FeTGAN: Federated Time-Series Generative Adversarial Network

1st Marcus Plesner, 2nd Lydia Y. Chen, 3rd Zilong Zhao, 3rd Aditya Kunar
1st Author, 2nd Responsible Professor, 3rd Supervisor
dept. Electrical Engineering, Mathematics and Computer Science
Technical University of Delft
Delft, Netherlands
{M.K.Plesner, A.Kunar}@student.tudelft.nl, {Y.Chen-10, Z.Zhao-8}@tudelft.nl

***Abstract*—The key to producing high-fidelity time-series data is to preserve temporal dynamics. This means that generated sequences respect the relationship between variables across time as in the original data. While new types of GANs have been used to generate time-series data, they, like previous GAN implementations, are time consuming to train. A novel federated framework is proposed, which generates realistic time-series data, by combining supervised and unsupervised training. The framework is based on the work in TimeGAN and Federated GAN (FeGAN). Using an embedded learning space, TimeGAN encourages the network to mimic the structure of the training data. FeGAN allows the results of TimeGAN to be combined at a central server, which has benefits for both throughput, and potential to improve data privacy. This also introduces the possibility of using cross domain data. The challenge with creating applying federated learning to TimeGAN, and time-series data in general is whether the learned temporal dynamics can be combined. This is accomplished by the combination of the weighting and sampling scheme used. This paper demonstrates, by qualitative and quantitative analysis, the ability novel framework proposed, to produce equivalent quality synthetic time-series data compared to the original TimeGAN, without sharing local data between nodes in the network. This is based on the predictive and discriminative scores described, as well as PCA and t-SNE analysis. Additionally, there is an approximate eleven percent increase in Floating Point Operations per second when using one machine, and up to a thirty percent increase when using multiple.**

I. Introduction

Generative adversarial networks (GAN)s are used to create high fidelity synthetic data for the purposes of training high dimensionality machine learning models and preserving differential privacy [1]. GANs have also been shown to perform well at tasks, such as text to image synthesis, drug discovery, and privacy maintenance. Drug discovery has become increasingly important in the recent past, and GANs show the ability to quickly generate new hypothesis and simultaneously test them. Privacy is another important field of study because GANs allow synthetic data to be shared with third-parties without compromising sensitive information.

A generative adversarial networks (GANs) is composed, in its most simple form, of two neural networks, a generator and discriminator. The generator produces fake data, and the discriminator tries to distinguish fake data from real data. The two then compete. This training strategy is interesting because it allows for quicker training with less human input and feedback.

Time-series data is particularly challenging, because the generator must preserve the temporal dynamics of the data. This is where TimeGAN's [2] latent feature space becomes important. This is because the temporal dynamics of the data are often driven by a few features. Therefore, an embedding network is used to both promote parameter efficiency, but also to facilitate the generator's learning of the temporal dynamics of the original data sample. This is the feature that makes TimeGAN uniquely suited to time series data.

TimeGAN introduced a latent feature space with lower dimensionality than the original data, in which data is generated. The data is embedded into, and recovered from this feature space using another set of components. This makes it easier for the generator to learn the structure of the data, due to the lower dimensionality. The federated learning paradigm has been used in conjunction with GANs as in the research behind Federated GAN (FeGAN) [3], which combined models from individually trained GANs on a central server. This turned out to be an improvement over the multi-discriminator GAN (MDGAN) [4], which had a central generator and multiple discriminators on different machines. The limitation of FeGAN is that it would not perform well on time-series data because the nodes were using Least Squares GAN (LSGAN) [5], which is not intended for time-series data.

The research question for this paper is: how can the federated learning paradigm be applied to GANs such that they can generate high fidelity time-series data, and accelerate training in a privacy preserving way . There are also questions about how GAN's should be trained, such as: which distance measurement produces the highest fidelity data, and is it the same for different types of data.

This paper will apply the federated learning paradigm to the generation of time-series data, as in TimeGAN [2], to generate synthetic time-series data, such as financial, web traffic, and weather data. It is tested whether individually trained models can be combined. The code for this paper uses loss based weighting to aggregate the data generating components of TimeGAN. The server implements a loss based weighting scheme. Each trainer in the system uses TimeGAN, and sends

the weights of the data generating components of TimeGAN to the server. Since TimeGAN is trained in three phases, there are three different losses used, the embedding loss, the supervised loss, and the unsupervised loss.

The process of federating TimeGAN to create Federated TimeGAN (FeTGAN) started with reproducing the original results of the TimeGAN paper. Because of some changes to the repository, reproducing the results required some communication with the original authors of TimeGAN. While no code from FeGAN was used in the final version of FeTGAN, reproducing the results did show how federated learning could be done.

To enable comparison between TimeGAN and FeTGAN, several experiments were performed. The first experiments were performed on the stock market and energy consumption datasets used in the original TimeGAN paper. These experiments compared TimeGAN to FeTGAN. TimeGAN and FeTGAN were also tested on a new dataset, which weather data from New York City over a period of eight years. The last experiment tested whether FeTGAN could combine the models from the workers, when the models were trained on completely separate data. The aim of this experiment was to show the utility of FeTGAN when training with cross-domain data.

II. Preliminary

Since this paper is based on combining the results of two previously written papers, this section discusses the two, namely TimeGAN and Federated GAN (FeGAN). The last experiment is conducted to study the effect off cross-domain data. [6]

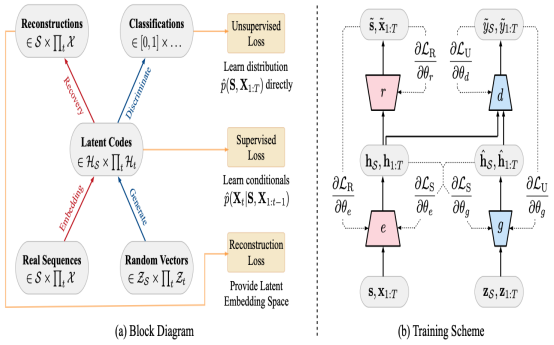
A. TimeGAN

TimeGAN is chosen for this research because it is composed of five distinct components, which can be shared with the server. It is also one of the first GANs, which produces high fidelity synthetic time-series data. Lastly, the TimeGAN code was well documented, and had already isolated the trainable variables for each component. This made integrating the model sharing functionality easier.

TimeGAN consists of two sets of components. The first set is the standard GAN generator and discriminator network. The second set is what makes TimeGAN unique, and it is the embedding component and recovery component. When combined, these two sets are the components of TimeGAN which make it uniquely suited to time-series data. The last component is the supervisor, which predicts the next sequence in the data given a sequence of data.

The generator in TimeGAN does not produce results directly in the feature space of the original data. The two produce and handle data in the latent space provided by the embedding function. This is crucial to improving parameter efficiency. A model of TimeGAN from the original paper is shown below. The model shows how components share their outputs, and the function of each loss.

Fig. 1. TimeGAN architecture



B. FeGAN

FeGAN is a federated GAN which takes LSGAN and runs it on separate nodes. The distinctive characteristics of FeGAN are its balancing and sampling methods. The balancing of the distributions on the separate nodes is done using the Kullback-Leibler divergence metric, which shows the divergence between two distributions. In FeGAN, this is calculated on the data distributions on each node, compared to the global data distribution. The second defining characteristic of FeGAN is the sampling method. Balanced sampling is used which means nodes with a more evenly distributed data distribution are sampled more often

FeGAN also demonstrates the performance benefits of federated learning, including being able to effectively sample models from up to 176 different devices. FeGAN also demonstrated an increase in throughput, which in their paper is measured as epochs per second, of up to five times over MDGAN. However, this increase in throughput only happens when there are more devices. With fewer devices, MDGAN has better throughput.

C. Cross Domain Experimentation

GANs have been used to create synthetic data in places where there is not enough real data available. This is the subject of image-to-image translation for rare-classes. This has been done for generating image data for rare species living in a given location. Previous research has demonstrated that using data from a different domain can improve the performance of the generator of a GAN. In the research, using images of common species of animals, enhanced the performance of the GAN, even when the goal was not to produce that species. For example, assume hogs and deer lived in the same forest. Furthermore, assume hogs are rare, but deer are common. If the goal is to generate synthetic images of hogs, one could include images of deer in the training data, and it would increase the performance when generating pictures of hogs.

The two main problems this paper addresses are the generation of synthetic time-series data, and increasing throughput in GAN training. The problem of synthesising time-series data is challenging because of the inherent temporal dynamics of the data. For example, a common form of time series data is stock

data. A stock may have a real data point at 10 euros, and one at 20 euros, however, those are unlike to come immediately after one another. Therefore the model must learn to generate the next data point based on the previous data point.

TimeGAN attempts to solve the temporal dynamics point by having an embedding and a recovery model. The embedding model transforms the data into a latent space with fewer dimensions. This makes it easier for the generator and discriminator to learn the structure of the data. The recovery function converts data from the latent lower dimension feature space into the original feature space.

Federated learning is proposed in the situation when throughput should be increase by using multiple machines. Federated learning is a learning technique where multiple nodes train machine learning models and aggregating them at a centralised server. The challenge is how to aggregate and how to sample. aggregation should be done in such a way that one bad node does not ruin the model, the model should not get stuck in mode collapse, and it should integrate the complexities of every node's model.

FeGAN solves the aggregation problem and the sampling problem using Kullback-Leibler weighting for the aggregation and balanced sampling for the sampling. Kullback-Leibler weighting is a type of weighting which prioritises updates from nodes based on their their local data's divergence from the global data distribution. Since this is not practical for time-series data, the divergence is calculated based on the divergence of the node's model from the server model. Balanced sampling samples nodes based on how balanced their data sample is. The more balanced the data sample is on the node, the more often it is sampled.

The way to combine these two is to have every node in a federated learning system running TimeGAN, and aggregating the results using FeGAN. Using Kullback-Leibler weighting does not make sense in this case because the local data distributions are intentionally very different. Therefore, loss based weighting is used. Since the dataset TimeGAN is on every node, balanced sampling is not relevant, since every node trains on the same dataset.

III. Federated TimeGAN (FeTGAN)

The architecture of FeTGAN is inspired by FeGAN. FeTGAN runs a server which aggregates the models and broadcasts the server model after each aggregation. FeTGAN runs a standalone GAN on each node, to keep private data off the server. Each node sends its model weights to the server after a user defined number of iterations, and receives the results of the server's aggregation after the server is done combining the models. Each node in FeTGAN is running TimeGAN, and the weights of each component of TimeGAN, can be aggregated separately.

There are three of the five components of TimeGAN that have to be shared with the server, in order to generate the synthetic data. The first is the generator. The generator generates data in the latent space, which is then recovered. The second component that is shared with the server is the supervisor. The supervisor takes as input the data from the generator, and creates the next sequence of data. The last component used for generating synthetic data is the recovery The data from the generator and supervisor are then fed into the recovery function. In the implementation, every component of TimeGAN is shared with the server, but this is not necessary to generate synthetic data. To generate Synthetic data, only the generator, supervisor, and recovery are needed.

All the weights are sent to the server to be aggregated. In addition to the weights, the losses are also sent to the server for the server to apply weighting to the different models. In the first phase of training, which is the embedding and recovery training, the embedding loss is shared. This is the loss when data in the original space is converted to the latent space and back again.

During the supervised phase of training the supervised loss is shared. The supervised loss is derived by giving the supervisor data that has been embedded into the latent space, and comparing the sequence returned by the supervisor to the actual data in the latent space. The the loss sent when aggregating the generator models is the unsupervised generator loss, which is given by the trained discriminator. When aggregating the models of the different components, the weight they are given is inversely proportional to the loss they report. Therefore, the lower the error a model has, the more weight it has. This ensures the best model has the most influence, but the less able models still count.

A primary issue with GANs is they take long to train. The solution to this is to use more computation resources. One way to do this is to scale horizontally using federated learning. This paper aims to integrate the federated paradigm with the research on TimeGAN. The idea is to increase the iterations performed per second, while creating high fidelity data.

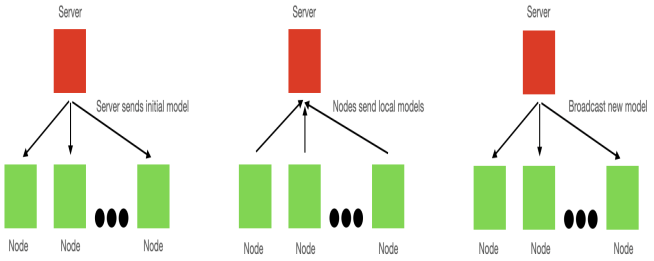
The implementationof FeTGAN is a parameter server architecture where the workers run TimeGAN and the server aggregates the result. The results are aggregated on the server using loss based weighting. While FeGAN used Kullback-Leibler (KL) weighting, which is based on the divergence between two distributions, this is not as suitable as loss based weighting. The KL weighting in FeGAN is calculated based on the divergence of the data distributions on the nodes from the global distribution. This does not make sense in the context of time-series data.

IV. Experimental Setup and Results

A. Setup

The setup is based on the batch update parameter server example provided by torch. There is a single server which aggregates the results, and each worker runs TimeGAN. FeTGAN can be run with a single node on one machine, which is exactly the same as the original TimeGAN, it can be used with multiple workers on one machine, or multiple workers on separate machines, as long as the server IP address

Fig. 2. Server broadcast model.



is known. The implementation is based on the parameter server architecture demonstrated by pytorch, using the Remote Procedure Call (RPC) framework.

B. Data

The experiment was conducted on five datasets. One dataset containing Google's historical stock data on a one day time frame. The features of the data are the volume, high, low, opening, closing, and adjusted closing prices. Stock prices are continuous-valued, but aperiodic. Features of stock data are also correlated with each other. This makes stock data ideal for testing a model's ability to generate time-series data.

Time-series data cannot be split randomly, since data points are dependent on previous data. This means there must be a continuous sample of data on each node. If this is not the case, there is no connection between the data sequences, which makes it impossible to learn the temporal dynamics. Therefore, each node must have a complete sequence. In the case where there are two nodes, one node will have the first half of the data, and the second node will have the second half. This means there is no overlap between the dataset on each node in these experiments.

The second dataset is shows energy usage statistics. This is a much larger dataset, and it has a lot more features.

There are three weather datasets, which include features such as temperature, humidity, wind speed, and wind direction. The three locations for these datasets are New York City NY, Charlotte SC, and Jacksonville FL. The first dataset is used in a way similar to the stock and energy dataset. The second two are used for the last experiment, which combines models from two nodes, one with the Charlotte dataset and one with the Jacksonville dataset.

All the data is cut by a sequence length of 24 and permuted, to make it as similar to independent and identically distributed data. This sequence length is a hyper-parameter can be changed, and the effects of changing it can affect the results.

The experiment setup is shown in figure 1. The server broadcasts its own model to the clients to initialize all the models. The clients then run TimeGAN, and send the intermediate results to the server every 100 iterations. The server computes the average model using loss based weighting, as described earlier.

C. One Node (Centralized)

This experiment is a reproduction of the original TimeGAN paper, to verify the results and enable comparisons. In this test, the single node running TimeGAN has access to the entire dataset. This test is still conducted with the federated architecture described previously, except there is only one node sending weights to the server.

Reproducing results from the original TimeGAN paper, and associated code turned out to be a challenge. The original authors of TimeGAN had modified their implementation to make use of Tensorflow 2. However, this had not been completed successfully. After contacting the authors, the changes were eventually reverted. This immediately fixed the problem. Jan-Mark Dannenberg, who was also working with TimeGAN at the time has made efforts to update the code to Tensorflow 2 successfully.

D. Two Nodes

This is the first test of the aggregation system used. In this experiment there are two nodes, each running TimeGAN, and a server averaging the models sent in by the two nodes. As mentioned earlier, the data cannot be randomly split, because of the nature of time-series data, so in this experiment the first node has the first half of the data and the second node has the second half of the data.

E. Eight Nodes

This is the test of whether FeTGAN can truly perform as a federated learning system. The server has to aggregate the models from eight different nodes, and capture the unique properties in each dataset. Furthermore, each node only has a sliver of the data, an eighth of the data exactly. This creates a huge challenge because of the short time frame of each dataset, and learning patterns across larger time frames. This is not an issue for the weather data sets, since they are very large. However, when using the stock dataset, which is a fraction of the size, it can become a problem.

F. Location Based Data Partition

This is a test of how well FeTGAN can learn the temporal dynamics of two distinct datasets. The previous tests have involved splitting a dataset into time chunks, and training one node per chunk. This experiment is different. It takes two datasets on the same time frame, one weather data set from Charlotte SC, and one from Jacksonville FL. One worker has the Charlotte data, one has the Jacksonville data. FeTGAN is then run as normal and the models are combined. The test is to see whether the aggregated model can generate synthetic data for both Charlotte and Jacksonville.

G. Evaluation

The three analysis tools used to analyse the results are as follows: t-SNE and PCA analyses, a discriminative score, and a predictive score. The t-SNE and PCA analyses are done on both the synthetic and original datasets, which shows how the

generated samples resemble the original in a two dimensional space.

The discriminative score is a time-series classification model, which distinguishes between sequences from the original and generated datasets. First the original and synthetic data are labeled real or fake respectively, then an off-the-shelf classifier is trained to distinguish the two as a supervised task. The discriminative score is the classification error. The predictive score uses a sequence prediction model used to predict the next-step temporal vectors over an input sequence. The trained model is then evaluated on the original dataset. The performance is measure by mean absolute error.

H. Results

As can be seen in Figure 3, the PCA and t-SNE analysis shows that the synthetic data is spreads out more and more as more workers are added. This is seen most clearly in the PCA analysis, where the left side of the graph in the central experiment is in a thin line, but with eight workers, it resembles a cluster, which is more similar to the original data. The t-SNE analysis reveals the same, since in the centralised case there is very little deviation from the pattern. In the eight worker case, the synthetic data is more noisy, which arguably better represents the data.

Fig. 3. PCA and t-SNE analysis of synthetic and original stock data

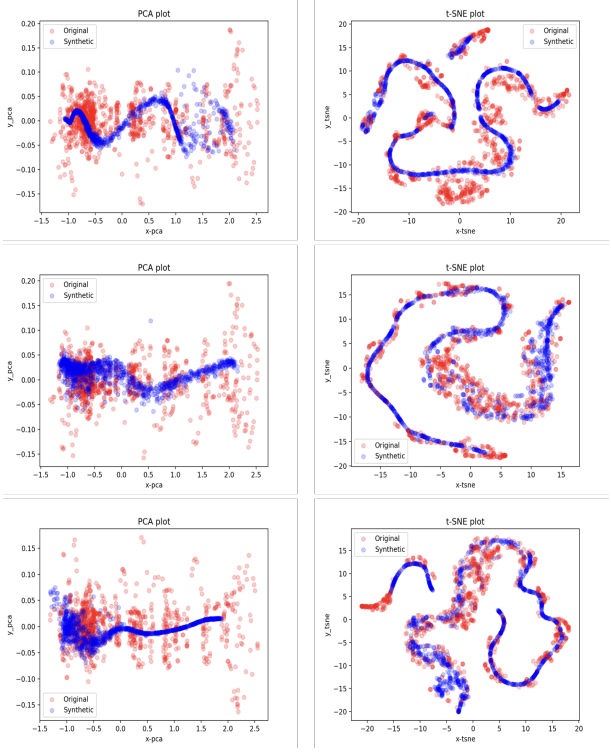


Fig. 4. Discriminative and Predictive scores on the stock data set

Stock Scores

	Centralised	Two Workers	Eight Workers
Discriminative	0,12285	0,14536	0,16834
Predictive	0,04184	0,037822	0,38169

The synthetic energy data has the most deviation from the original data. It is clear to see both by the visual analysis and the quantitative analysis that the two worker case performed the best. The eight worker case achieved similar predictive and discriminative scores compared to the centralised case, but the visual analysis shows lower quality data. The this was the case in which both FeTGAN and TimeGAN performed the worst, but the improvement made by having two nodes is curious.

The energy dataset is the only dataset for which the hyper parameters had to be changed to achieve good results. In this model, all components use five layer recurrent network. In every other model, three layers were used. The number of layers was found to be the one which affected the performance the most.

Fig. 5. PCA and t-SNE analysis of synthetic and original energy data

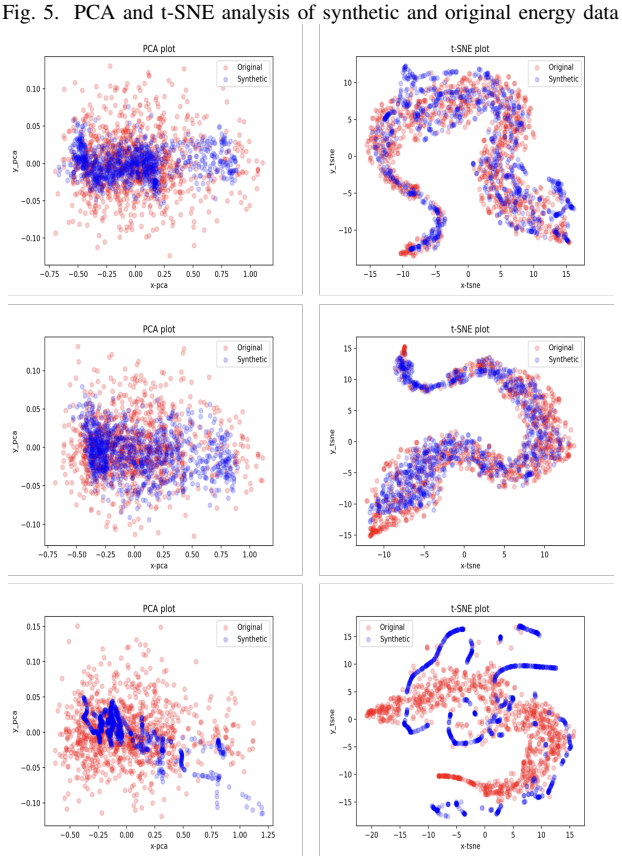


Fig. 6. Discriminative and Predictive scores on the energy data set

Energy Scores

	Centralised	Two Workers	Eight Workers
Discriminative	0.46051	0.21746	0.44142
Predictive	0.29553	0.28481	0.33441

The New York City weather data set is marred by an outlier both for the centralised and two node case, which makes the visual analysis more difficult. However, all of the PCA analysis, and the eight worker t-SNE analysis shows the ability of FeTGAN to produce high fidelity data.

Fig. 7. PCA and t-SNE analysis of synthetic and original New York weather data

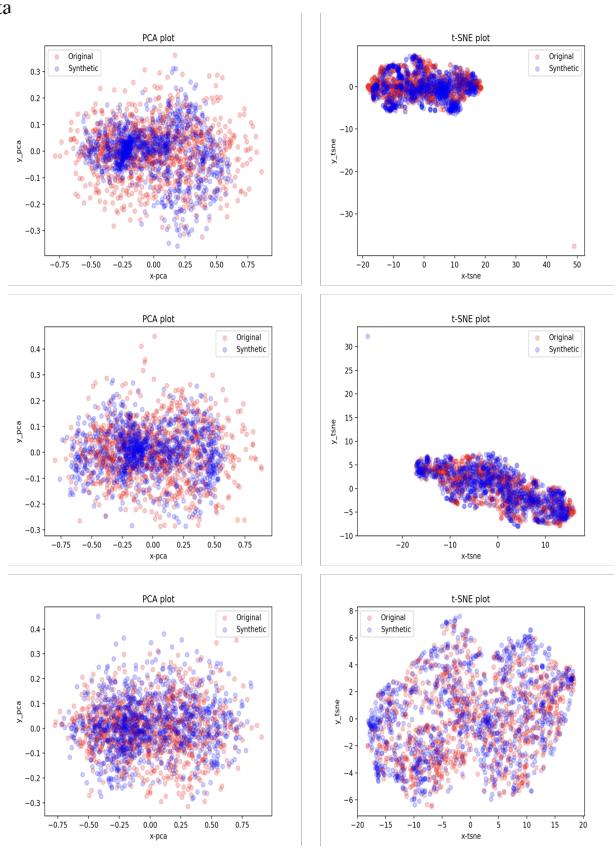


Fig. 8. Discriminative and Predictive scores on the weather data set

Weather Scores

	Centralised	Two Workers	Eight Workers
Discriminative	0.36865	0.36154	0.35472
Predictive	0.26875	0.25357	0.25210

The discriminative and predictive scores of the different experiments can be seen in the tables. It shows that the discriminative and predictive scores vary only slightly. For the stock dataset, since it is rather small, it is understandable that as more nodes are added, the lower the amount of data on each node, the less able the node is to learn the distribution of the data. The energy dataset yielded the most interesting scores because of the divergence of the two node experiment. So far, there has not been a plausible explanation for this. The weather dataset is by far the largest dataset, which seems to create better scores with more workers. One possible explanation for this is that each worker can learn the nuances in its own dataset.

I. Location Based Data Partition

This is the most interesting experiment because it demonstrates the flexibility of FeTGAN, and its ability to integrate meta data. The dataset includes the longitude and latitude, which could be aiding the model in generating data for the different locations. It should be noted, that the two areas have similar climates, since they are both coastal cities on the east coast. The tables below show that for both Charlotte and Jacksonville the model, which is composed of individual models can produce high fidelity synthetic data. In the ”aggregated” columns in the tables below, they are clearly very close to models trained just for that individual location

Fig. 9. Discriminative and Predictive scores for Charlotte for central & aggregated models

Weather scores Charlotte

	Centralised	Aggregated
Discriminative	0.41439	0.30593
Predictive	0.30675	0.25921

Fig. 10. Discriminative and Predictive scores for Jacksonville for central & aggregated models

Weather scores Jacksonville

	Centralised	Aggregated
Discriminative	0.32497	0.29115
Predictive	0.27293	0.28251

Notice the improvement in the aggregation scores on the Charlotte SC dataset. There could be several reasons for this. One is simply that a model that is trained for 10,000 iterations on two machines is trained for a total of 20,000 iterations. This could make it perform better when compared to a model trained for 10,000 iterations on one machine. This counter the expected results that since there is data for another location, which does not have exactly the same climate, the model should perform worse when aggregated.

The second, and more interesting, explanation is that the mixing of data from another location actually harmed the model slightly. However, the benefit gained could be a result of the aggregated model learning general weather patterns better, because it has been exposed to more of them. By combining the two models it is possible that the aggregated model was far better at generating data which conforms to regular patterns. An important component of this hypothesis is that Charlotte and Jacksonville experience similar climates. In the future, it should be tested whether this could be true when combining data from more dissimilar locations.

Below are the PCA and t-SNE analyses comparing the data generated by the aggregated model to real Charlotte data, and real Jacksonville data. The analysis shows that the model manages to produce high fidelity data for both locations. The first row of both figures is generated by TimeGAN, with the data for a single location. The second row is data generated by FeTGAN, with data for both Charlotte and Jacksonville.

Fig. 11. PCA and t-SNE analysis of Charlotte weather Data

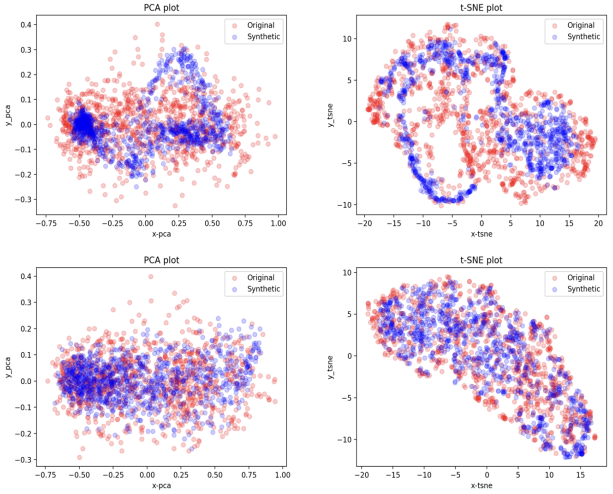
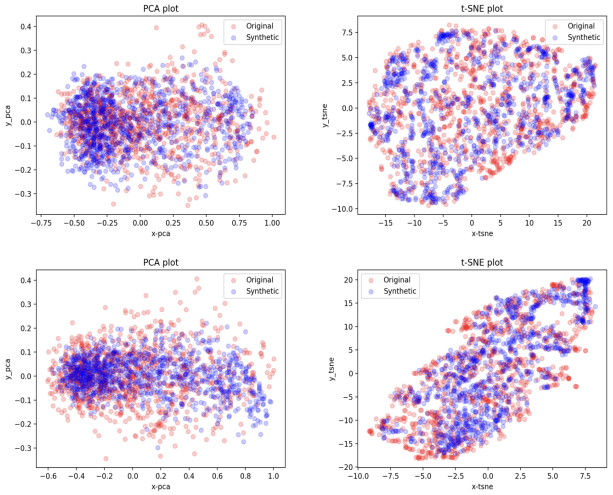


Fig. 12. PCA and t-SNE analysis of Jacksonville weather Data



Federated TimeGAN also increases throughput, when compared to the centralized version of TimeGAN. Throughput is defined as the number of times a pass of all the data is done per second. TimeGAN has three stages of training, the embedding stage, the supervised loss stage, and the joint training stage. The only time the components used to generate data are trained is during the joint training, so that is when the models begin to be shared.

V. Responsible Research

Ethics are extremely important in the field of GAN research. One of the foremost ethical questions is that of deepfakes. A deepfake is a form of synthetic media in which a person in an existing image or video is replaced with someone else's likeness. This can be used nefariously in the form of blackmail, pornography, and politics. Therefore, GANs must be used responsibly.

This is especially important in the age of social media, in which a piece of content produced using a GAN could easily be spread fast enough to ruin someone's future career in a matter of hours. Furthermore, content produced by GANs can

The second important part of GAN research is data privacy. Federated learning is a step forward in data privacy because each node, which could be an individual user, can share a trained GAN model, without sending the data to a centralised server. Since some information could be extracted from the model, a further improvement could be to use Differential Privacy GAN on each node.

VI. Discussion

To see if Federated TimeGAN offers any improvements, it should be compared with the original version of TimeGAN in its ability to generate data, and it can be compared with FeGAN to determine if there is a comparable and significant increase in throughput. As seen in Figures 3 to 11, the original TimeGAN performs similarly to FeTGAN. It should be noted that this is dependent on the dataset.

To compare FeTGAN to FeGAN, it is necessary to calculate throughput. Throughput in this case, is measured in FLOPS per second. This is measured by computing the FLOPS required by each pass of the TimeGAN training, multiplying that number by the number of nodes, and the number of iterations, and then dividing the whole result by the time taken. It can be seen by the figures below, that there is a greater marginal increase per node when the nodes are on different computers. It should be noted that in figure 15, there are three machines used for the two worker and three worker experiment, since this was the maximum available at the time. The two worker case features the server on one machine, and a worker on each of the other two. The eight worker experiment features the server on one machine, and four workers on each of the other two.

With regard to FeGAN, the most important comparison is throughput. FeTGAN shows an increase in throughput when compared to the original TimeGAN. This can be for a couple of reasons. The most compelling reason for this is the splitting

Fig. 13. Change in throughput with FeTGAN on a single machine

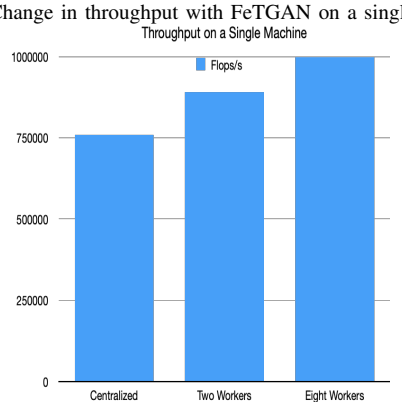
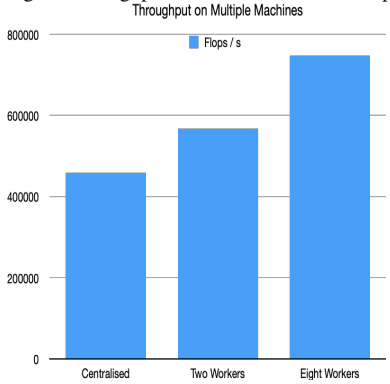


Fig. 14. Change in throughput with FeTGAN on multiple machines



of the dataset, which makes calculations faster since each node is handling a smaller dataset.

As expected, when running FeTGAN across more machines, the increase in throughput is greater than when running FeTGAN with more workers on one machine. The increase in throughput between two and eight workers in the multiple machines case is thirty two percent. In the single machine case, there is an increase of only twelve percent. This is encouraging, and suggests that running even more machines would have a large effect on the throughput.

These experiments are conducted on stock data to enable comparisons between FeTGAN and TimeGAN. Stock data is available publicly, so it does not make sense to use this implementation because of privacy concerns in practice. On the stock dataset, it also does not appear to offer any benefits in terms of performance. Using FeTGAN does, however, make sense with other datasets in which data privacy is of greater concern. This includes, but is not limited to: energy consumption, typing patterns, and internet traffic. FeTGAN can also be used to increase the amount of data available because data that is not directly related may still offer performance increases as seen with the Charlotte experiment.

VII. Conclusions and Further Work

In this paper, FeTGAN is introduced as an extension of the original TimeGAN implementation, but adapted for federated learning using parts of FeGAN. This combination leverages the embedding and recovery network developed in TimeGAN to create high fidelity time-series data, while using part of the aggregation process described in FeGAN. Federated TimeGAN demonstrates significant throughput improvements in generating high fidelity time-series data. The splitting of data by location also demonstrates the robustness of FeTGAN, and show it can be used in more real life federated learning situations.

Further work should seek to integrate differential privacy, and improve the currently naive sampling used in FeTGAN. Furthermore, a weighting scheme that is more appropriate for time-series data should be found. Future research should also be directed at finding a sampling method for Federated TimeGAN.

Further work should also be done to identify in which situations a GAN can be aided by data that is not necessarily related. For example, the question of generating data for rare classes of data is one that could particularly benefit from other data.

References

[1] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. Technical report, Computer Science and Engineering, Michigan State University and Department of Computer Science, Rutgers University and Department of Healthcare Policy and Research, Weill Cornell Medical School, 2018.

[2] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. Technical report, University of Cambridge and University of California Los Angeles, 2019.

[3] Rachid Guerraoui, Arsany Guirguis, Anne-Marie Kermarrec, and Erwan Le Merrer. Fegan: Scaling distributed gans. Technical report, EPFL and Univ Rennes, Inria CNRS Irisa, 2021.

[4] Yotam Intrator, Gilad Katz, and Asaf Shabtai. Mdgan: Boosting anomaly detection using multi-discriminator generative adversarial networks. Technical report, Department of Software and Information Systems Engineering Ben-Gurion University of the Negev, 2018.

[5] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. Technical report, Department of Computer Science, City University of Hong Kong, and Department of Mathematics and Information Technology, The Education University of Hong Kong, and Department of Information Systems, City University of Hong Kong, and Center for Optical Imagery Analysis and Learning, Northwestern Polytechnical University, and CodeHatch Corp., 2017.

[6] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. Technical report, International Conference on Machine Learning, pages 1857–1865, 2017.