Towards synthethic generation of mobility data with TimeGAN

Kjell Binici and Sophie Schaible

HAW Hamburg, Berliner Tor 7, 20099 Hamburg, Germany
kjell.binici@haw-hamburg.de
sophie.schaible@haw-hamburg.de

Abstract. In an era defined by the centrality of data in technological progress mobility data, that describe the movement patterns of humans, is becoming more important, helping in areas like city planning and transportation. However, there is a challenge in acquiring extensive, real-world mobility data, which is crucial for companies leveraging predictive analytics and machine learning. To solve this problem, we can synthetically generate gps data using Generative Adversarial Networks (GANs). This research explores the potential of a GAN model tailored for time-series data, called TimeGAN, for this specific purpose. We explain the foundational principles of its architecture as well as conduct and evaluate a series of empirical experiments. Finally we answer the question how suitable the model appears to be for the task of generating synthetic mobility data. In summary our results show that TimeGAN poses various challenges when dealing with two-dimensional gps data.

Keywords: TimeGAN · Mobility Data · Synthetic Data Generation.

1 Introduction

In today's digital age, data stands as the cornerstone of many technological advancements. Specifically, mobility data — information that describes the movement patterns of entities, typically in the form of routes — has gained increasing attention, offering numerous insights for areas ranging from urban planning to logistics. For companies relying on predictive analytics and machine learning models, having a robust dataset of routes is essential. However, not every entity is privileged to have access to vast amounts of mobility data, which presents a challenge when attempting to build effective, high-performing models. As a solution to this limitation, the synthetic generation of mobility data emerges as a promising avenue.

Synthetic data generation, the act of producing artificial data that mirrors the properties of real-world data, has witnessed advancements with the introduction of Generative Adversarial Networks (GANs). Amongst various models, TimeGAN has become particularly relevant for the generation of time-series data. Given its architecture, which is specially made for time-series, it stands to reason that TimeGAN could be used for synthetic mobility data generation.

This paper is structured as follows. It begins with Section 2, where we review the related work, setting the foundation by discussing existing literature on synthetic data generation with a special focus on mobility data. In Section 3, we delve into the exploration of TimeGAN, covering its unique architecture and training schema, alongside a foundational overview of Generative Adversarial Networks (GANs) and their application to time-series data. Section 4 is dedicated to our experimental approach, detailing the setup, data processing methods, and the evaluation techniques employed. The findings and their implications are thoroughly presented in Section 5, where we explore various analyses and comparisons to evaluate the effectiveness of our approach. The paper closes in Section 6 with a conclusion and outlook, summarizing our key discoveries.

2 Related Work

In this paper we use TimeGAN, a special version of GAN to generate time-series data, which was introduced by Yoon et al. in 2019 [18] and demonstrated with sine, stock and energy data [17]. There are several other works that deploy TimeGAN to generate and augment data when there is a lack of sufficient and high quality data, or the need for privacy-preserving data, e.g. [2, 11, 20]. However, the main difference between these works and ours is that their input and output data are one-dimensional, while ours are two-dimensional consisting of latitude and longitude. A recent work argues for enhancing TimeGAN to augment data in two dimensions, one being the time dimension and the other one the feature dimension [10]. From that point of view our data is three-dimensional, as the time dimension is indispensable in time-series data. To the best of our knowledge at this point there is no work yet that uses TimeGAN to generate data where the features are more than one-dimensional.

However there are papers that cover the generation of two-dimensional mobility data with other types of machine learning including neural networks and in particular GANs. In 2018 Sapre et al. [14] generated urban traffic data with the k-nearest-neighbor (kNN) algorithm and statistical approaches motivated by avoiding the high privacy and security risks associated with tracking an individual's movements. Their input data contained no time dimension but aggregated GPS paths provided by volunteering citizens with no separation between individual traces recordings. They concluded their model is effective in modeling real-world traffic as their simulated results of traffic density on several junctions correlated with the predictions of Google Maps.

Recently, there has been significant activity in the field of generating data with Deep Neural Networks (DNNs). In 2022 Gao et al. [6] analyzed what has been done so far with GANs in the spatio-temporal (ST) data domain, i.e. data that contains time and space features. The authors argue that Non-GAN Networks, in particular Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Autoencoder (AE) and Graph Convoluation Networks, are usually integrated in GAN architectures for ST data. Analogously, TimeGAN, which is used in this work, appends an Autoencoder Network and uses Gated

recurrent units (GRUs) or optionally Long Short Term Memory (LSTM) layers in its GAN component. Gao et al. divide the deployment of GANs with ST data in subcategories as shown in Figure 1. Our work can be categorized in the data type Time Series and the task Generation as it aims to generate new data based on time-dependent input and the chosen model is focused on the time aspect. Besides Musical, Medical, Stock and other data, only 1 out of 15 analyzed works in the category Time Series also deals with traffic flow and movement data and is called NAOMI [19]. However their goal is Imputation, i.e. a technique to fill null values appropriately. There is no work covered in the work of Gao et al. that combines Time Series data related to mobility and the task of Generation. Although TimeGAN is mentioned it is only regarded as model for sines, stocks, energy and events data. Another possibility is to categorize our work into the ST data type Trajectories, as all of the considered papers relate to mobility. However, the only task mentioned for this category is Prediction, which involves predicting future trajectories based on existing ones. This is not the main goal of our work and is used only as an evaluation method. 5 of the 9 mentioned works in the Trajectory category use GAN and LSTM in combination, the other 4 use individual combinations of GAN with LSTM, GRU, CNN and/or AE.

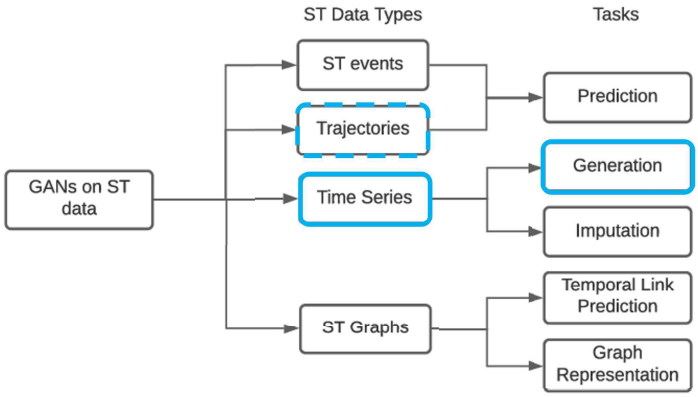


Fig. 1: Taxonomy of GANs on spatio-temporal (ST) data mining and modelling tasks [6] with categorization of this work in blue.

Additionally there are recent works that concentrate particularly on synthetic generation of mobility data. Berke et al. [3] use an RNN including hidden LSTM layers to synthetically generate realistic mobility data again motivated by protecting user privacy and augmenting data. They trained their model with a "location based services (LBS) dataset, that generates data representing individuals' mobility over a 5-day workweek" [3, p. 964] and additionally inferred the home and work locations of each individual. In the generation process the model only takes a home and a work position and based on that produces realistic location data for multiple days or weeks. Mauro et al. [13] aim to generate

deceptively real urban mobility networks with their own new kind of Neural Network architecture called moGAN (mobilityGAN). For this purpose, the mobility network is modeled as a graph. The models generator and discriminator are both CNNs and trained with data from trips with taxis and bikes. Here the focus is not on individual traces but on an overall network, e.g. all bike stations in Manhattan or a daily movement density network in one city. The approach most similar to ours is a Master thesis written by Alhasan in 2022 [1]. Nevertheless, one of the main differences is the restriction of the input data - while Alhasan uses a very narrow region and vehicle type (only truck driver movement inside Stockholm), we cover data independent from means of transportation all over Germany and sometimes further. Another substantial difference is the choice of DNN architecture that was experimented with. While Alhasan uses a LSTM model and a GAN made of two RNN networks the goal of this work is to evaluate the effectiveness of TimeGAN for the purpose of generating route data of individuals.

3 TimeGAN

Time-series data has found application in many fields including finance, healthcare, energy, and meteorology. While there are various models for time-series forecasting and analysis, the introduction of Generative Adversarial Networks (GANs) in the domain has brought a revolutionary change. TimeGAN, in particular, focuses on generating synthetic time-series data. This chapter aims to delve deep into the architecture and training schema of TimeGAN.

3.1 Brief overview of Generative Adversarial Networks (GANs)

Generative Adversarial Networks, often abbreviated as GANs, were introduced by Ian Goodfellow and his collaborators in 2014 [7]. The core idea is as follows:

- Generator (G): This network aims to generate data.
- Discriminator (D): This network tries to distinguish between real and generated data.

The two networks play a continuous game, where the generator tries to produce data to fool the discriminator, and the discriminator tries to get better at distinguishing real data from fake. As they train together, both get better at their tasks. The equilibrium (optimum) of this game is reached when the generator produces data almost indistinguishable from real data, making the discriminator's task as hard as guessing at random.

3.2 Time-Series Data

Time-series data, represented as a sequence of data points ordered by successive time points, holds significant relevance in various domains such as finance,

healthcare, and meteorology. Modeling such data requires capturing both the distribution of features at each time point and the dynamic interdependencies across time steps. TimeGAN, as introduced by Yoon et al. in 2019 [18], emerges as a specialized Generative Adversarial Network (GAN) tailored for time-series data generation.

A time series consists of observations at consecutive time intervals. For every time point $t \in T$, there's a specific observation, potentially composed of multiple features. To model time-series data effectively:

- The model should capture the feature distribution at each time point.
- The model should grasp the potential intricate dynamics of features over time.

Given the scale, dimensionality, and distribution of training data, achieving these objectives can be challenging for a standard GAN framework.

3.3 Architecture of TimeGAN

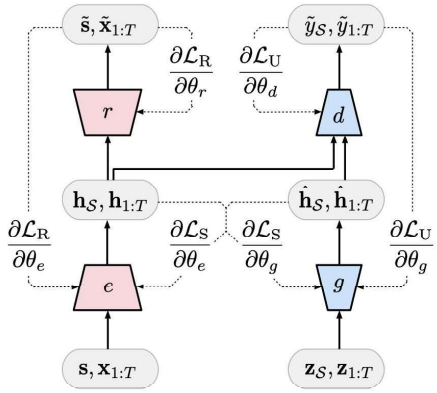


Fig. 2: The training structure of TimeGAN [18]

TimeGAN is crafted with a unique architecture to address the aforementioned challenges. At its core, it consists of four primary components:

1. **Embedder**: Transforms the original time-series data into a latent space, essentially compressing the multivariate features and relationships of the time series.
2. **Recovery**: This module operates inversely to the embedder, reconstructing the time-series data from the latent representation.
3. **Generator**: Analogous to traditional GANs but leverages the latent representation provided by the embedder to produce synthetic time-series data.
4. **Discriminator**: Discriminates between real and synthetic time-series sequences. In contrast to conventional GANs, the discriminator in TimeGAN takes into account both the original and latent space representation.

The union of the embedder and recovery functions forms an autoencoder. This autoencoder, in collaboration with the GAN components, ensures the model's capability to generate meaningful time-series data while respecting the data's inherent dynamics.

Figure 2 schematically illustrates the TimeGAN's training structure. The left side showcases the autoencoder components, whereas the right focuses on the GAN components. Solid arrows indicate forward propagation of data, while dotted arrows represent back-propagation of gradients.

3.4 Training Schema of TimeGAN

The training of TimeGAN is more complex than the straightforward application of traditional GAN principles to time-series data. It has been adapted and augmented to accommodate the unique characteristics and challenges of time-series data. The process is broken down into various steps, each tailored to ensure that the synthetic time-series data not only looks real at a single point in time but also has coherent and plausible dynamics over time.

1. **Auto-encoding Training**:
   - The initial phase of training focuses on the embedder and recovery networks.
   - The primary goal here is to ensure that there is a seamless transformation between the original data and its representation in the latent space. In other words, it's vital that no significant information is lost when translating the data back and forth between these two domains.
2. **Supervised Training**:
   - This phase is essential to ensure the generated time-series data has coherent temporal dynamics.
   - The generator is trained, not just to produce data that looks real, but also to anticipate the subsequent representation in the latent space. This aspect is pivotal for the generation of sequences that have logical temporal progression.
3. **Adversarial Training**:
   - This phase draws parallels with the training process of traditional GANs.
   - The generator produces synthetic time-series sequences, while the discriminator's task is to distinguish these generated sequences from actual data. This adversarial process ensures that the generated data becomes increasingly indistinguishable from real data.
4. **Joint Training**:
   - This is a holistic phase in the training process.
   - All components of the network—embedder, recovery, generator, and discriminator—are trained in tandem. By doing so, the performance of each component can be optimized in the context of the others, leading to a more robust and consistent TimeGAN model.

4 Experiments

This chapter deals with our empirical experiments about deploying TimeGAN on mobility data.

4.1 Data Preprocessing

Creating Routes The basis of our experiments is mobility data as introduced in [15]. It contains locations from probands living in Germany that were given a GPS recorder to measure mobility for 7 and 14 days respectively. The dataset is limited to the latest available year of 2021 with 2.500 probands. As the goal of the experiment is to generate individual routes of high quality the data was preprocessed. Based on the work from [1] and modified to fit our own purposes the following rules are applied:

1. Basic cleaning: Only keeping necessary columns (hubId (to identify the proband), time, latitude and longitude) and dropping duplicate rows. There are no null values in the remaining data set.
2. Clean outliers based on speed: At each data point the proband must have a speed of higher than 1 km/h and lower than 180 km/h.
3. Split up the data based on inactive time: After a proband did not move for 60 min. a new route starts.
4. Filter based on route length: Each route must contain at least 500 data points (i.e. GPS locations).
5. Filter based on distance travelled: The proband must have travelled at least 10km in total for each route.
6. Filter based on average time between consecutive points: The average time per route that a proband needs from one to a consecutive location must be less than 20min (to avoid heavily chopped routes).
7. Filter based on maximum time between consecutive points: The maximum time per route that a proband needs from one to a consecutive location must be less than 60min (to avoid heavily chopped routes).

(a) fourteen subroutes (b) four subroutes (c) one subroute

Fig. 3: Sample routes after preprocessing. Each route gets cropped in subroutes with equal sizes of 500 points. Each color indicates one subroute.

Routes that do not fit the rules are dropped. In total there remain 604 routes for model training. Table 1 shows a summary of the characteristics of these routes. On average each of the routes has 886 gps points (699 as median) leading up to 7,873 points on the longest route. The routes take about 82 min. on average (66 min. as median) with the longest route taking 8 h 5 min. and the shortest taking 26 min. while on average covering 54 km (median 37 km) but at least 6 km and maximum 657 km. Generally the probands move at a speed around 59 km/h, but not slower than 1 km/h and not faster than 180 km/h as limited beforehand. The variations between average and median values in datapoints, time and distance could be explained because we limit these variables only downwards, i.e. at least 500 data points, no inactivity of more than 60 minutes and at least 10 km in total. But the shift does not have to be considered any further because next the routes get cut in equally sized subroutes of 500 data points each anyway. Figure 3 shows three sample routes and their cuts. This results in 858 subroutes.

Table 1: Statistics about the chosen routes for training.

Measure	Value
routes (total count)	**604**
datapoints_avg (count)	886
datapoints_median (count)	699
datapoints_min (count)	500
datapoints_max (count)	7,873
time_avg (min.)	82
time_median (min.)	66
time_min (min.)	26
time_max (min.)	485
distance_avg (km)	54
distance_median (km)	37
distance_min (km)	6
distance_max (km)	657
speed_avg (km/h)	57
speed_median (km/h)	59
speed_min (km/h)	1
speed_max (km/h)	180

Creating Sliding Windows Furthermore, the data is sliced into sliding windows as it is done in the original TimeGAN experiments [18, 17]. This results in the structure of a four-dimensional list of dimensions $(w, s, r, 2)$ as shown in Figure 4. Each field represents one gps point (P) of one route (ID) consisting of a two-dimensional latitude-longitude-tuple. The following parameters are set as default (for some experiments the values are adjusted as stated later in chapter 4.4):

- count of routes $r = 858$
- count of gps points per route $p = 500$
- sequence length $s = 100$
- count of windows $w = p - s = 400$

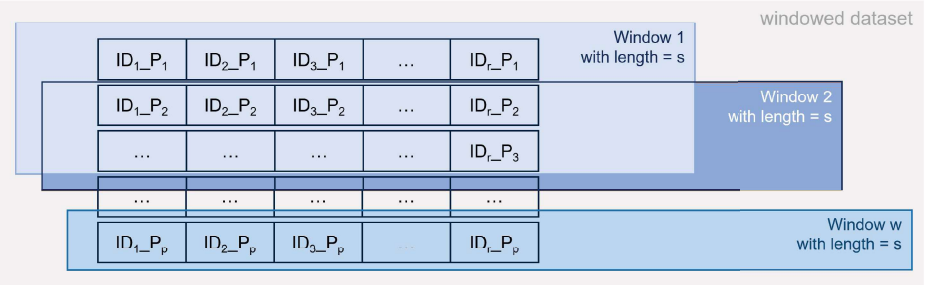


Fig. 4: Structure of the windowed training data.

The structure of the generated output data is the same but interpreted differently due to the fact that the model produces several variations for each route. The windows do not overlap, i.e. they do not hold equal points several times. In the output data each window holds one variation (V) of each route (see Figure 5). The number of windows, and thus the number of variations per route, is determined by the batch size b that is used during training. Consequently the output data has the dimensions $(b, s, r, 2)$.

Fig. 5: Structure of the output data.

Normalization Before the location data is fed into the model for training, it gets normalized. The coordinates get re-scaled to values between 0 and 1 based on certain min and max values for latitude and longitude. This is meant to improve the results of the model. Two different approaches were implemented. The first one is inspired by the work of Alhasan [1]. The min and max are set to fixed values (fixed normalisation). In our case they are derived from the most south, north, east and west points of Germany. The second approach is to set the values based on the input data (dynamic normalisation). Here the algorithm searches for the minimum and maximum longitude and latitude in the training data. In both scenarios the other points get rescaled accordingly. At the generation stage the produced data is rescaled to original coordinate values again.

4.2 Architectures

Baseline Architecture In this chapter, we present the baseline architecture of the synthetic sequential data generation system. This foundational architecture sets the stage for subsequent variations, emphasizing the common characteristics, model definitions, training methodologies, and loss functions employed in both versions.

The architecture comprises of several key components: the Embedder, Recovery, Generator, Discriminator, and Supervisor. Each of these plays a specific role in the processing and generation of sequential data:

1. **Input and Output Dimensions**: All models accept inputs and generate outputs that preserve the sequence length (`seq_len`) and the dimensionality of each proband (`proband_len`), ensuring structural consistency across the system.
2. **Use of Time Distributed Layers**: TimeDistributed wrappers are used extensively to apply the same operation across every time step in the input sequence, which allows for the processing of each time step while retaining the temporal sequence structure.
3. **Flattening and Reshaping**: Flattening of the input sequences is a common step before processing through the model's internal layers, followed by a subsequent reshaping to restore the sequence's original dimensions or transform it to the required output format.
4. **Recurrent Layers**: Both variations rely heavily on recurrent layers (GRU or LSTM) for processing the data. These layers are crucial for capturing the temporal dependencies within the sequences.
5. **Dropout Regularization**: To mitigate overfitting and improve generalization, dropout is applied within recurrent layers, providing a form of regularization by randomly setting input units to 0 at each update during training time.
6. **TimeDistributed Dense Layers**: These layers are applied after recurrent processing to either map the data to a higher-dimensional space or bring it back to its original dimensionality.

While there are unique elements in each variation, the overarching framework remains consistent. Below are the common model definitions shared by both versions:

- **Embedder**: The Embedder models across both variations receive the input sequence and map it into a latent space. The structure consists of input layers, recurrent layers, and TimeDistributed Dense layers, ending with a reshaping to the latent space.
- **Recovery**: The Recovery models take the latent representations and attempt to reconstruct the original sequence. This model mirrors the structure of the Embedder but operates in reverse, mapping from the latent space back to the sequence space.

- **Generator**: The Generators in both versions are designed to produce synthetic sequences. They incorporate initial layers to process the input noise, recurrent layers for sequence generation, and TimeDistributed Dense layers for output shaping.
- **Discriminator**: The Discriminator models assess the authenticity of the generated sequences. They process the latent representations and output a discrimination score for each time step in the sequence.
- **Supervisor**: The Supervisor models guide the generated sequences to be more realistic. They take the embedded representations and refine them to align more closely with the actual data characteristics.

The training of the model's components is a blend of several methodologies, including concepts from auto-encoders, supervised learning, and adversarial training, which are unified in a joint training strategy.

Auto-encoder Initialization The training commences with the auto-encoder, which employs an embedding function `H = embedder(X)` followed by a reconstruction to `X_tilde = recovery(H)`. The reconstruction loss `embedding_loss_t0` is optimized to improve the fidelity of the input reconstruction.

```
x_tilde = autoencoder(x)
embedding_loss_t0 = mse(x, x_tilde)
```

Supervised Training The supervisor module is crucial for instilling temporal coherence. It is trained on the latent space representations with the goal of accurately forecasting future latent states:

```
h = embedder(x)
h_hat_supervised = supervisor(h)
g_loss_s = mse(h[:, 1:, :], h_hat_supervised[:, :-1, :])
```

Adversarial Training Adversarial dynamics are introduced through a generator-discriminator confrontation. The generator creates synthetic representations while the discriminator evaluates their authenticity:

```
E_hat = generator(Z)
H_hat = supervisor(E_hat)
Y_fake = discriminator(H_hat)
```

Joint Training Strategy The culmination of training is a concerted process where the auto-encoder, supervisor, generator, and discriminator are concurrently refined. The loss functions across these components are balanced, such as the sequence coherence loss, which maintains the temporal structure in generated sequences:

```
G_loss_C = sequence_coherence_loss(h[:,1:,:],
    h_hat_supervised[:,1:,:])
```

Optimizers are calibrated to ensure stable training across the different components, a task executed by applying gradients computed from each model's respective loss function:

```
gradients = tape.gradient(e_loss, var_list)
embedding_optimizer.apply_gradients(zip(gradients, var_list))
```

In synthesizing these elements, the architecture iteratively improves the quality of synthetic sequences, ensuring they are statistically representative and temporally consistent.

The loss functions used across both variations of the model play a crucial role in the training process, serving distinct purposes in shaping the model's behavior.

Reconstruction Loss At the core of our architecture lies the reconstruction loss, quantified as the mean squared error (MSE) between the input data, denoted as `X`, and its corresponding reconstruction, represented as `X_tilde`, derived from the latent space representation `H`. This loss serves a fundamental purpose—ensuring that both the embedder and recovery (decoder) components of the model excel in faithfully reproducing the original input data, thereby preserving vital data characteristics within the latent space.

```
embedding_loss_t0 = mse(x, x_tilde)
```

Notably, the reconstruction loss is amplified by a scaling factor during gradient updates. This amplification factor acts as a hyperparameter, carefully balancing the importance of reconstruction quality with other training aspects.

Supervised Loss Our model features a supervisor module, trained using a supervised loss. In our implementation, this loss metric measures the mean squared error (MSE) between the original embedded sequence `H` and the predicted subsequent sequence `H_hat_supervised`. The purpose of this loss is to guide the supervisor in accurately predicting future states within the latent space—a pivotal aspect for achieving temporal coherence in the synthetic sequences generated by our model.

```
g_loss_s = mse(h[:, 1:, :], h_hat_supervised[:, :-1, :])
```

Moment Loss A novel addition to our model is the moment loss, designed to align the first and second statistical moments (mean and variance) of the synthetic data with those of the real data. This loss ensures that our synthetic data not only exhibits realism on an individual instance basis but also matches the statistical characteristics of the real data distribution.

```
generator_moment_loss = get_generator_moment_loss(x, x_hat)
```

Sequence Coherence Loss Our model introduces a sequence coherence loss, aimed at minimizing the divergence between the evolution of real and synthetic data over time. This loss computes the mean squared error (MSE) between the differences (changes) in consecutive latent space representations of real and synthetic data points.

```
G_loss_C = sequence_coherence_loss(h[:,1:,:],
    h_hat_supervised[:,1:,:])
```

Adversarial Losses Adversarial training plays a pivotal role in imbuing realism into the synthetic data. The generator's objective is to deceive the discriminator by crafting synthetic sequences that can be indistinguishable from real ones. Conversely, the discriminator strives to discriminate between genuine and synthetic data. These adversarial losses are operationalized through binary cross-entropy (BCE), closely aligning with the generative adversarial network (GAN) framework.

```
generator_loss_unsupervised = bce(y_true=tf.ones_like(y_fake)
    , y_pred=y_fake)
```

In the discriminator's loss function, penalties are incurred when it fails to identify real data accurately or when it erroneously labels synthetic data produced by the generator.

```
discriminator_loss_real = bce(y_true=tf.ones_like(y_real),
    y_pred=y_real)
```

In tandem, these loss components orchestrate a comprehensive and multi-faceted training regimen:

- Reconstruction Loss: Ensures the preservation of data fidelity within the latent space.
- Supervised Loss: Facilitates the accurate prediction of future latent states, crucial for maintaining temporal coherence in generated sequences.
- Moment Loss: Aligns the statistical properties of synthetic data with those of the real data distribution.
- Sequence Coherence Loss: Minimizes divergence between the temporal evolution of real and synthetic data.
- Adversarial Losses: Infuses an element of realism and diversity into the synthetic data, facilitating the generation of data that closely resembles real-world data.

By diligently balancing these loss components, our model can be fine-tuned to produce synthetic sequences that should exhibit both realism and coherence, all while adhering to the complex statistical properties characterizing the original dataset. These loss functions transcend their roles as guiding metrics to serve as fundamental performance indicators, providing valuable insights into the model's convergence and the quality of the generated data.

Variation 1: Enhanced RNN Architecture with GRU Layers and Initial State Optimization Variation 1 introduces several key enhancements to the baseline architecture that set it apart. Central to its design is the exclusive use of Gated Recurrent Unit (GRU) layers within the recurrent processing components. This variation foregoes the LSTM layers found in Variation 2, capitalizing on the efficiency and similar performance characteristics that GRUs provide. The Key Specialties of Variation 1 are as follows:

1. **GRU-based Recurrent Layers**: Variation 1 leverages GRU layers for all recurrent processing needs across the Embedder, Recovery, Discriminator, and Supervisor modules. GRUs are known for their ability to capture dependencies without the additional complexity associated with LSTMs, providing a balance between computational efficiency and modeling capacity.
2. **Initial State Optimization**: Another aspect of Variation 1 is the initialization of the GRU layers within the Generator. Instead of random initial states or zero vectors, this variation employs a dedicated small feed-forward network to generate optimized initial states, which is hypothesized to contribute to the rapid convergence and improved quality of the synthetic sequences.
3. **Residual Connections in Generative Layers**: Residual connections are incorporated into the Generator's GRU layers, an architectural feature not present in Variation 2. These connections allow the gradients to flow more easily through the network, potentially addressing issues related to vanishing gradients and enabling the training of deeper models.
4. **Customized TimeDistributed Processing**: Both the Embedder and Recovery models utilize a TimeDistributed Dense layer immediately before reshaping the output to the latent space or reconstructed sequence. This allows the network to make fine-grained adjustments at each timestep, potentially enhancing the detail and fidelity of the generated sequences.

The combination of these features in Variation 1 offers a powerful yet computationally efficient approach to synthetic sequential data generation. The use of GRU layers, initial state optimization, and residual connections distinguishes it from its counterpart, providing a distinct architectural pathway aimed at enhancing generative performance.

Variation 2 Variation 2 of the synthetic sequential data generation architecture incorporates specific features that differentiate it from Variation 1 and the baseline structure:

1. **LSTM-based Recurrent Layers**: Unlike Variation 1 which uses GRU layers, Variation 2 primarily employs LSTM layers for its recurrent processing across all components. LSTMs are well-regarded for their ability to remember long-term dependencies, which is particularly beneficial when dealing with complex sequential data that requires long-range temporal understanding.
2. **Convolutional Feature Extraction**: Prior to recurrent processing, a convolutional layer (Conv1D) is used for feature extraction. This allows Variation 2 to potentially capture more complex patterns in the sequential input data before it is processed by the LSTM layers.

3. **Multiple LSTM Layers with Dropout**: Variation 2 stacks multiple LSTM layers with dropout regularization to deepen the model's capacity for learning hierarchical temporal features while controlling for overfitting. This contrasts with Variation 1, where single GRU layers are used without stacking.
4. **Consistent Use of Dropout**: A uniform dropout rate of 0.25 is applied across all LSTM layers in Variation 2, providing a consistent regularization strategy that can be beneficial for learning stability and model robustness.
5. **Smoothing Loss**: In addition to the standard loss components, Variation 2 introduces a smoothing loss which encourages the generated sequences to exhibit temporal smoothness. This is crucial for sequences where sudden changes between consecutive points are unrealistic or undesirable. The smoothing loss is computed as the mean squared error between consecutive points in a sequence, thereby penalizing large deviations in the derivative of the sequence.

```
def smoothing_loss(sequence):
"""Computes a loss based on the difference between
    consecutive points in a sequence."""
return tf.reduce_mean(tf.square(sequence[:, 1:, :] - sequence
    [:, :-1, :]))
```

These key specialties underscore Variation 2's commitment to capturing deeper temporal dynamics in sequential data generation, employing a more layered and regularized approach than its counterpart. The inclusion of LSTM layers, convolutional feature extraction, uniform dropout regularization, and the smoothing loss endows Variation 2 with a distinctive architecture optimized for complex sequence modeling tasks that require not just fidelity and realism but also smooth temporal transitions.

By integrating the smoothing loss into Variation 2, the architecture is further refined to ensure that the generated sequences not only preserve the complex patterns within the data but also maintain a realistic and coherent flow, mirroring the continuity often found in real-world temporal data. This enhancement addresses one of the subtle yet crucial aspects of sequence generation—ensuring that each generated point is consistent with its temporal context, thereby improving the overall quality and usability of the synthetic data.

4.3 Setup Default Configuration

This section describes the fundamental hyperparameters and settings established as the default configuration for our model.

Learning Rate Configuration The learning rate is a critical component in the training of neural networks, governing the adjustments made to the model weights with each iteration. In our default setup, different learning rates are assigned to various components of the model:

- Generator Optimizer: Utilizes the standard Adam optimizer without explicit learning rate specification, thereby adopting the default learning rate of the Adam optimizer.
- Discriminator Optimizer: Employs an Adam optimizer with a learning rate of 0.0005, a value chosen to balance fast learning while avoiding overshooting the minima in the optimization landscape.
- Embedding Optimizer: Similarly uses the Adam optimizer, relying on its default learning rate setting.

Batch Size The batch size, set at 64, determines the number of data points processed before the model updates its parameters. This size is selected to optimize the computational efficiency and memory usage, ensuring a smooth training process across various hardware setups.

Data Normalization Normalization is crucial for aligning data within a specific scale, enhancing model performance. Our approach offers two normalization strategies:

- Fixed Geographical Window: Applies a predetermined range for longitude and latitude values, standardizing data inputs to this fixed scale.
- Dynamic Range Calculation: Dynamically calculates the minimum and maximum longitude and latitude from the data, offering a flexible normalization tailored to the dataset's specific geographical spread.

These normalization methods ensure that the model receives data in a consistent format, crucial for learning patterns and dependencies effectively.

Training Frequency of Discriminator and Generator A distinctive aspect of our training is the discriminator's training frequency, set at three times the rate of the generator. This strategy fosters a balanced adversarial dynamic, crucial for the progressive improvement of both components in generating realistic synthetic data.

Weights of Losses The model's training involves several loss functions, each contributing to different aspects of learning:

- Sequence and Coherence Loss: These losses are crucial for ensuring the temporal coherence and sequence accuracy of the generated data. We weigh these losses (multiplied by factors of 10 for the sequence coherence loss and smoothing loss) to emphasize their importance in the model's training objective.
- Moment Loss: This loss aligns the statistical distribution of the generated data with the real data, weighted significantly (multiplied by 100) to ensure the synthetic data accurately reflects the statistical properties of the real-world data.

4.4 Experiments Overview

Table 2 shows the experiments that were conducted. Each experiment is assigned a number for identification in the subsequent chapters. They vary in model architecture (following chapter 4.2), in dimensions of training data, in amount of trained epochs and in other setup modifications (that differ from the default setup as explained in chapter 4.3).

Table 2: Experiment runs and their specifications.

Exp. Nr.	Model Architecture	Dimensions Training Data (w, s, r, 2)	Epochs	Setup Modifications
Exp. 1	Baseline	(400, 100, 5, 2)	200	-
Exp. 2	Variation 1	(456, 50, 10, 2)	200	-
Exp. 3	Variation 1	(456, 50, 10, 2)	200	dynamic normalisation
Exp. 4	Variation 1	(400, 100, 5, 2)	200	-
Exp. 5	Variation 1	(400, 100, 10, 2)	5.000	-
Exp. 6	Variation 2	(456, 50, 10, 2)	200	-
Exp. 7	Variation 2	(400, 100, 5, 2)	200	-
Exp. 8	Variation 2	(400, 100, 5, 2)	600	-
Exp. 9	Variation 2	(400, 100, 10, 2)	5.000	-
Exp. 10	Variation 2	(1.000, 1.000, 5, 2)	200	batch size 16

4.5 Data Postprocessing

In several experiments it could be observed that the first GPS point of a synthetically generated route is often an extreme outlier. As a result a postprocessing step for all output routes is added where the first coordinate gets cut off. Like this the route has a more suitable starting point. An example is provided in Figure 6.

(a) without postprocessing (b) with postprocessing

Fig. 6: In the postprocessing step the first point of each synthetic route gets cut. The synthetic route (blue) often has a starting point that is an outlier (a). After removing that point the route is more smooth (b) and more similar to the real route (red).

4.6 Evaluation

Two qualitative and three quantitative methods are employed to evaluate the experiment outputs.

The first qualitative method, a combination of the **Principal Component Analysis (PCA)** [4] and the **t-Distributed Stochastic Neighbor Embedding (t-SNE)** [12], are drawn from the original TimeGAN Paper [18]. These methods demonstrate the similarity between original and generated distributions. Visually, greater overlap of the real and synthetic data points indicates more similar distributions. Additionally, as we are dealing with gps data we come up with a second qualitative method: the **visual comparison** of the real route and one ore more synthetic pendants on a map.

Furthermore we also use two quantitative approaches suggested by the original TimeGAN work. The first one is the **post-hoc training of a supervised classification model** that has to distinguish between real and generated sequences. For that we use a simple sequential model of one GRU and one Dense Layer. A lower accuracy means the real and synthetic routes are more indistinguishable. The second one is the **train-on-synthetic test-on-real (TSTR)** framework [5, 8] to "evaluate how well the generated data preserves the predictive characteristics of the original" [18]. A predictive model, that has to predict new values of the time sequence, gets trained on synthetic data and then tested on real data. The TSTR model's performance indicates how effectively a predictive model, trained on synthetic data, can generalize to real data, thereby reflecting the extent to which synthetic data characteristics represent those of the real data. For that we use again a sequential model consisting of one GRU and one Dense Layer. It gets evaluated via the Mean Absolute Error (MAE). Additionally, we compare that result with a model that is trained and tested on real data (TRTR) to assess how well the TSTR model performs in comparison to predictions learned from real data [16]. Additionally, we introduce two quantitative measures, **MDD (Mean Distance Difference)** and **Mean Radius Difference (MRD)**, to concretely assess similarity to the original routes. MDD measures the average of the absolute and relative deviation between the travel distance of the original route and the travel distance of all corresponding generated routes. MRD measures the radius in the same way. Radius in this case refers to the circle that is span when regarding the most distant gps points of a route so that all route points lie within this circle. In both cases we do not differentiate between negative and positive deviations, but only absolute values, i.e. high values can mean both longer and shorter paths or both bigger and smaller radii respectively.

Table 3 summarises the aforementioned methods with their respective type and main goal. Derived from the original TimeGAN authors [18] these goals can be described as **Usefulness** ("samples should be just as useful as the real data when used for the same predictive purposes"), **Diversity** ("samples should be distributed to cover the real data") and **Fidelity** ("samples should be indistinguishable from the real data"). Furthermore each evaluation method is defined by several measures based on which the experiments get rated.

Table 3: Summary of evaluation methods.

Evaluation Method + related measures	Type	Main Quality Goal
(1) Visual Comparison	qualitative	Fidelity
(1.1) Smoothness when changing directions		
(1.2) (Non-)Spinning		
(1.3) Diversity of variations		
(1.4) Proximity to original route		
(1.5) Following road network		
(2) PCA and t-sne	qualitative	Diversity
(2.1) PCA plot		
(2.2) t-sne plot		
(3) MDD and MRD	quantitative	Fidelity
(3.1) MDD		
(3.2) MRD		
(4) Post-hoc Classification	quantitative	Fidelity
(4.1) Training Accuracy		
(4.2) Validation Accuracy		
(5) TSTR vs. TRTR	quantitative	Usefulness
(5.1) Test MAE (TSTR)		
(5.2) Deviation to Val MAE (TSTR)		
(5.3) Deviation to Test MAE (TRTR)		

5 Results and Discussion

In this chapter, the results of the evaluation are presented and discussed. First an overall result is given (see Table 4) before the individual evaluation results are presented in detail in the following subsections.

Table 4: Summary of evaluation results. A cross means the respective experiment did best in this evaluation measure. Some measures are won by multiple experiments if they achieved similar results or by zero experiments if none performed well at all. The final count shows how often each experiment was among the winners (the higher the better).

	Evaluation measure												
	(1)					(2)		(3)		(4)		(5)			
	(1.1)	(1.2)	(1.3)	(1.4)	(1.5)	(2.1)	(2.2)	(3.1)	(3.2)	(4.1)	(4.2)	(5.1)	(5.2)	(5.3)	**Count**
Exp. 1		x		x											**2**
Exp. 2	x	x	x	x											**4**
Exp. 3	x	x	x	x											**4**
Exp. 4			x	x		x									**3**
Exp. 5	x	x													**2**
Exp. 6	x	x		x								x		x	**5**
Exp. 7	x		x	x											**3**
Exp. 8	x		x	x											**3**
Exp. 9		x		x				x		x	x		x		**6**
Exp. 10			x	x		x			x						**4**

Overall, Exp. 9 is the leading experiment with performing best in 6 of the 14 measures, followed by Exp. 6 with a score of 5. Nevertheless, none of the models performed to our satisfaction as none of the results are keeping up with results of TimeGAN applications in other domains, e.g. in sines, stocks, energy or events [18]. Possible indications and reasons are given in the conclusion of this work (see chapter 6).

5.1 Visual Comparison

Each experiment displays distinct key characteristics in its output data when visualized on a map. Visual examples for one input route each are given in Figure 7. The observations on key characteristics can be described as follows.

- **Smoothness when changing directions (sharp vs. smooth vs. mixed) and similarity to original route**: Certain models exhibit a preference for specific ways of changing directions, independent of the respective input route's behavior. Exp. 4 and 9 predominantly produce sharp turns and Exp. 10 smooth curves, even if the input route has other characteristics. In contrast, the outputs of Exp. 2, 3 and 5 to 8 seem to follow the turning behaviour of the respective input route (either mainly smooth, mainly sharp or mixed, dependent on the input route). This indicates that the model learned the characteristics of turning on some level.
- **(Non-)Spinning**: While all other experiments seem to be able to generate paths that do not turn around 180 degrees unrealistically often, Exp. 4, 7, 8 and 10 have the tendency to do so (contrary to their input routes).
- **Diversity of variations**: For each input route the trained model generates several variations of output routes. These variations differ more or less from each other. While Exp. 1, 5 and 6 tend to generate routes that are identical in most points, Exp. 9 is the only model where the generated routes differ much in location and arrangement of the points. The outputs for each route of Exp. 2 to 4, 7, 8 and 10 tend to follow the same pattern but with mild changes. It can be assumed that Exp. 1, 5 and 6 only memorize the training route, Exp. 9 did not learn enough from the original route and Exp. 2 to 4, 7, 8 and 10 seem to generalize just enough to be able to produce varying routes while maintaining some original characteristics.
- **Proximity to original route**: The generated routes of all experiments are geographically near the original route except those of Exp. 5. Its outputs are consistently several kilometers apart from the original route which indicates that the model did not focus on the absolute values of the data during training.
- **Following road network**: An overall problem that can be seen here is that none of the routes knows the road network. They do not follow paths but instead cross terrain that would not be efficient or even possible in real life. Obviously the models did not learn that implicitly from the training routes. To improve that it would be necessary to explicitly add information about road networks in model training.

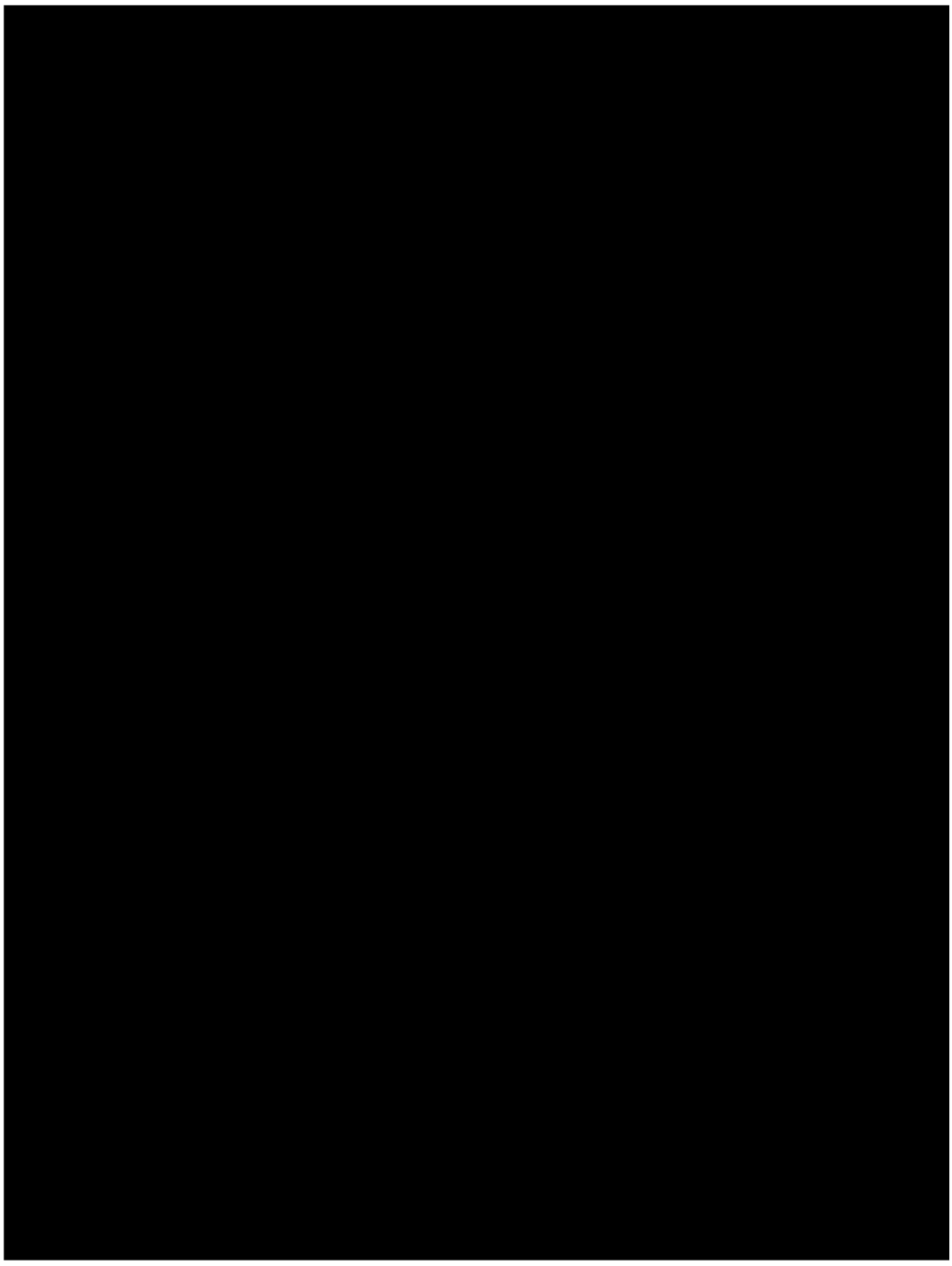
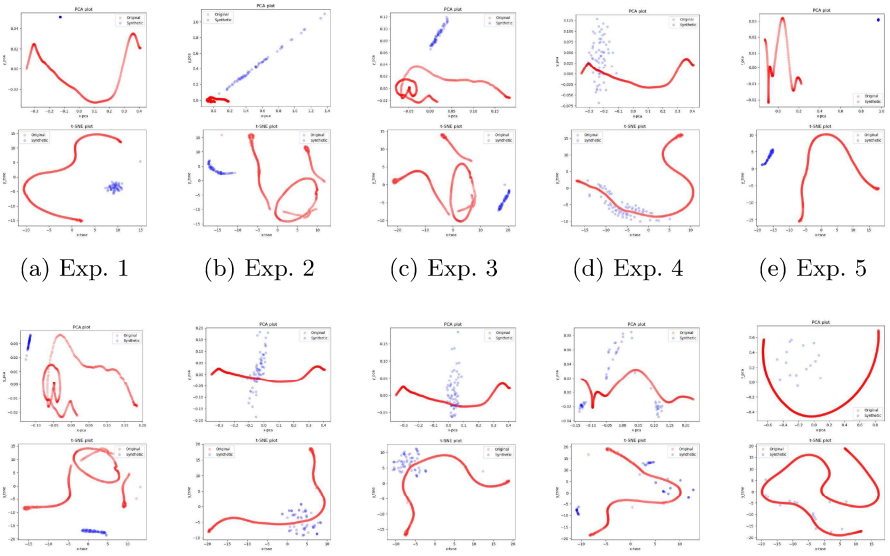


Fig. 7: Real (red) versus two variations of subroutes (blue and green) generated by different model setups (a to j).

5.2 PCA and t-sne



(a) Exp. 1 (b) Exp. 2 (c) Exp. 3 (d) Exp. 4 (e) Exp. 5

(f) Exp. 6 (g) Exp. 7 (h) Exp. 8 (i) Exp. 9 (j) Exp. 10

Fig. 8: Results for the qualitative methods PCA (top) and t-sne (bottom). The more the synthetic points (blue) overlap with the original points (red) the more similar the distributions.

Figure 8 visualizes the results of the PCA and t-sne analyses. The visible difference in the number of points does not play a crucial role in this assessment. This is due to the real dataset containing more gps points in sum than the synthetic one because of the window data structure that is used in TimeGAN. As mentioned in chapter 4.1 the input structure consists of the dimensions $(w, s, r, 2)$ while the output data has the dimensions $(b, s, r, 2)$. Since the batch size b is usually smaller than the window size w the output data contains fewer points in total across all routes (in our case the batch size was mostly set to 64 while the window size is either 400, 456 or 1,000 depending on the chosen sequence length). What is essential is the position of the points. For both methods the rule is the more overlapping the points the more similar the distributions. Overall the results do not show this behaviour in most of the experiments. In fact the points of the original and the generated data deviate much in their proximity and their patterns which means their distributions are not very similar. However Exp. 4 and to some extent Exp. 10 stand out positively as their t-sne result show that the generated distributions tend to follow the original ones.

5.3 MDD and MRD

MDD measures the mean deviation of path length (distance) between original and synthethic routes, while MRD measures the same for radius. The goal is to

have deviations as low as possible which would mean the generated routes are similar in length and circumference to their original pendants. Table 5 shows the MDD and MRD values for each experiment.

Overall the average distance deviations lie between 51% (Exp. 9) and 1,324% (Exp. 10). This means the generated routes of all experiments deviate several kilometers from their original pendants on average. The dominance of high values indicates that most models did not learn the distances of their training routes. However, four experiments at least managed to generate routes that are max. 100% longer or shorter than the real routes on average (Exp. 1, 3, 6 and 9).

Regarding the radius the models performed better overall. The mean differences lie between 25% (Exp. 10) and 137% (Exp. 5), apart from one outlier that has a difference of 1,704% (Exp. 2). Eight of the models succeeded in generating routes that span a circle that is maximum 100% bigger or smaller on average. This indicates that the models did learn the circumferences of their training routes to some extent. Only in Exp. 2 and 5 the generated routes tend to have a much huger span then the original routes (mean deviation of 1,704% and 137%).

Table 5: Mean Distance Difference (MDD) and Mean Radius Difference (MRD) for each Experiment (Lower Values Indicate Better Performance).

Exp. Nr.	MDD (km)	MDD (%)	MRD (km)	MRD (%)
1	30	98%	11	100%
2	123	634%	47	1,704%
3	44	56%	5	46%
4	320	973%	3	57%
5	49	127%	19	137%
6	66	88%	10	86%
7	65	231%	3	87%
8	94	277%	2	41%
9	20	**51%**	11	69%
10	2,176	1,324%	16	**25%**

5.4 Post-hoc Classification

The outputs of each experiment were evaluated with a classification model that has to distinguish between real and fake routes (each trained for 250 epochs). The result is that after maximum 90 epochs the classification model reaches a plateau of 100% accuracy with train and validation data in Exp. 1 to 7 (see Table 6). This means in these experiments the synthetic and real routes have such different characteristics that they are highly distinguishable. This observation is negative as the synthetic routes should be similar to the original routes and thus not be classified easily. In Exp. 8 and 10 the model needs considerably longer but finally reaches 100% (Exp. 10) or almost 100% (Exp. 8) train and validation accuracy as well. Additionally in these two output data sets there are more accuracy drops after first reaching the highest value and thus no stable plateau in comparison to Exp. 1 to 7. This means the classifier seems to struggle to distinguish between synthetic and real to a small extent. However the most

significant exception is Exp. 9 where the classifier only reaches 82.13% accuracy with train and 85.16% with validation data. This still high but comparably good result indicates that these synthetic routes are harder to distinguish and thus have similar characteristics as their original pendants to some degree.

Table 6: Best Training and Validation Accuracy of post-hoc Classification for each experiment output (the lower the better) and when first reached respectively.

Exp. Nr.	Training Accuracy	Validation Accuracy	Epochs (train/val)
1	100.00%	100.00%	30/29
2	100.00%	100.00%	38/31
3	100.00%	100.00%	76/72
4	100.00%	100.00%	42/40
5	100.00%	100.00%	26/25
6	100.00%	100.00%	30/22
7	100.00%	100.00%	90/81
8	97.85%	97.66%	122/79
9	**82.13%**	**85.16%**	100/119
10	100.00%	100.00%	193/184

5.5 TSTR vs. TRTR

In this evaluation method a model that has to predict the next value in a time series sequence in one case sees only synthetic data (TSTR) during training and in another case only real data (TRTR). Both models get evaluated with real test data after training. This is done with each experiment output separately. Training, Validation and Test Phase are evaluated by Mean Absolute Error (MAE). The values are normalized before training so that regional gps differences do not skew the MAE comparisons. The TRTR is used to validate the TSTR results.

Overall the training progress of all experiments is very similar (see Figure 9 for an example). The training and validation loss shrinks rapidly in the beginning and finally finds a plateau after some epochs in both TSTR and TRTR setting. The Test loss is remarkably higher than the Train and Val loss in the TSTR model whereas in the TRTR model it is just slightly higher.

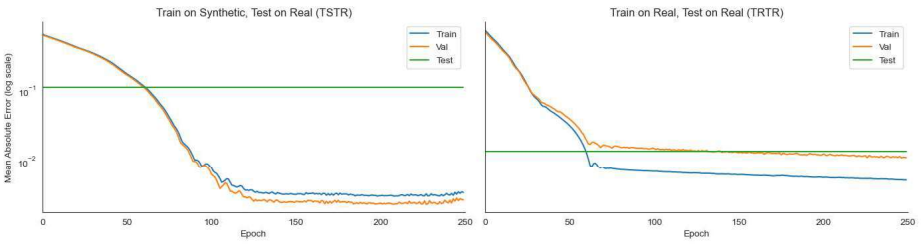


Fig. 9: MAE for TSTR and TRTR predictive models with output data from Exp. 2 as representative for all experiments *(caution: Test MAE is only evaluated once after training, x axis does not apply here).*

The following aspects can be observed.

Diversity of synthetic routes: The synthetic variations of each route are not sufficiently diverse. This finding is based on the fact that in the TSTR setup Train and Val loss is almost identical in each epoch. Both datasets contain a different part of synthetic variations for all routes. Like that the model learns the training variations and and as the validation variations do not differ enough it performs almost identically well (no matter how Train and Val is split).

Usefulness: The most useful synthetic data when looking only at the Test MAE in the TSTR setup is that from Exp. 6 followed closely by Exp. 3 and Exp. 9 (see Table 7 column 1). Nevertheless it is assumed that all TSTR models are overfitting to the synthetic data and can not generalise well on real data. This can be seen when comparing the best Test MAE of TSTR to the best Val TSTR in each experiment (see column 2). In the test phase (with real data) the errors are remarkably higher than in the validation phase (with synthetic data). The only comparably good result is Exp. 9 with only being 31.75% worse with real data than with synthetic data. Finally, the Test MAE of TSTR is always higher than the Test MAE of TRTR (see column 3) which shows that training on real data obviously is much more effective than on synthetic data in terms of performance on real data. However Exp. 6 followed by Exp. 3 and Exp. 9 have the best results and thus are closest to their counterparts trained on real data.

The final conclusion is that the predictive models trained on the synthetic data of our experiments are not well applicable on real data. In comparison the predictive models trained on real data are in fact applicable to real data. This indicates that the synthetic routes apparently have strong deviating characteristics from the real ones. Only Exp. 6 and 9 stood out with best values and thus can generalise to some extent.

Table 7: Test MAE for TSTR predictive model for each experiment output and comparison to Val MAE (TSTR) and Test MAE (TRTR) (for all of them, the lower the better).

Exp. Nr.	Test MAE (TSTR)	Deviation to Val MAE (TSTR)	Deviation to Test MAE (TRTR)
1	12.82×10^{-2}	+99.18%	+74.00%
2	18.58×10^{-2}	+98.64%	+92.33%
3	2.03×10^{-2}	+97.34%	+37.60%
4	14.34×10^{-2}	+79.82%	+82.08%
5	33.47×10^{-2}	+99.65%	+95.78%
6	$\mathbf{1.72 \times 10^{-2}}$	+93.09%	+**24.81**%
7	12.91×10^{-2}	+72.13%	+80.89%
8	9.68×10^{-2}	+74.87%	+83.72%
9	2.52×10^{-2}	+**31.75**%	+43.55%
10	11.34×10^{-2}	+77.88%	+83.82%

6 Conclusion and Outlook

In this paper we demonstrated how to use TimeGAN to generate synthetic GPS data of human mobility. We conducted experiments with ten different setups based on three architectures. The results got evaluated using two qualitative and three quantitative methods.

Suitability of TimeGAN for synthetic mobility data generation The aim of our experiments was to examine how well TimeGAN can imitate mobility data. Therefore we evaluated the output data in terms of three objectives: Usefulness, Diversity and Fidelity, i.e. the synthetic data should be similarly useful and distributed as the real data as well as indistinguishable from it. Concerning all three goals there were nuances between the different results (scoring between 2 and 6 out of 14 possible points) but nevertheless all of them did not perform to our satisfaction. Neither were the synthetic data suitable for prediction tasks, nor did the distributions of synthetic and real data match, and finally, all outputs were clearly distinguishable from their respective inputs.

One reason for the unsatisfying results could be the two-dimensional nature of our input data. As mentioned earlier, to the best of our knowledge, at this point there is no work yet that uses TimeGAN to generate data where the features are more than one-dimensional. To reduce the two-dimensional nature of our data (latitude and longitude), we applied TimeDistributed layers in our networks after each Input Layer. "This wrapper allows to apply a layer to every temporal slice of an input" [9] and therefore reduces one dimension for the following layers. This customisation could be one reason for poor performance with TimeGAN as these two concepts were not designed together. Another reason could be a poor performing autoencoder being responsible for too much loss of information in the data during the transformation into the latent space.

Furthermore, the heterogeneous characteristics of training routes could be a cause for imprecise results. We did not group the training data based on types of routes, such as car versus bike trips and urban versus rural routes. This approach would require engineering different sets of preprocessing rules, while we applied one rule set for the whole data, as mentioned in chapter 4.1. This could offer insights whether specific rules enhance the performance for certain types of data.

A final major problem is the circumstance that the synthetic routes do not follow the road network because the model simply does not have information about it during training. It would be necessary to append road information in training, e.g. by restricting the models´ output to certain gps areas or by training a second model and combining it with the generative model.

To conclude, TimeGAN might perform well in other domains, but we would not recommend using TimeGAN for the synthetic generation of human mobility data.

Architectural decisions The conducted experiments performed at different levels as summarised in the results in Table 4. When tracing back these results on

the architectural design and setup of the individual experiment some conclusions can be drawn.

- **TimeGAN architecture adjustments**: Among the two variations and the baseline architecture, Variation 2 (see chapter 4.2), which exclusively uses LSTM layers for recurrent processing, showed the most promising results. This variation stands out due to its specific architectural choices: the use of LSTM layers, the incorporation of a Convolutional Layer for feature extraction, stacking of LSTM layers with consistent dropout application, and the novel introduction of a smoothing loss. These elements collectively enhance the architecture's suitability for generating synthetic mobility data.
- **Training time**: The number of epochs the model trains does play a role in its quality under one condition. Exp. 9, which scored best, trained for 5,000 epochs, while, on the other hand, Exp. 5 trained just as long and is among the worst results. Exp. 6 who scored second trained for 200 epochs. Concluded, training time is improving the models results under the condition that the underlying architecture and setup is already suited for the present task.
- **Learning Setup**: The default setup with custom Adam Optimizer learning rates, batch size of 64, fixed data normalization, a training ratio of 3:1 between discriminator and generator and custom loss weights (see chapter 4.3) led to evaluable results for our experiments. As this setup is implemented in the two best experiments it seems to be a stable base. Other modifications, implemented in Exp. 3 (dynamic instead of fixed normalisation) and in Exp. 10 (batch size of 16 instead of 64), did not show extraordinary behaviour.

Nevertheless, besides the indications above, the modifications regarding the architecture variations, the data dimensions, the training length and the learning setup, were not systematically adjusted and evaluated. A more systematic approach of testing different options could lead to more significant insights on how crucially each adjustment influences the final result.

Evaluation methods For evaluating the synthetic output we used two qualitative and three quantitative methods. They appear differently suitable for the task of evaluating synthetic gps data. The following conclusions can be drawn.

- **Visual Comparison**: The measures of this method, such as diversity of variations, smoothness when changing directions, proximity to original route, (non-)spinning and following road network, were introduced by ourselves. The focus lies on the question, if qualitative aspects of the synthetic routes are similar to the original routes and if they behave like real routes to the human eye. We believe that these questions are well answered by the five measures but not yet fully sophisticated. More aspects could be added and validated by literature of the domain of geographies to extract qualitative requirements of gps data.

- **PCA and t-sne**: These two visual methods appear to be an effective and fast way to compare the complex distributions of high-dimensional real and synthetic mobility data. Despite the different amount of data points of both datasets the diverging patterns of the distributions could be perceived straightforwardly and thus allowed comparisons between their inner structures.
- **MDD and MRD**: We came up with two measures concerning distance and radius of routes after realising that reflecting on them only visually is not significant enough while being convinced that they are important factors of high-quality synthetic mobility data. Indeed they enabled insights on a professional level that no other quantitative method could because the latter ones are designed for synthetic data in general while MDD and MRD are only applicable in a geographic settings.
- **Post-hoc Classification**: The post-hoc training of a real vs. synthetic classifier was helpful as it confirmed our assumptions on how distinguishable the two datasets are. While such assumptions can arise on random samples, an evidence for the whole dataset is only efficiently given through methods like the post-hoc classification, i.e. in our case, the fact that a simple classifier model reaches 100% accuracy in most cases proves that the characteristics of the datasets are highly distinct.
- **TSTR vs. TRTR**: There are two different implementations of the TSTR framework used in combination with TimeGAN. We mixed them in a sensible way. Firstly, the original TimeGAN codebase [17] does not conduct TRTR but only TSTR. However, we found it necessary to validate if results trained on synthetic data are comparable to results trained the conventional way on real data. The source of the TRTR application [16] though uses the real data already in the validation phase of the training in contrast to using it only after training for one-time testing. This seemed not target-oriented to us as the model will learn from real data in the validation phase and thus there are no insights on the actual question, if a model fully trained on synthetic data can generalise on real data when it first sees it. Therefore, we suggest using both the TRTR and TSTR approaches, but with the train-, validation- and test-split as mentioned in the original TimeGAN work.

To summarise, we perceive the used evaluation methods as helpful with some restrictions. We believe we added two valuable professional methods in the field of geographic data (visual comparison and MDD/MRD) to the already existing evaluation methods for synthetic data evaluation in general and thus had an appropriate mix of both. Finally we would like to emphasize that we consciously decided to have a focus on the evaluation goal Fidelity (main objective of three of the five methods) as we perceive the non-distinguishability as a prerequisite for the goals Diversity and Usefulness. More balanced method choices could be advantageous in other settings nevertheless. Another conscious focus was not to mainly compute summarised scores like in the Original TimeGAN (e.g. one predictive score instead of comparing different MAEs in the TSTR/TRTR

method). Our aim was to give detailed information on where TimeGAN performs how well in the task of generating synthetic mobility data.

Outlook As analyzed by Gao et al. in 2022 [6], there are no scientific approaches yet where the goal is to generate synthetic data while treating mobility or trajectory data as a Time Series data type. More common is the use of models focused on predicting the future values of trajectory data. Therefore, this work contributes to viewing mobility data as a Time Series problem and uses methods like TimeGAN, while also revealing several challenges that could suggest a different approach.

First of all, one area for further research is the verification of our aforementioned assumptions on why TimeGAN did not perform well on mobility data, i.e. quantifying the impact of the TimeDistributed layers, the autoencoder and the heterogenous route types in the training data.

When continuing to view mobility data primarily as a time-series problem, we suggest putting effort into learning road networks, as this is a major reason why our synthetic data is not yet useful for real scenarios. Moreover, further examination of time-series generation models from other domains (see Gao et al. [6, p. 1:10]) could be conducted to evaluate their suitability for two-dimensional GPS data.

The challenges we faced might also lead to a focus on machine learning models already designed for tasks with mobility data. Future research could examine DNNs, developed for predicting further values in trajectory data, such as those analyzed by Gao et al. [6, p.1:10] (GD-GAN, SocialGAN, SoPhie, Social Ways, Social-BiGAT, APOIR, CoL-GAN, AdattTUL, and MT-ASTN), or moGAN as introduced by Mauro et al. [13], for their applicability in synthetic generation tasks.

References

[1] Ahmed Alhasan. "Generating Geospatial Trip Data Using Deep Neural Networks". MA thesis. Linköping: Linköping University, 2022. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-182591 (visited on 12/16/2023).

[2] Hiba Arnout, Johanna Bronner, and Thomas Runkler. "Differentially Private Time Series Generation". In: *ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2021, pp. 617–622. DOI: https://doi.org/10.14428/esann/2021.ES2021-20.

[3] Alex Berke et al. "Generating Synthetic Mobility Data for a Realistic Population with RNNs to Improve Utility and Privacy". In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22)*. 2022, pp. 964–967. DOI: https://doi.org/10.1145/3477314.3507230.

[4] Fred B. Bryant and Paul R. Yarnold. "Principal-components analysis and exploratory and confirmatory factor analysis." In: *Reading and understanding multivariate statistics.* American Psychological Association, 1995, pp. 99–136. ISBN: 1-55798-273-2.

[5] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. *Real-valued (medical) time series generation with recurrent conditional gans.* 2017. DOI: https://doi.org/10.48550/arXiv.1706.02633. preprint.

[6] Nan Gao et al. "Generative Adversarial Networks for Spatio-temporal Data: A Survey". In: *ACM Transactions on Intelligent Systems and Technology* 13.2 (2022), pp. 1–25. DOI: https://doi.org/10.1145/3474838.

[7] Ian J. Goodfellow et al. *Generative Adversarial Networks.* 2014. DOI: https://doi.org/10.48550/arXiv.1406.2661. preprint.

[8] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. "PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees". In: *International Conference on Learning Representations.* 2018. URL: https://openreview.net/forum?id=S1zk9iRqF7 (visited on 12/16/2023).

[9] Keras. *Keras 3 API documentation: TimeDistributed layer.* URL: https://keras.io/api/layers/recurrent_layers/time_distributed/ (visited on 12/16/2023).

[10] Ling-Man Liu et al. "Dual-dimension Time-GGAN data augmentation method for improving the performance of deep learning models for PV power forecasting". In: *Energy Reports* 9 (2023), pp. 6419–6433. DOI: https://doi.org/10.1016/j.egyr.2023.05.226.

[11] Zigang Liu et al. "Generation of Realistic and High-Quality Appliance Trajectories Based on TimeGAN for Non-intrusive Load Monitoring". In: *2023 6th International Conference on Energy, Electrical and Power Engineering (CEEPE).* 2023, pp. 1500–1504. DOI: https://doi.org/10.1109/CEEPE58418.2023.10166781.

[12] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html (visited on 12/16/2023).

[13] Giovanni Mauro et al. "Generating Mobility Networks with Generative Adversarial Networks". In: *EPJ Data Science* 11.58 (2022). DOI: https://doi.org/10.1140/epjds/s13688-022-00372-4.

[14] Varun Sapre et al. "Synthetic Generation of Traffic Data for Urban Mobility". In: *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* 2018, pp. 2151–2157. DOI: https://doi.org/10.1109/ICACCI.2018.8554633.

[15] Sophie Schaible. "Explorative Datenanalyse Am Beispiel von Mobilitätsdaten Aus GPS-Trackingsystemen". Hamburg: HAW Hamburg, 2023.

[16] Stefan Jansen. *GitHub Repository: machine-learning-for-trading.* URL: https://github.com/stefan-jansen/machine-learning-for-trading/tree/main/21_gans_for_synthetic_time_series (visited on 12/16/2023).

[17] Jinsung Yoon. *Codebase for "Time-series Generative Adversarial Networks (TimeGAN)"*. URL: https://github.com/jsyoon0823/TimeGAN (visited on 12/16/2023).

[18] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-Series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216-bbe2a7d26d490-Paper.pdf (visited on 12/16/2023).

[19] Yukai Liu et al. "NAOMI: Non-autoregressive multiresolution sequence imputation". In: *International Conference on Advances in Neural Information Processing Systems*. Vol. 32. 2019, pp. 11238–11248. URL: https://papers.neurips.cc/paper_files/paper/2019/hash/50c1f44e426560f3f2cdcb3e19e39903-Abstract.html (visited on 12/16/2023).

[20] Yunfei Zhang et al. "Data augmentation for improving heating load prediction of heating substation based on TimeGAN". In: *Energy* 260 (2022). DOI: https://doi.org/10.1016/j.energy.2022.124919.