applications in the transportation industry, particularly in the context of intelligent vehicles and automated driving systems.

INDEX TERMS Generative Adversarial Network, Federated Learning, Vehicle Data, Time-Series

I. INTRODUCTION

INTELLIGENT vehicles have the potential to revolutionize transportation by enhancing safety, efficiency, and comfort. However, developing and optimizing Machine Learning (ML) models for these vehicles requires large amounts of data. A possible solution consists in using synthetic data like images [1]. Ideally, synthetic data should be generated automatically by a generative model, like the Generative Adversarial Network (GAN) [2]. However, also generative models need to be trained on real to generated synthetic data with high fidelity. Anonymously collecting this data is essential to protecting privacy, while ensuring that the ML models are trained on diverse and representative data to achieve good performance. Data privacy and non-Independent and Identically Distributed (non-IID) nature of the data across vehicles pose challenges to this development. In this work, we propose a decentralized Generative Adversarial Network (GAN) model that can generate synthetic data, mimicking the behavior of local data, while preserving privacy, i.e., without sharing sensitive information across different vehicles.

We focus on three main contributions:

1) We present a decentralized GAN model that can effectively learn from multiple non-IID time-series data across vehicles. Our model is trained on edge devices, with individual discriminators on each vehicle. This approach ensures the convergence to a solution that fools all discriminators. Experimental results demonstrate that our method outperforms existing centralized methods, such as TimeGAN, in terms of the quality of the generated synthetic data.
2) We introduce a novel discriminator architecture that uses Convolutional Neural Networks (CNNs) and an adaptive pooling layer. This design facilitates the generation of sequences with variable lengths, improving

the generator's ability to accurately reproduce such sequences.

3) We propose a unique normalization technique to enhance the training process. This method determines the global minimum and maximum values for each feature considering the lowest minimum and highest maximum values received from all workers. The global boundaries are then disseminated to all workers for normalization of their local data. Experimental evidence shows that this approach outperforms the normalization based on the known range of the specific sensors involved.

These novel contributions have significant implications for the development of decentralized GANs in intelligent vehicles and other domains where privacy is paramount.

II. RELATED WORK

GANs are powerful models first introduced by Goodfellow et al. These involve training two neural networks, a generator and a discriminator, in a zero-sum game [2]. Both the generator and the discriminator are typically based on a Convolutional Neural Network (CNN), a fundamental building block in deep architectures [3], that alternates between convolution and pooling layers. Having a pooling layer with fixed stride can limit the performance, therefore, adaptive pooling layers have been proposed for object detection [4], video annotation [5], and feature extraction from near-infrared spectroscopy data [6]. Adaptive pooling is also effective in learning large graph neural networks [7]. Therefore, in this work, we also exploit adaptive pooling to train GANs on time-series data.

In the realm of time-series data, TimeGAN [8] is a notable development. This model employs Long Short Term Memory (LSTM) [9] to encode data into a latent space and reconstruct it, thereby preserving the correlations and temporal dynamics of the original data.

In response to the growing importance of data privacy, decentralized GANs have become prominent. These models are trained on edge devices without centralizing the data [10]. An extensive review for data privacy and security can be found in [11]. Notable examples include Forgiver-First Update (F2U) [12] and Forgiver-First Aggregation (F2A), which train individual discriminators on local data and update a generator to deceive the most lenient discriminator. In [13] the authors investigate and mitigate the problem of non-IID in the image context. F2A further enhances this process by adaptively aggregating discriminators with a focus on the most forgiving ones. FeGAN [14] advances this concept by fully distributing both the generator and the discriminator, allowing each worker to locally train a private GAN. Another recent work, primarily focused on images, is FedGAN [15]. Ensembles of Fine-tuned Federated GAN (EFFGAN) [16] introduces an ensemble of local expert generators capable of learning the data distribution across all workers, effectively reducing client drift. It also demonstrates high communication efficiency and the capability to train with numerous local epochs. Other examples are collected in [17], a very recent review featuring the use of GANs for time-series, but none of which in the field of autonomous vehicles.

An intersection of Federated Learning, GANs, and autonomous vehicles is explored in the work by Xu et al. [18], which focuses on the reliability of GAN generated data for training and validating perception systems in autonomous vehicles. The authors propose solutions for trusting GAN-generated data, particularly for handling corner cases that are difficult to collect large amounts of data for.

Despite these advancements, our work identifies and addresses certain limitations. We adapt the F2U and F2A architectures to work with time-series data, introduce a novel discriminator architecture, and propose a unique normalization technique. We demonstrate that these innovations outperform existing methods in terms of the quality of the generated synthetic data.

III. THE GENERATIVE ADVERSARIAL NETWORK ARCHITECTURE

In our approach, we adopted F2U to learn from non-IID data. In the original architecture, designed for image data, the layers used 2D convolution and transpose convolution, with the width and height dimensions representing the spatial dimensions of the images. The original F2U uses CNNs for both the generator and discriminator networks. The generator supports five layers of 2D transpose convolution with batch normalization and Rectified Linear Unit (ReLU) activation functions. On the other hand, the discriminator includes five layers of 2D convolution with spectral normalization [19] and Leaky ReLU, then later a Fully Connected (FC) layer. It should be noted that the input size to the FC layer must be initialized based on the input size. That means changing the input size to the discriminator is unfeasible in the current setup. In the rest of this section, we show how to adapt F2U to learn from time-series data.

A. TIME-SERIES ARCHITECTURE ADAPTATION

The first step to adapt the architecture for time-series data is to set up the convolutional layers. This can be done using one of the following structures (see Fig. 1):

1) 1D convolution and transpose convolution can be organized in two ways: First, each channel can represent a specific feature, with the width dimension symbolizing the time domain, resulting in M channels and a 1D convolution width of T seconds. Alternatively, each channel can exist in the time domain, with the width representing a different feature, leading to T channels and a 1D convolution width of M features.
2) 2D convolution and transpose convolution, with the width axis in time domain and the height axis in feature domain. Each channel would be a copy of the previous one, except the first row is moved to the bottom.

We use 1) (top-left structure in Fig. 1) in our approach. The reason is that applying convolutions along the time dimension allows the network to learn patterns across different time

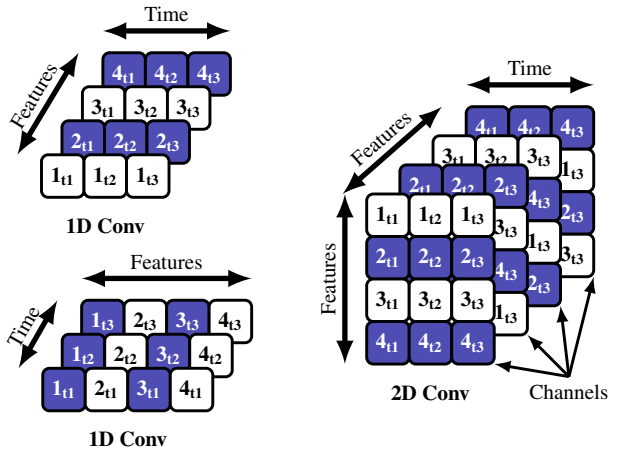


FIGURE 1. Depiction of 1D and 2D convolution layers for time-series analysis with feature count of 4 and time dimension of 3. Cell entries are denoted as X_{ty}, where X represents the feature number and y denotes the time step. Alternating features are color-coded for enhanced visibility.

points in the data. For a given feature, there exists more dependency on the previous time step of that feature compared to the other features [20]. While the use of 2D convolution with feature mirroring might improve the network's ability to identify feature dependencies, it would also significantly complicate the network and increase its size. Furthermore, implementing 2D convolution would require modifying the output from the generator to match the new 2D convolution setup. For these reasons, we did not pursue a 2D convolution in this paper. We switched both the generator and discriminator to 1D convolution as previously mentioned and further modified the sizes of the kernel, stride, and output channels to fit the feature count and sequence width of the time-series data.

B. TIME-SERIES BASED NEURAL NETWORK ARCHITECTURE

The proposed generator and discriminator are shown in Fig. 2. In this paper, we explored two sliding window approaches for creating training samples. The first uses a fixed-length window of with size t and step size s to divide the original time series into N sub-sequences. The second approach employs a variable-length (dynamic) window. Using a dynamic window when training a GAN may enhance the generator's ability to create sequences of varying lengths, as opposed to a fixed-size window that restricts the generator to producing sequences of a specific length. Implementing a dynamic window is a challenge considering that the discriminator is CNN-based with a FC layer. Once such a network is initialized, changing the input dimensions during training can cause problems with the internal structure and weights of the network. Nonetheless, we employed the use of adaptive average pooling after the convolution layers to make the network work with inputs of different sizes [21].

The dynamic window implementation is simpler for the generator as it only requires changing the dimension of the noise input. In order to control the output sequence width, we used Alg. 1 to calculate the size of the required noise input. When provided with the target sequence width and the generator parameters (kernel sizes, paddings, and strides), this algorithm approximates the noise size by working backwards through the layers calculating the input size of each layer. The noise size is then rounded up and used to calculate the actual output size by working forwards through the layers. Note that the output size may not be exactly equal to the target sequence width due to the fixed sizes of the generator parameters.

Algorithm 1 Sequence Length Finder

Input: kernels k, paddings p, strides s, target_width w_t

$o_s \leftarrow w_t$ ▷ Output size
for $i \leftarrow$ `len`(k) to 0 **do**
  $i_s \leftarrow (o_s - k_i + 2p_i)/(s_i + 1)$ ▷ Input size
  $o_s \leftarrow i_s$
$n_s \leftarrow$ `ceil`(o_s) ▷ Noise size
$i_s \leftarrow n_s$
for $i \leftarrow 0$ to `len`(k) **do**
  $o_s \leftarrow (i_s - 1)s_i + k_i - 2p_i$
  $i_s \leftarrow o_s$
return n_s, o_s

We further use the calculated actual sequence width from the generator to divide and create samples out of the real time-series data. That means both the real and synthetic samples would have the same dimensions. We selected an initial sequence width to initialize the discriminator. This involves calculating the number of neurons after flattening the output from the last convolution layers using the backward loop in Alg. 1. The calculated number of neurons is then used to initialize the expected input size for the linear layer. Furthermore, it also specifies the required output dimension for the adaptive average pooling layer to match the input size of that linear layer. Following steps outline the training sequence:

- Create a list of increasing sequence lengths [e.g., 64, 128, 192, 256, and 320 seconds].
- Calculate the required noise size and actual sequence length for each of the sequence lengths using Alg. 1 [e.g., 65, 129, 193, 257, and 321 seconds].
- Use the shortest sequence length to initialize discriminator and specify input size for the linear layer. Choosing the shortest sequence guarantees that the average pooling layer would always be down-sampling.
- To ensure that the neural network sees the full data-set at every epoch while maintaining a constant batch size, it is important to keep the sequence length constant during the iterations of each epoch. Both real and synthetic data are sampled to match this sequence length. As a result, the iteration count will vary across epochs.
- During the sequence length change, the output from the last convolutional layer in the discriminator changes.

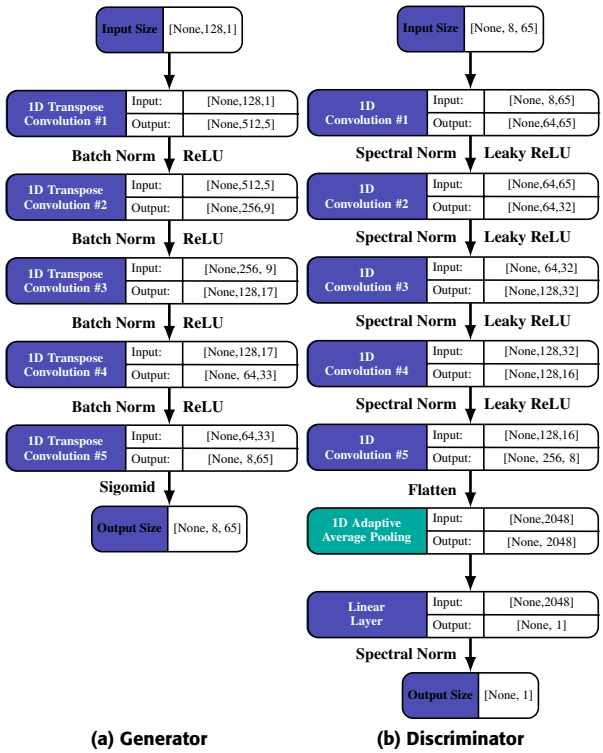


FIGURE 2. **Generator and discriminator architecture for time-series input.**

The adaptive average pooling layer makes sure to change that dimension to the initialized dimension for the linear layer.

The proposed F2U architecture is capable of processing time-series data and of handling sequence of varying length both for training and generation.

IV. DECENTRALIZED TRAINING STRATEGIES

During centralized GAN training, the generator takes noise as input and produces fake data. The discriminator tries to classify it as fake or real. If the discriminator is successful, it provides feedback to the generator to help it improve its data generation. This process continues until the generator is able to produce data that is realistic enough to fool the discriminator.Centralized training requires sharing of (possibly sensistive) data, which makes it unsuitable in our setting. On the contrary, decentralized GAN training exploits one generator and multiple discriminators. At the server side, the generator takes a random Gaussian noise as input and outputs fake samples. The fake samples are sent to each worker. The local discriminator at each worker is trained on both the local real samples and the fake samples received from the server.Being each discriminator trained locally, there is no sensitive data from the vehicles shared with the server. Two losses are then calculated, the loss on real samples (Real Loss) and the loss on fake samples (Fake Loss). The fake loss is expected to be high when the discriminator is deceived by the fake samples. Conversely, if the fake loss is low, it means the discriminator was able to classify the fake sample correctly. The fake losses from all workers are compared to each other. Depending on the considered strategy, information about the fake losses and the weights of the discriminators are used to training the centralized discriminator. Also in this phase, no sensitive data from the vehicles is shared with the server.

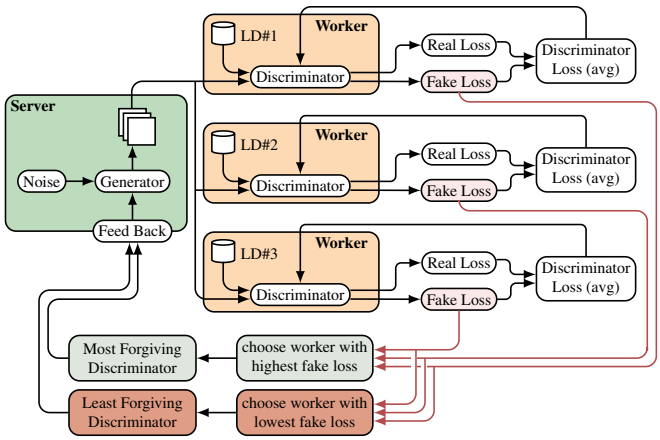


FIGURE 3. **Comparison of the most and least forgiving networks training process. "LD" stands for local data, which is private and unique for each worker.**

Algorithm 2 Weighted Federated Averaging [F2A]

Input: model_weights w_m, contribution c

$w_{\text{avg}} \leftarrow w_m[0,:]$ ▷ First row of w_m
for $j \leftarrow 0$ to $\texttt{len}(w_{\text{avg}})$ **do**
  $t_w \leftarrow \texttt{zeros\_like}(w_m[0,j])$
  for $i \leftarrow 0$ to $\texttt{rows}(w_m)$ **do** ▷ Loop over rows of w_m
    $t_w \leftarrow t_w + c[i]w_m[i,j]$
  $w_{\text{avg}}[j] \leftarrow t_w$
return w_avg

We examined three possible strategies to compare fake losses. If we choose the worker with the highest fake loss, this would be the discriminator that was most deceived by the fake samples, which could be called *most forgiving*. If we select the worker with the lowest fake loss, this corresponds to the discriminator that was most successful at distinguishing fake from real samples. This discriminator could be considered the *least forgiving* or the most formidable adversary. The generator at the server side is trained against the chosen discriminator using back-propagation, whether it is the *most* or *least forgiving*. The break down of the architecture and logic for both strategies are shown in Fig. 3.

The third strategy is a *weighted average*. In this setting, the discriminator parameters from each worker are all aggregated to form a global discriminator model. In order to emphasize the parameters of the discriminator with the highest loss (*most forgiving*), we use a weight that reflects the calculated loss. This can be achieved by employing a Softmax function as shown in Fig. 4. The Softmax function ensures that the output is a probability distribution, where each element is non-negative and the sum of all the elements is 1. The averaging

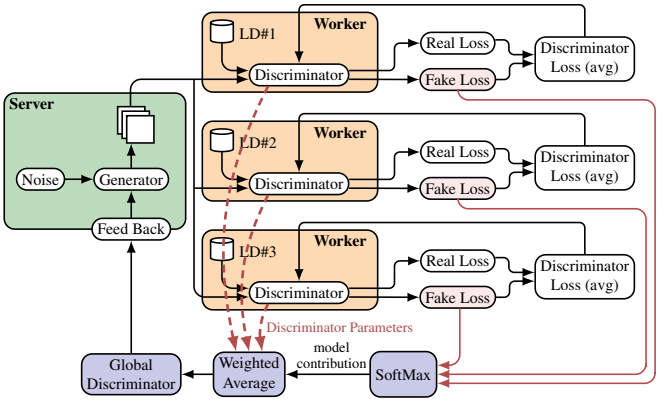


FIGURE 4. **Schematic of the F2A network using a Softmax function. "LD" stands for local data, which is private and unique for each worker.**

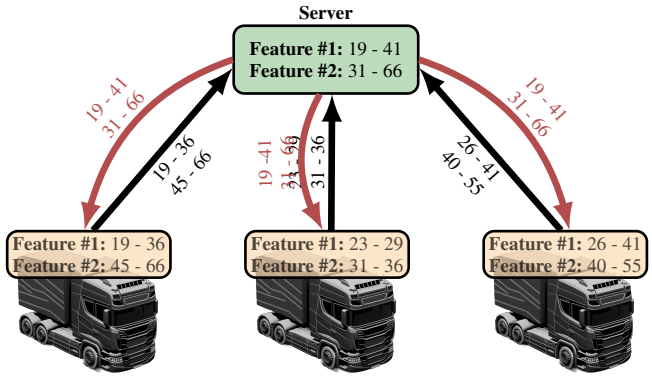


FIGURE 5. **Decentralized data normalization based on workers' minimum and maximum feature values.**

process is shown in Alg. 2. The algorithm described is a modified Federated Average algorithm, which emphasizes the *most forgiving* discriminator. However, we could easily switch the behaviour to favoring the *least forgiving* (lowest loss) by multiplying fake losses by -1 before applying the Softmax.

A. DATA-SET PRE-PROCESSING

The data-set comes from Scania CV AB and it is specific to the Air Processing System (APS) system on the vehicle. It comprises real operational data collected from different Scania trucks, encompassing both normal functioning and scenarios with system faults to represent a wide range of real-world conditions. The data-set is confidential, so the feature names have been redacted. It consists of 8 main features and the time domain is in seconds. Some of the data-set's features are discrete while others are continuous. Furthermore, the data-sets contain runs with normal behaviour, and others contain physical injected faults. Figure 7a shows three runs performed under the same road conditions, with the first one having a normal APS behaviour, the second one exhibiting anomalous behaviour, and the third having the fault injected midway through the run (mixed data). Features 4, 6, and 7 are overlapping.

To improve the training, data are usually normalized using, e.g., Min-Max normalization. In our case, each feature can be normalized based on the known (constant) capabilities of that specific sensor. However, in practice, the span between the minimum and maximum values for each feature is much larger than the normal operating values and my not represent the real distribution. To improve on the normalization technique, we implemented a slightly different approach. At the beginning of the communication, all workers would send their minimum and maximum values for each feature in their local data-set to the server. The server decides on minimum and maximum values based on the lowest minimum value and the highest maximum value among all the workers for each feature. The server sends the decided min and max values to all workers to be used to normalize their local data. The Figure 5 show an example of such strategy. Note that the output from the generator will have data that is normalized based on the worker limits. That means there need be an extra step to un-normalize the data to switch it back to either normalized data based on the known capabilities of the features, or to real values.

B. DECENTRALIZED TRAINING SEQUENCE

In the decentralized training sequence of our time-series GAN model, we employ the Adam optimizer with a learning rate of $2\mathrm{e}{-4}$, and parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$ for both generator and discriminator. In GANs, the discriminator D and the generator G are learned simultaneously by minimizing a loss, either the Mean Square Error (MSE) or the Binary Cross Entropy (BCE) [22].We found that the MSE loss outperformed the BCE, especially in a decentralized setting (see the Appendix). The MSE loss is defined as

$$\min_G \max_D \mathbb{E}_{x\sim p_{\mathrm{data}}(x)}[D(x)-1]^2 + \mathbb{E}_{z\sim p_z(z)}[D(G(z))]^2, \quad (1)$$

where $D(x)$ and $G(z)$ are the output of the discriminator and the generator, respectively, and z is the input of the generator, typically sampled from a Gaussian or uniform distribution $p_z(z)$. The goal of the generator is to approximate the distribution of the data x, while the discriminator is a classifier trained to distinguish training data from generated ones. In our setting, the noise dimension for the generator was set to 128 and the batch size used was 4. As shown in the Appendix, smaller batches tend to result in a reduced loss.Our testing showed that in our decentralized training setup, overriding the models with the winning discriminator led to a degradation in performance. Therefore, the chosen worker does not override the other workers. The full training sequence of the decentralized time-series GAN model involves the following steps:

- For each epoch, a target sequence width is determined in order to calculate the necessary noise size and actual sequence width output from the generator.
- Random noise is generated and used to create fake sequences, which are then sent to all workers.

- Workers create samples from their local real data that match the sequence width of the fake data received.
- During each iteration, workers feed their discriminator a real sample with a target of one, and then feed it a fake sample with a target of zero.
- Loss is calculated for both the real and fake data using MSE loss. The overall loss for each discriminator is the average of both losses.
- The overall loss is back-propagated to improve each discriminator using Adam optimizer [23].
- The fake losses for all workers are compared, and based on the training strategy either:
  - -- The worker with the highest/lowest loss is chosen.
  - -- The weighed average of all discriminator parameters is used to create a global discriminator.
- The chosen worker's discriminator or the global discriminator's output on the fake data is used to calculate the loss of the generator with a target of one.
- The generator loss is used to back-propagate and improve the generator using Adam optimizer.
- Steps repeat with a different target sequence width.

C. EVALUATION METRICS

In the early stages of our project, while the model was primarily designed for image data, we used established evaluation metrics such as the Fréchet Inception Distance (FID) score. The FID score is a popular choice for evaluating the quality of images generated by GANs. However, as we shifted our focus to time-series data, it became clear that the FID score is not directly applicable in this new context. This necessitated the development of a new evaluation methodology tailored to time-series data.

Consequently, we decided to implement an auto-regressive model as a primary evaluation measure. Auto-regression is a tried and tested method for the analysis and forecasting of time-series data and has proven effective as a performance metric for time-series forecasting models [24]. This auto-regressive approach allows for the evaluation of a GAN model in three distinctive ways:

- *Train on Real Test on Real (TRTR):* Using only real data is useful to establish a baseline that determines how well the model is able to capture the statistical properties of the real data. This can then be used to validate the performance of the other methods.
- *Train on Real Test on Synthetic (TRTS):* Instead of testing on real data, the synthetic data generated by the GAN model is used. This checks if the GAN model has learned the underlying structure of the real data effectively.
- *Train on Synthetic Test on Real (TSTR):* This is opposite of the previous method and will be used as a secondary metric. The method determines if the synthetic data generated by the GAN is of sufficient quality to be used in place of real data for training purposes.

The testing process focuses specifically on the generator network of the GAN. Figure 6 shows the details of the process

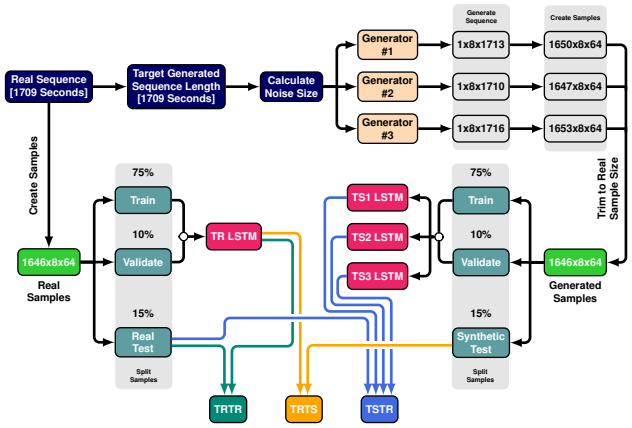


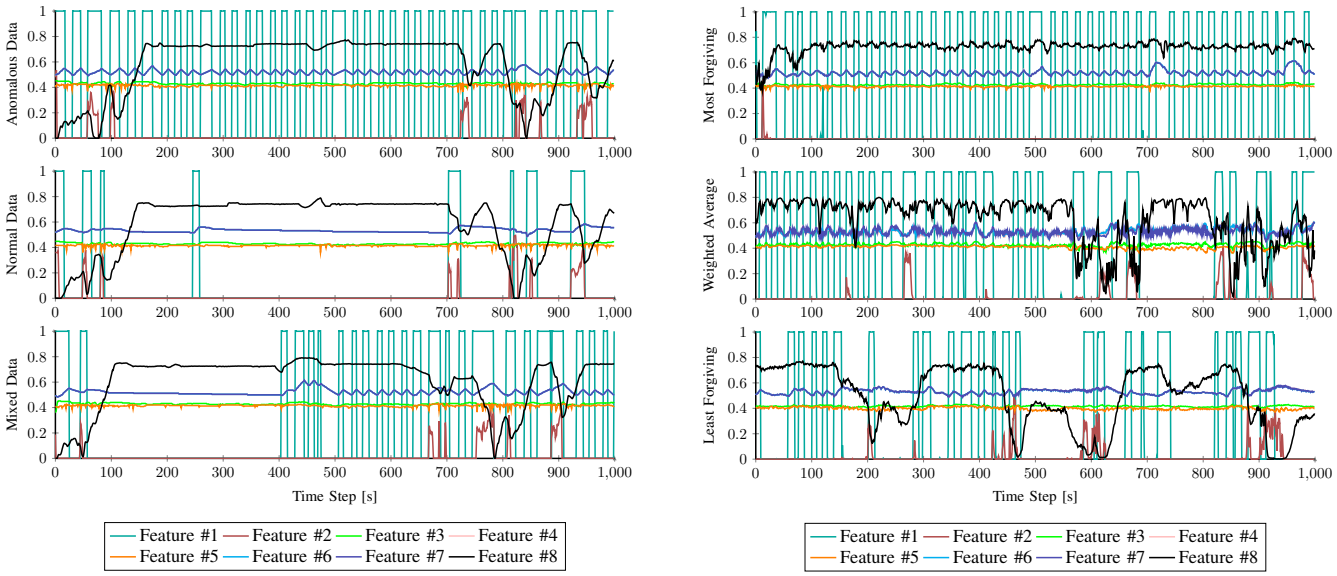
FIGURE 6. Detailed example of steps taken to prepare samples from generators for their evaluations.

from creating the samples to evaluating the three different testing methods. The details of the steps are as follows:

- The length of the real sequence is used as target sequence length for the generated data. This in turn calculates the required noise sizes for each generator being tested to generate a sequence close to the length of the real sequence.
- The real sequence along with the generated sequences from all generators are divided into samples of 64 seconds with a stride length of one second. The extra samples from the generated data are discarded to match the count of samples from the real data.
- Both real and generated samples are separately divided into three data-sets (75% training, 10% validation, 15% testing). The train and validate data-set for each sample set is fed to a unique LSTM auto-regression model for training. This means a separate auto-regression model for the real data and for each generator data. The model trained on the real data would create the train on real (TR) while the models trained on synthetic data would create the train on Synthetic (TS).
- The testing sub data-set from the real samples are both used to test the TR model to evaluate the TRTR method and test the TS model to evaluate the TSTR method. The testing sub data-set from each generated sample are used to evaluate the TRTS for each generator separately.

To asses the auto-regressive model's output we used three key metrics, which are the R2 Score, Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). R2 Score measures the predictive capacity of the auto-regression model. It's used in TRTR to establish a baseline performance, in TRTS to measure if the synthetic data preserves real data properties, and in TSTR to see if knowledge from synthetic data applies to real data.

MAE and RMSE are our chosen error metrics. MAE shows the average magnitude of prediction errors, while RMSE highlights larger errors, indicating outliers. Together, they reveal the error distribution and the resemblance of synthetic



(a) Real APS data. **(b) Generated APS data.**

FIGURE 7. **(a) The real non-IID APS time-series data (anomalous, normal, and mixed). (b) The output of our decentralized GAN (most forgiving, weighted average, and least forgiving).**

TABLE 1. **TRTR results for each data behaviour**

Data Type	TRTR		
	R2	MAE	RMSE
Anomalous	0.588	0.062	0.165
Normal	0.819	0.039	0.104
Mixed	0.530	0.071	0.172

TABLE 2. **Dynamic window effect (anomalous behaviour)**

Sequence Width	TRTS			TSTR		
	R2	MAE	RMSE	R2	MAE	RMSE
64 s	0.375	0.089	0.203	0.609	0.055	0.160
128 s	0.352	0.084	0.206	0.599	0.054	0.165
256 s	0.153	0.177	0.236	-1.84	0.215	0.284
Dynamic	0.550	0.070	0.171	0.621	0.051	0.162
TimeGAN	0.471	0.073	0.187	0.604	0.057	0.162

data to real data. Low MAE and RMSE with high R2 signal successful GAN models.

Tab 1 displays the TRTR results, wherein the auto-regression model yields a higher R2 score when evaluated on the normal behavior data-set, suggesting that the model more accurately captures the underlying patterns in normal data compared to anomalous or mixed data. These results will act as our baseline to verify the results of our TRTS results.

V. RESULTS

We compared the performance of a centralized GAN (i.e., a single worker) with our decentralized approach. For evaluation, we trained 3 separate auto-regression models for each data behaviour (normal, anomalous, mixed). Each TRTR model is needed to calculate the corresponding TRTS for the GAN getting tested. The auto-regression models are set-up to predict the last five time steps. We use the TRTR metric (Tab. 1) to verify the results when applied to synthetic data. In particular, we tested each TRTR on 10 different synthetic data samples from each generator to calculate an average for the TRTS metric. Similarly, we trained an auto-regression model with synthetic data 10 different times to calculate an average for the TSTR metric.

A. CENTRALIZED GAN RESULTS

We tested the model initially using fixed sequence widths of 64, 128, and 256 seconds. The results show a sequence width of 64 seconds being the better performer if we used the fixed sequence strategy width. After applying the dynamic window strategy, both visual inspection and the TRTS / TSTR (see Tab. 2) showed an improvement. This confirms that the dynamic window in fact improves the capabilities of the generator in contrast to using a fixed sequence width. Our dynamic window strategy, confirmed superior via visual inspection, notably outperformed the comparable yet non-decentralized TimeGAN model in both TRTS and TSTR metrics (see Tab. 2). This result reinforces the advantage of our method over related existing models.

The normalization we used in previous tests employed the first technique mentioned in Sec. IV-A, where each feature is normalized based on the known capabilities of that specific feature. We then further tested the second technique, where the normalization occurs based on the limits of the workers. The TRTS / TSTR results in Tab. 3 showed that using the worker limits yields a significant improvement. We adapted the second technique to all upcoming tests.

TABLE 3. Worker normalization effect (normal behaviour)

Normalization	TRTS			TSTR		
	R2	MAE	RMSE	R2	MAE	RMSE
Fixed Limits	0.472	0.096	0.169	0.743	0.133	0.164
Worker Limits	0.724	0.041	0.128	0.901	0.028	0.070

TABLE 4. FL strategy effect (mixed behaviour)

FL Strategy	TRTS			TSTR		
	R2	MAE	RMSE	R2	MAE	RMSE
Most Forgiving	0.504	0.070	0.176	0.426	0.067	0.177
Weighted Average	0.348	0.090	0.201	0.522	0.071	0.170
Least Forgiving	0.525	0.076	0.172	0.588	0.055	0.158

B. DECENTRALIZED GAN RESULTS

We set-up the network with one server and four workers. Workers #1 and #2 had anomalous data, while workers #3 and #4 had normal data. As shown in Fig. 7b, the output sequence generated by the most forgiving strategy only contain behaviour belonging to workers #1 and #2 (Anomalous), while the least forgiving strategy produces a mix of classes from both workers. This output shows both normal and anomalous behaviour, which looks similar to the mixed data behaviour shown in Fig. 7a. This can also be corroborated by examining the worker contribution count, which tracks the number of times each worker was selected for generator training (see Fig. 8. The most forgiving strategy exhibits a strong bias towards workers #1 and #2, while the least forgiving strategy exhibits a more balanced distribution. This explains the difference in output observed in the generated sequences. Figure 9 illustrates the top two components as analyzed by Principal Component Analysis (PCA), comparing real versus synthetic data across both the most and least forgiving strategies. It is noteworthy that the overlap between real and synthetic data is significantly greater in the least forgiving strategy than in the most forgiving one.

The behaviour of the most forgiving strategy can be understood as follows; in this approach, the generator is updated based on the worker with the highest loss on the fake data produced by the generator. This leads to the generator creating fake data that closely resembles the data of the selected worker. In subsequent iterations, it is expected that the previously selected worker will have a higher loss on the fake images, as the generator is more specialized to their data. On the other hand, the rest of the workers are likely to have a lower loss, insuring a much less likelihood of being selected. As the training process continues, the generator is expected to converge on a small subset of workers, effectively ignoring the data of other workers as the distribution of generated data is very different from their local data distribution.

We also implemented and tested two different forms of weighting for the averaging technique (F2A), one that favoured the most forgiving worker and another that favoured the least forgiving worker. We used the Softmax function to determine the contribution of each worker to the overall average model. In the next iteration, all workers used the newly averaged discriminator model. However, there was a noticeable output degradation as suggested by the TRTS / TSTR results and the output sequence shown in Fig. 7b.

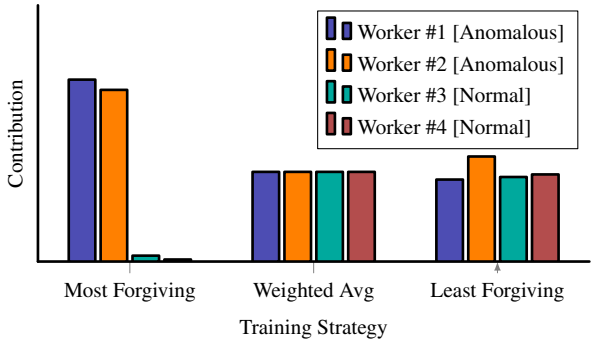
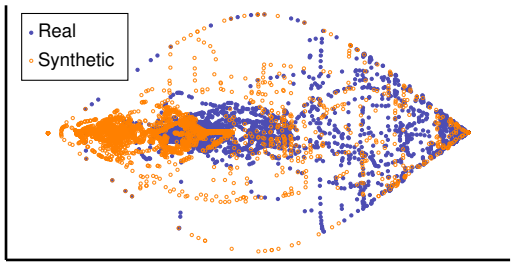
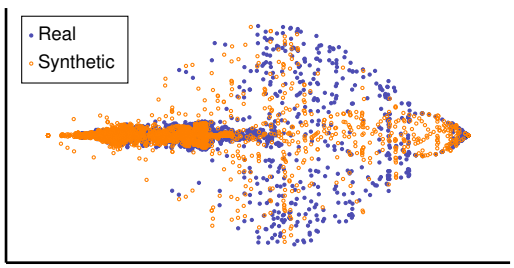


FIGURE 8. Worker contribution based on training strategy.



(a) Most Forgiving

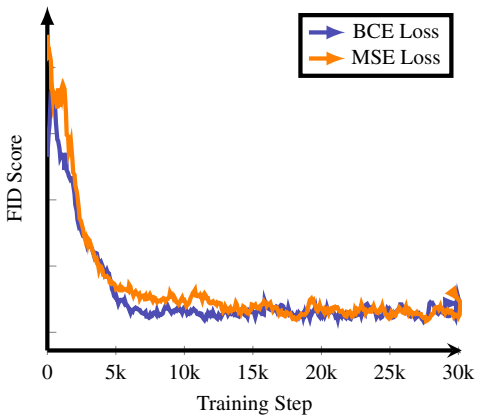


(b) Least Forgiving

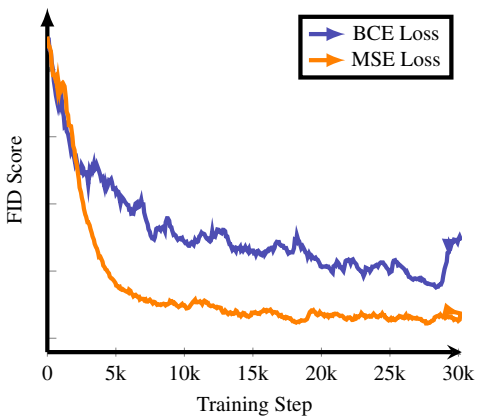
FIGURE 9. PCA output of real vs. synthetic mixed data based on FL strategy.

VI. CONCLUSION AND FUTURE WORK

We presented an approach to generate synthetic anomalous vehicle sensor data that combines GAN and Federated Learning (FL) to preserve the privacy of personal information. For the training strategies, we evaluated the two forms of the weighted averaging (F2A) technique, namely most and least forgiving. We found the least forgiving training strategy to be the most effective in decentralized learning. It was able to maintain stable and non-divergent training, even with non-IID data. Additionally, the generated data produced by this strategy included all classes represented by the different workers, indicating good generalization capability. Furthermore, the generated features maintained the same correlations exhibited in the real data. Testing concluded that the dynamic window strategy and normalization based on worker limits



(a) One worker centralized result



(b) Five worker decentralized result

FIGURE 10. The FID score results of BCE vs. MSE loss.

both improved the performance of the output.

In our future research, we plan to employ the use of *model compression* to reduce the size of the model while maintaining its performance level [25] and to investigate the applicability of *diffusion models* [26] to federated learning scenarios. Moreover, we will investigate the possibility to make our model explainable relying on approaches from *scenario engineering* [27], [28].

APPENDIX. HYPER-PARAMETERS TUNING ON IMAGES

To realise a work with these characteristics, a hyper-parameters tuning is a must. For this reason, we conducted a series of preliminary work on image-based GANs before adapting it to time-series data. In particular we studied:

- The effects of different loss functions, particularly MSE and BCE. In a centralized setup, there were no major difference between the performance of BCE and MSE loss, however, when we move on decentralized training MSE performed significantly better than BCE. See Figure 10.
- The impact of batch size on the performance of the image-based GANs. The network was able to learn much faster using a smaller batch size of 16 compared to 64. Both the image output and the FID score confirm that the output improved with a smaller size batch given a larger number of workers and limited local data. See Figure 11. This highlights the need to use small batches, indeed this result is confirmed in the FL on time-series.

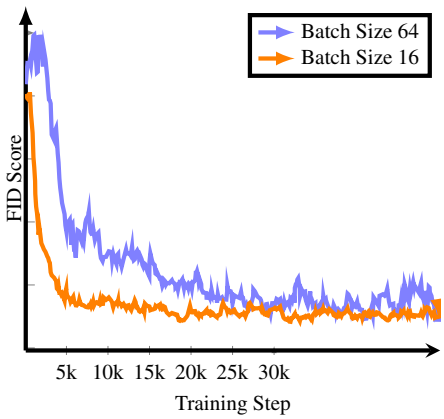


FIGURE 11. FID score comparison with different batch size for the image-based GAN model.

REFERENCES

[1] X. Li, K. Wang, X. Gu, F. Deng, and F.-Y. Wang, "Paralleleye pipeline: An effective method to synthesize images for improving the visual intelligence of intelligent vehicles," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[2] 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*, 2014.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[4] Y.-H. Tsai, O. C. Hamsici, and M.-H. Yang, "Adaptive region pooling for object detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[5] D. Liu, Y. Zhou, X. Sun, Z. Zha, and W. Zeng, "Adaptive pooling in multi-instance learning for web video annotation," in *IEEE International Conference on Computer Vision Workshops*, 2017.

[6] H. Chen, A. Chen, L. Xu, H. Xie, H. Qiao, Q. Lin, and K. Cai, "A deep learning cnn architecture applied in smart near-infrared analysis of water pollution for agricultural irrigation resources," *Agricultural Water Management*, 2020.

[7] E. Ranjan, S. Sanyal, and P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," in *AAAI Conference on Artificial Intelligence*, 2020.

[8] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, 2019.

[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 1997.

[10] C. Hardy, E. L. Merrer, and B. Sericola, "MD-GAN: Multi-discriminator generative adversarial networks for distributed datasets," in *International Parallel and Distributed Processing Symposium*, 2019.

[11] Z. Cai, Z. Xiong, H. Xu, P. Wang, W. Li, and Y. Pan, "Generative adversarial networks: A survey toward private and secure applications," *ACM Computing Surveys (CSUR)*, 2021.

[12] R. Yonetani, T. Takahashi, A. Hashimoto, and Y. Ushiku, "Decentralized learning of generative adversarial networks from non-iid data," *arXiv*, 2019.

[13] Z. Ma, Y. Liu, Y. Miao, G. Xu, X. Liu, J. Ma, and R. H. Deng, "Flgan: Gan-based unbiased federatedlearning under non-iid settings," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[14] R. Guerraoui, A. Guirguis, A.-M. Kermarrec, and E. Le Merrer, "FeGAN: Scaling distributed GANs," in *Annual ACM/IFIP Middleware conference*, 2020.

[15] M. Rasouli, T. Sun, and R. Rajagopal, "Fedgan: Federated generative adversarial networks for distributed data," *arXiv*, 2020.

[16] E. Ekblom, E. L. Zec, and O. Mogren, "EFFGAN: Ensembles of fine-tuned federated GANs," *arXiv/2206.11682*, 2022.
[17] E. Brophy, Z. Wang, Q. She, and T. Ward, "Generative adversarial networks in time series: A systematic literature review," *ACM Computing Surveys*, 2023.
[18] W. Xu, N. Souly, and P. P. Brahma, "Reliability of gan generated data to train and validate perception systems for autonomous vehicles," in *2021 IEEE Winter Conference on Applications of Computer Vision Workshops (WACVW)*, 2021.
[19] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, 2018.
[20] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, 2019.
[21] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in *Advances in Neural Information Processing Systems*, 2014.
[22] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *IEEE international conference on computer vision*, 2017.
[23] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.
[24] J. Connor, R. Martin, and L. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, 1994.
[25] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.
[26] J. M. L. Alcaraz and N. Strodthoff, "Diffusion-based time series imputation and forecasting with structured state space models," *Transactions on Machine Learning Research*, 2022.
[27] X. Li, P. Ye, J. Li, Z. Liu, L. Cao, and F.-Y. Wang, "From features engineering to scenarios engineering for trustworthy ai: I&i, c&c, and v&v," *IEEE Intelligent Systems*, 2022.
[28] X. Li, Y. Tian, P. Ye, H. Duan, and F.-Y. Wang, "A novel scenarios engineering methodology for foundation models in metaverse," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

BASEM SHAKER a graduate in Mechanical Engineering from the University of Manitoba, furthered his expertise with dual M.Sc. degrees in Engineering Automation from KTH Royal Institute of Technology and University of Trento. As a professional with over six years of industry and academic experience, Basem specializes in Machine Learning, Data Science, and Automation.

GASTONE PIETRO ROSATI PAPINI graduated in Automation Engineering at the University of Pisa in 2012 and earned his Ph.D. at Sant'Anna School of Advanced Studies in Emerging Digital Technologies in 2016. He is currently an Associate Professor in advanced control systems applied to the field of autonomous driving and ADAS at the Department of Industrial Engineering, University of Trento. He is also the Co-Founder of Cheros s.r.l., a spinout company from the University of Pisa that specializes in renewable energies and ICT solutions. With over five years of experience in mechanics and control systems, he has recently focused on the application of machine learning techniques for modeling and controlling mechanical systems. Webpage: https://tonegas.com/

MATTEO SAVERIANO received his B.Sc. and M.Sc. degree in automatic control engineering from University of Naples, Italy, in 2008 and 2011, respectively. He received his Ph.D. from the Technical University of Munich in 2017. Currently, he is an assistant professor at the Department of Industrial Engineering (DII), University of Trento, Italy. Previously, he was an assistant professor at the University of Innsbruck and a post-doctoral researcher at the German Aerospace Center (DLR). He is an Associate Editor for RA-L and IJRR. He is the coordinator of the EU HE INVERSE project. His research activities include robot learning, human-robot interaction, and understanding and interpreting human activities. Webpage: https://matteosaveriano.weebly.com/

KUO-YUN LIANG received the M.Sc. degree in electrical engineering from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2011 and the industrial Ph.D. degree from the Department of Automatic Control and the ACCESS Linnaeus Centre, School of Electrical Engineering, KTH, and the Research and Development Department, Scania CV AB, Södertälje, Sweden, in 2016. He is a full-time Senior Data Scientist at the Connected Systems Department, Scania CV AB, Södertälje, Sweden. His main research interests include vehicle onboard anomaly detection, machine learning, federated learning, and cybersecurity.