applied mechanics

MDPI

Article

Evaluation of Different Generative Models to Support the Validation of Advanced Driver Assistance Systems

Manasa Mariam Mammen [1], Zafer Kayatas [1,*] and Dieter Bestle [2]

[1] Mercedes-Benz AG, Kolumbusstr. 19+21, 71063 Sindelfingen, Germany; manasa_mariam.mammen@mercedes-benz.com
[2] Department of Engineering Mechanics and Vehicle Dynamics, Brandenburg University of Technology Cottbus-Senftenberg, Siemens-Halske-Ring 14, 03046 Cottbus, Germany; bestle@b-tu.de
* Correspondence: zafer.kayatas@mercedes-benz.com

Abstract: Validating the safety and reliability of automated driving systems is a critical challenge in the development of autonomous driving technology. Such systems must reliably replicate human driving behavior across scenarios of varying complexity and criticality. Ensuring this level of accuracy necessitates robust testing methodologies that can systematically assess performance under various driving conditions. Scenario-based testing addresses this challenge by recreating safety-critical situations at varying levels of abstraction, from simulations to real-world field tests. However, conventional parameterized models for scenario generation are often resource intensive, prone to bias from simplifications, and limited in capturing realistic vehicle trajectories. To overcome these limitations, the paper explores AI-based methods for scenario generation, with a focus on the cut-in maneuver. Four different approaches are trained and compared: Variational Autoencoder enhanced with a convolutional neural network (VAE), a basic Generative Adversarial Network (GAN), Wasserstein GAN (WGAN), and Time-Series GAN (TimeGAN). Their performance is assessed with respect to their ability to generate realistic and diverse trajectories for the cut-in scenario using qualitative analysis, quantitative metrics, and statistical analysis. Among the investigated approaches, VAE demonstrates superior performance, effectively generating realistic and diverse scenarios while maintaining computational efficiency.

Keywords: advanced driver assistance system; real cut-in maneuver; synthetic data; variational autoencoder; generative adversarial networks; machine learning

check for updates

Received: 7 March 2025
Revised: 14 May 2025
Accepted: 21 May 2025
Published: 27 May 2025

Citation: Mammen, M.M.; Kayatas, Z.; Bestle, D. Evaluation of Different Generative Models to Support the Validation of Advanced Driver Assistance Systems. *Appl. Mech.* **2025**, *6*, 39. https://doi.org/10.3390/applmech6020039

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Automated driving (AD) technology is rapidly evolving, driven by substantial investments from leading technology companies and automotive manufacturers aiming to revolutionize transportation [1]. These advancements are driving the development of vehicles capable of navigating in complex environments autonomously, with automation levels ranging from basic driver assistance (level 1) to full autonomy (level 5) [2]. Level 3 systems, such as Mercedes-Benz's Drive Pilot, can already handle specific driving tasks under defined conditions. The safe deployment of such systems relies on rigorous validation against a diverse array of challenging driving scenarios.

Scenario-based testing has emerged as an important methodology for validating the functionality and safety of Highly Automated Driving (HAD) systems [3,4]. By replicating real-world driving conditions, it provides a structured framework for assessing system performance in safety-critical situations. Traditional scenario-based testing, however, often

relies on parameterized models to describe and simulate driving events [5,6]. While effective for basic scenarios, these models tend to oversimplify the complexity of real-world traffic, failing to capture the variability and unpredictability of actual driving environments. This limitation poses significant issues for thoroughly testing and validating the robustness of AD systems.

In response to these challenges, the automotive industry is increasingly turning to Artificial Intelligence (AI) methods to enhance scenario-based testing [7–9]. AI offers the capability to model complex, real-world scenarios more accurately by learning from large-scale driving data. Unlike traditional methods, AI can generate diverse synthetic scenarios that are both statistically consistent with observed data and reflective of the intricacy of actual driving environments. This approach not only improves the realism of scenario-based testing but also accelerates the development cycle by reducing the modeling effort.

Despite the application of AI techniques for scenario-based testing, generating realistic, diverse, and simulation-ready driving scenarios remains a challenging task. For example, the AE-GAN framework [7] focuses on reconstructing trajectory shapes without modeling full maneuver dynamics. Similarly, the BézierVAE approach [8] relies on drone-based data, limiting its ability to capture ego-centric driving interactions accurately. Multi-trajectory generators like MTG [9] primarily focus on generating short, disconnected sequences. These methods mainly rely on recurrent architectures or parametric curves. However, convolutional neural networks, which have shown strong performance in capturing spatial dependencies, remain underexplored for driving scenario generation.

In [10], the effectiveness of a Variational Autoencoder combined with a convolutional neural network (VAE) is demonstrated for generating realistic and diverse synthetic driving scenarios. This approach proves to be capable of learning and replicating the probabilistic structures of real driving behavior. Building on this foundation, the current study expands the focus by comparing the performance of GAN variations, which have demonstrated their superiority in applications like synthetic image generation and time-series modeling [11–14]. Other approaches, such as Normalizing Flows and Diffusion Models, although powerful for density estimation or sample refinement, are either less effective in capturing multi-modal diversity or computationally intensive for large-scale scenario synthesis. Given these considerations, we select three representative GAN variations that capture different strengths within adversarial learning: a basic Generative Adversarial Network (GAN) [15], known for its simple formulation, Wasserstein GAN (WGAN) [16], which offers better training stability, and Time-Series Generative Adversarial Network (TimeGAN) [14] designed to capture temporal coherence. By applying and evaluating these models in the present study, we aim to identify the most effective approach for generating realistic and diverse driving scenarios.

In this work, the focus is on the cut-in scenario, which is both common in real-world driving and widely regarded as one of the most challenging for scenario generation in simulation testing. The abrupt nature of cut-ins, coupled with their limited reaction time and high variability in driver behavior, makes them particularly difficult to model and replicate realistically. This high-risk event involves a vehicle abruptly entering the ego vehicle's lane, requiring immediate action to avoid collision. It tests the ability of an automated system to adapt to sudden changes in traffic flow and dynamic conditions, making it a crucial benchmark for evaluating the decision-making capabilities. The generative models need to replicate cut-in scenarios while maintaining the temporal coherence and spatial fidelity of measured maneuvers.

The preparation of measured data for cut-in scenarios, as discussed in Section 2, involves filtering large-scale real-world driving datasets to extract relevant events. In particular, the lateral position and longitudinal velocity of vehicles are segmented for

training the generative models and comparison. The explored generative models are detailed in Sections 3–5, while their training details and model efficiency are discussed in Section 6. Sections 7 and 8 cover qualitative and quantitative analyses of the cut-in maneuvers generated by the models across three key dimensions: scenario realism, which evaluates how closely the generated scenarios align with real-world driving data; diversity, ensuring a broad representation of observed driving behaviors; and statistical similarity, which examines the model's ability to capture stochastic properties such as trajectory likelihood and event duration.

2. Measured Cut-In Maneuvers

The models are trained using real-world measurement data collected during test drives. These measurements include Electronic Control Unit (ECU) signals from the ego vehicle and sensor data from surrounding objects, sampled at 50 Hz (0.02 s time interval). These high-resolution data capture critical parameters such as the relative longitudinal and lateral position $(s(t), d(t))$ and absolute speed $v(t)$ of the vehicles in the surrounding, forming a comprehensive basis for scenario modeling and analysis [10].

A representative cut-in scenario is illustrated in Figure 1, where a neighboring vehicle merges laterally into the ego vehicle's lane. To identify such scenarios, a two-step classification framework is implemented, combining rule-based classification with a Time-Series Forest (TSF) machine learning model. In [17], it is shown that TSF outperforms rule-based classification with respect to accuracy and robustness. By combining both methods, the system retains the efficiency of rule-based filtering while improving detection reliability with TSF.

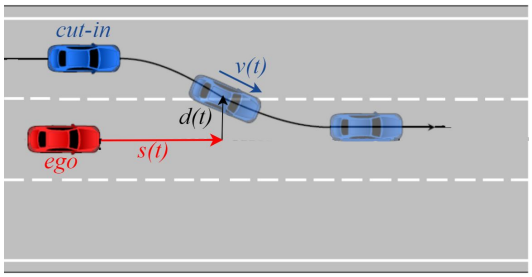


Figure 1. Cut-in scenario (modified figure from [10]).

Before being processed by the TSF classifier, the data undergo preprocessing, like the removal of dropouts and correction of curved roads as outlined in [10]. The final dataset comprises refined time-series data specifically tailored for training AI models to recognize and simulate cut-in scenarios. It consists of N_S extractions, each containing N_t time steps which provide a structured representation of vehicle motion over time. The key variables used in this investigation are the lateral position $d(t)$ and longitudinal velocity $v(t)$, both measured at discrete time points t_k where $k = 1 \ldots N_t$. Each extraction i forms a collection of measurements represented as

$$(t_k^{(i)}, d_k^{(i)}, v_k^{(i)}), \quad i = 1, \ldots, N_S, k = 1, \ldots, N_t. \tag{1}$$

To separately analyze $d(t)$ and $v(t)$, their measurements are summarized in feature vectors

$$\mathbf{x}_i = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \ldots & x_{N_t}^{(i)} \end{bmatrix}^\top \in \mathbb{R}^{N_t}, \quad x \in \{d, v\}, i = 1, \ldots, N_S. \tag{2}$$

Figure 2 illustrates heatmaps of measured lateral positions $\mathbf{d}_i$ and velocities $\mathbf{v}_i$ for cut-in scenarios, derived from over 8000 real-world cut-in trajectories recorded by a vehicle fleet. The lateral position heatmap highlights high-density regions where cut-in maneuvers

are the most frequently observed. These maneuvers typically start at a lateral offset of 3 to 4 m from the center of the ego vehicle's lane, corresponding to the standard lane width on German highways. As the maneuver progresses, the vehicles perform a lane change and eventually stabilize near the center of the ego's lane. Similarly, the velocity heatmap reveals that most cut-in events occur at speeds ranging from 20 m/s to 35 m/s, reflecting typical highway driving conditions.

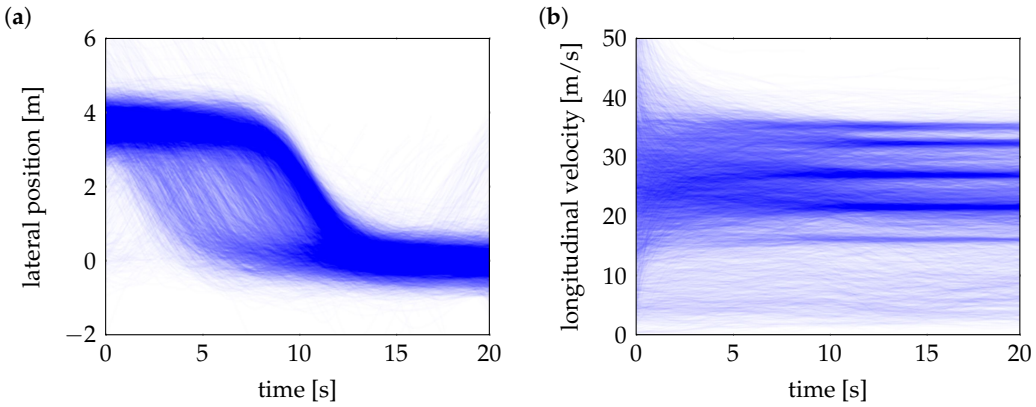


Figure 2. Heatmaps of (**a**) measured lateral position $d(t)$ and (**b**) longitudinal velocity $v(t)$ for cut-in scenarios.

For training generative models, only a randomly selected 25% part of these measured cut-in maneuvers (approximately 2000 trajectories), as shown in Figure 3, is used. While initial experiments used a larger training split, the final experiments deliberately use a reduced subset to simulate a more challenging data-scarce environment. This setting allows to better evaluate the robustness and generalization capabilities of the generative models under limited data conditions, which are common in real-world autonomous driving applications. During evaluation, each model generates the same number of synthetic trajectories as the full set of measured cut-in maneuvers, and the generated scenarios are then compared to the original dataset to assess how accurately the models replicate real-world distributions.

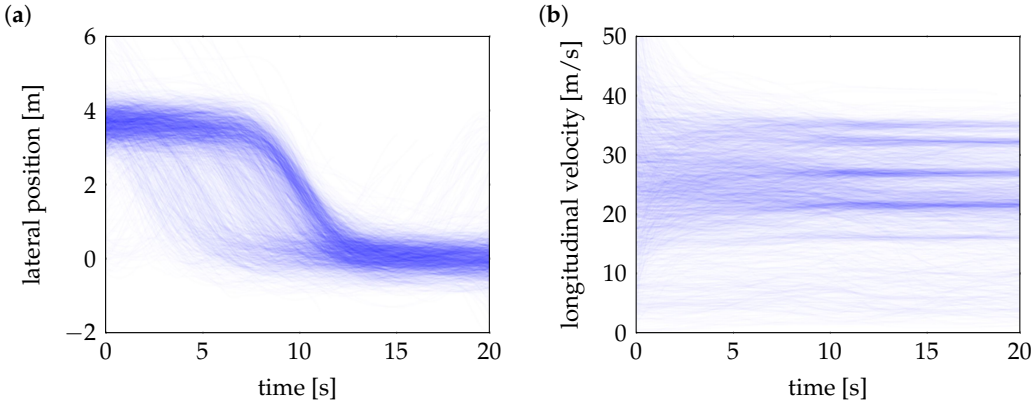


Figure 3. Heatmaps of randomly selected cut-in maneuvers used for training the generative AI models: (**a**) lateral position $d(t)$ and (**b**) longitudinal velocity $v(t)$.

3. Variational Autoencoder

Before introducing specific generative models, it is essential to clarify their primary goal. Generative models aim not just to replicate or approximate a given dataset but to learn the underlying probability distribution $p_x(\mathbf{x})$ of the measured data to enable the creation of new, diverse, and realistic maneuvers $\tilde{\mathbf{x}}_j$ from the learned generator probability distribution

$p_g(\tilde{\mathbf{x}})$. The generated maneuvers capture the essential statistical properties and variability present in the original data if $p_g \approx p_x$. In this work, the models are trained on observed cut-in maneuvers from real-world driving and learn to generate an arbitrary number of synthetic but realistic lateral positions $\tilde{\mathbf{d}}_j$ and longitudinal velocities $\tilde{\mathbf{v}}_j$ of trajectories that exhibit similar behavioral patterns to those seen in the measured data.

A VAE is a generative model designed to learn a lower-dimensional latent representation of data, enabling controlled and diverse sample generation. Unlike standard autoencoders, which deterministically encode and decode input data, VAEs model the latent space as a probability distribution rather than a fixed set of feature vectors [18]. The VAE consists of an encoder that maps high-dimensional inputs $\mathbf{x} \in \mathbb{R}^D$ onto a low-dimensional latent space $\mathbf{z} \in \mathbb{R}^d$, $d \ll D$ and a decoder that maps the latent space representations back into the original space, generating an approximate example $\tilde{\mathbf{x}} \in \mathbb{R}^D$; see Figure 4.

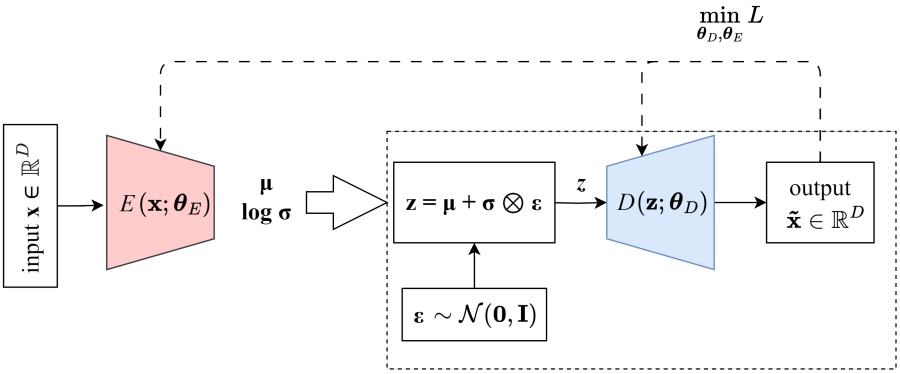


Figure 4. Architecture of VAE with the framed part to be used as generative model (modified figure from [10]).

The encoder predicts the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$ of the sampled latent representation $\mathbf{z}$. It learns to approximate the posterior distribution $q_{\boldsymbol{\theta}_E}(\mathbf{z}|\mathbf{x})$, ensuring that the latent variables retain a meaningful structure. The decoder, parameterized by $\boldsymbol{\theta}_D$, reconstructs an estimate $\tilde{\mathbf{x}}$ of the measured input $\mathbf{x}$ from the sampled latent variable by modeling the likelihood function $p_{\boldsymbol{\theta}_D}(\tilde{\mathbf{x}}|\mathbf{z})$. This probabilistic formulation ensures that the learned latent space remains continuous and structured, allowing for smooth data interpolation and the synthesis of diverse yet coherent maneuvers. The VAE optimizes two key objectives: minimizing the reconstruction loss to ensure accurate data reconstruction and enforcing a prior distribution on the latent space, defined as $p_z(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. The reconstruction loss is given by

$$MSE = \frac{1}{N_s} \sum_{i=1}^{N_s} \|\tilde{\mathbf{x}}_i - \mathbf{x}_i\|^2. \tag{3}$$

To enforce the prior, VAE minimizes the Kullback–Leibler (KL) divergence between the approximate posterior $q(\mathbf{z}|\mathbf{x})$ and the prior $p_z(\mathbf{z})$, leading to the total loss [19]

$$L = MSE + D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x}) \parallel p_z(\mathbf{z})). \tag{4}$$

In practice, balancing reconstruction accuracy and latent space regularization is crucial. Standard VAEs weigh these terms equally, but in some cases, the KL divergence term dominates early training, leading to poor reconstructions. To address this, the β-VAE framework [20] introduces a weighting parameter β, adjusting the trade-off between reconstruction and regularization, i.e.,

$$L = MSE + \beta D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x}) \parallel p_z(\mathbf{z})). \tag{5}$$

A higher β encourages better latent space organization at the expense of reconstruction accuracy, while a lower β prioritizes reconstruction fidelity. This makes β a key hyperparameter when designing VAEs for structured generative modeling.

Here, the VAE follows the architecture discussed in [10], utilizing convolutional layers for both the encoder and decoder to effectively process time-series data; see Table 1. The encoder progressively reduces the spatial dimensions of the input (1) and (2) from $D = N_t = 100$ through successive convolutional layers to a compact latent representation with $d = 10$ ensuring that the model captures the most salient features of the input while avoiding redundancy. The decoder uses transposed convolutions to reconstruct D-dimensional trajectories.

Table 1. Network topology of the VAE model using convolutional neural networks.

Layer	**Output**
(a) Encoder	
Input	100×3
Conv, ReLU	100×25
Conv, ReLU	100×50
Conv, ReLU	100×100
Flatten	10,000
Dense (2x), ReLU	10
Lambda	10
(b) Decoder	
Input	10
Dense, ReLU	1000
Reshape	100×100
DeConv, ReLU	100×50
DeConv, ReLU	100×25
Flatten	2500
Dense	300
Reshape	100×3

Once the VAE model is trained, it learns the distribution parameters of $\mathbf{z}$, i.e., mean $\boldsymbol{\mu}$ and the natural logarithm of variance σ^2 for each of the 10 latent parameters. It should be noted that $\boldsymbol{\mu}$ and σ^2 are not constants but follow individual probability distributions; see [10]. To generate a new cut-in maneuver, independent samples $\boldsymbol{\varepsilon}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are drawn from Gaussian distributions, and $\boldsymbol{\mu}_j$ and $\boldsymbol{\sigma}_j^2$ are picked randomly from their individual distribution described by Gaussian mixture distributions. These quantities are then combined as Hadamard (element-wise) product $\mathbf{z}_j = \boldsymbol{\mu}_j + \boldsymbol{\sigma}_j \odot \boldsymbol{\varepsilon}_j$, where each latent variable is computed as

$$z_i^{(j)} = \mu_i^{(j)} + \sigma_i^{(j)} \varepsilon_i^{(j)}, \quad i = 1, 2, \ldots, 10. \tag{6}$$

The resulting latent vector $\mathbf{z}_j = [z_1^{(j)}, z_2^{(j)}, \ldots, z_{10}^{(j)}]$, is then passed through the decoder to generate a synthetic maneuver trajectory $\tilde{\mathbf{x}}_j = D(\mathbf{z}_j)$, see framed part in Figure 4. This describes a cut-in trajectory $\tilde{x}_k^{(j)}$, $k = 1 \ldots N_t$. By sampling different $\mathbf{z}_j$ vectors, we can generate diverse and realistic maneuver trajectories while preserving the underlying distribution learned during training.

4. Generative Adversarial Networks

A GAN as shown in Figure 5 consists of two neural networks: a generator (G), similar to the decoder of the VAE, and a discriminator (D), which learns to distinguish between real data and fake data generated by $G(z)$ from random input. Both are trained in a competitive

learning process [15]. The generator aims to synthesize data that closely resemble real samples, while the discriminator classifies samples as real or fake. This competition forces the generator to iteratively refine its outputs until the discriminator can no longer reliably differentiate between them.

In this research, we explore both a basic GAN and a WGAN to evaluate their ability to model the underlying data distribution. The basic GAN provides a standard framework for adversarial training, while the WGAN introduces modifications to improve training stability and reduce mode collapse. The following subsections outline the architectures and training formulations of these models.

4.1. Basic GAN

The basic GAN architecture, illustrated in Figure 5, consists of two neural networks: a generator and a discriminator. The generator transforms a random vector $\mathbf{z}_j \in \mathbb{R}^d$, sampled from a prior distribution $p_z(\mathbf{z})$ typically chosen as a standard Gaussian distribution, into a generated cut-in maneuver $\tilde{\mathbf{x}}_j = G(\mathbf{z}_j) \in \mathbb{R}^D$, where $D = N_t$. Meanwhile, real cut-in maneuvers $\mathbf{x}_i$ are selected from the true measured data with the unknown distribution $p_x(\mathbf{x})$. The discriminator assigns a probability $D(\mathbf{x}) \in [0, 1]$ to a provided sample, indicating how likely it is to be real. Its objective is to distinguish between real and generated maneuvers, aiming for $D(\mathbf{x}_i) \to 1$ for real and $D(\tilde{\mathbf{x}}_j) \to 0$ for generated ones [21].

To achieve this, the discriminator maximizes the probability of classifying real trajectories correctly, i.e.,

$$\max_D \mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[\log D(\mathbf{x})]. \tag{7}$$

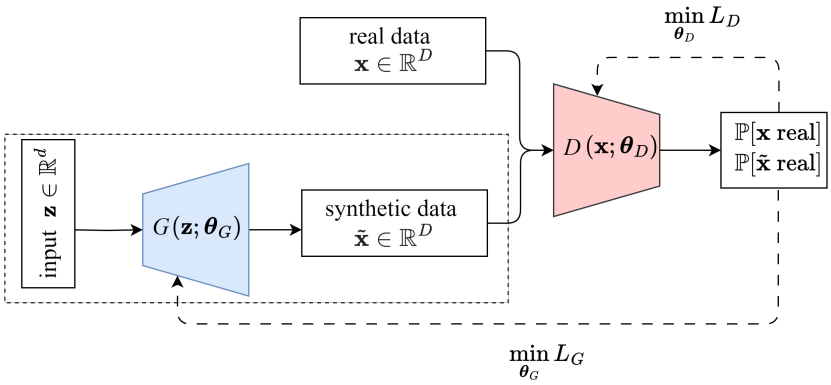


Figure 5. Architecture of basic GAN with framed generator part.

For generated trajectories $G(\mathbf{z})$, it aims to classify them as fake. Due to $\log(1 - D) \to 0$ as $D \to 0$, this is achieved by maximizing the expectation

$$\max_D \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{8}$$

Both objectives can be summarized in a common goal for the discriminator:

$$\max_D \Big(\mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \Big). \tag{9}$$

Since neural networks are optimized using gradient descent (which minimizes rather than maximizes), the discriminator loss function is usually defined as the negative of objective (9), i.e.,

$$L_D = -\mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[\log D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{10}$$

In contrast, the generator aims to fool the discriminator by generating trajectories that are classified as real, which may be expressed as

$$\max_G \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log D(G(\mathbf{z}))]. \tag{11}$$

For a minimization algorithm the generator loss function is then

$$L_G = -\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log D(G(\mathbf{z}))]. \tag{12}$$

Both the generator and discriminator are designed using convolutional layers to effectively capture spatial structures in the data, which are in the form of time series in the actual application. The generator employs transposed convolutions to upsample low-dimensional Gaussian noise $\mathbf{z} \in \mathbb{R}^d$, $d = 10$, into structured high-dimensional data $\tilde{\mathbf{x}} \in \mathbb{R}^D$, $D = N_t > d$, ensuring realistic feature generation. The discriminator, in contrast, applies standard convolutions to extract hierarchical features, enabling accurate differentiation between real and generated samples. The detailed architectural specifications of the used GAN are summarized in Table 2. Maximization $\max_D$ and $\max_G$ in Equations (9) and (11) refers to training the corresponding convolutional neural network parameters $\boldsymbol{\theta}_D$ and $\boldsymbol{\theta}_G$.

Table 2. Network topology of the GAN model using convolutional neural networks.

Layer	**Output**
(a) Generator.	
Input	10
Dense, ReLU	800
Reshape	25×32
DeConv, ReLU	50×64
DeConv, ReLU	100×32
Conv, tanh	100×3
(b) Discriminator.	
Input	100×3
Conv, ReLU	50×32
Conv, ReLU	25×64
Flatten	1600
Dense, ReLU	32
Dense, sigmoid	1

For generating new cut-in maneuvers, only the framed part in Figure 5 is applied. More specifically, a 10-dimensional random vector $\mathbf{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is picked from a standard Gaussian distribution and fed to the generator, resulting in a vector $\tilde{\mathbf{x}}_j = G(\mathbf{z}_j)$ which represents a synthetic cut-in maneuver according to Equation (1). Throughout the evaluations, we refer to the basic GAN simply as GAN.

4.2. Wasserstein GAN

The basic GAN suffers from several challenges, including training instability, mode collapse, and vanishing gradients. These issues arise from the binary classification nature of the discriminator, which can lead to saturated gradients when it becomes too strong, making it difficult for the generator to improve. To improve stability, WGAN modifies the adversarial training framework by replacing the discriminator with a critic that learns to approximate the Wasserstein distance between real and generated distributions [16]; see Figure 6. Unlike a discriminator, which assigns a probability to each trajectory, the critic produces a real-valued score, indicating how real or fake a trajectory is. The Wasserstein

distance provides meaningful gradients even when the critic is well trained or when generated trajectories are very poor, which prevents vanishing gradients and mode collapse, resulting in more stable learning dynamics.

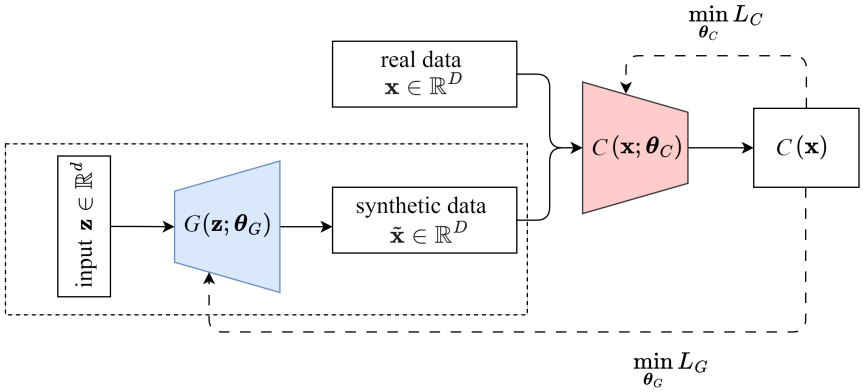


Figure 6. Architecture of WGAN with framed generator part.

The Wasserstein distance between the real data distribution $p_x(\mathbf{x})$ and $p_g(\tilde{\mathbf{x}})$ of generated trajectories is given by [16]

$$W(p_x, p_g) = \inf_{\gamma \in \Gamma(p_x, p_g)} \mathbb{E}_{(\mathbf{x},\tilde{\mathbf{x}})\sim\gamma}[\|\mathbf{x} - \tilde{\mathbf{x}}\|], \tag{13}$$

where $\Gamma(p_x, p_g)$ represents the set of all joint distributions $\gamma(\mathbf{x}, \tilde{\mathbf{x}})$ whose marginals are p_x and p_g. The inf (infimum) operator finds the lowest possible expected transport cost required to morph one distribution into another. Since solving the above equation directly is intractable, the Kantorovich–Rubinstein duality reformulates the Wasserstein distance into an easier-to-optimize form [22]

$$W(p_x, p_g) = \sup_{\|f_w\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[f_w(\mathbf{x})] - \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g(\tilde{\mathbf{x}})}[f_w(\tilde{\mathbf{x}})]. \tag{14}$$

Here, sup (supremum) finds the maximum value over all possible 1-Lipschitz functions f_w. This allows us to approximate the Wasserstein distance by maximizing the difference in critic scores between real and generated trajectories:

$$\max_C \mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[f_w(\mathbf{x})] - \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g(\tilde{\mathbf{x}})}[f_w(\tilde{\mathbf{x}})], \tag{15}$$

which ensures that the score for real trajectories remains higher than that for generated ones, guiding the generator to produce more realistic cut-in maneuvers. The Wasserstein metric f_w is generated by the critic $C(\mathbf{x};\boldsymbol{\theta}_C)$ to be trained. The maximization is then substituted by the minimization of the loss function

$$L_C = -\mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[C(\mathbf{x})] + \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g(\tilde{\mathbf{x}})}[C(\tilde{\mathbf{x}})]. \tag{16}$$

Originally, the Lipschitz constraint was enforced by clipping weights, which can lead to poor gradient flow and suboptimal performance. To address this, Wasserstein GAN with Gradient Penalty (WGAN-GP) introduces a gradient penalty that better enforces the Lipschitz condition [23]. Instead of weight clipping, the gradient penalty is applied to interpolated points $\hat{\mathbf{x}}$, which are sampled along straight lines between real and generated data points. This ensures that the critic function maintains a smooth gradient norm

close to 1, improving stability and performance. The improved optimization objective is formulated as

$$L_C = -\mathbb{E}_{\mathbf{x}\sim p_x(\mathbf{x})}[C(\mathbf{x})] + \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g(\tilde{\mathbf{x}})}[C(\tilde{\mathbf{x}})] + \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{x}}(\hat{\mathbf{x}})}\left(\|\nabla_{\hat{\mathbf{x}}} C(\hat{\mathbf{x}})\|_2 - 1\right)^2. \tag{17}$$

The generator aims to produce trajectories that receive higher scores from the critic. This is achieved by maximizing

$$\max_G \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g(\tilde{\mathbf{x}})}[C(\tilde{\mathbf{x}})] = \max_G \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[C(G(\mathbf{z}))], \tag{18}$$

which is reformulated as a minimization problem with the generator loss function

$$L_G = -\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[f_w(G(\mathbf{z}))]. \tag{19}$$

By leveraging the Wasserstein distance, WGAN already improves training stability compared to basic GAN and reduces mode collapse, while WGAN-GP further enhances convergence and robustness by addressing gradient-related issues. The architecture, detailed in Table 3, is the same for both WGAN and WGAN-GP and closely resembles the GAN in Table 2. However, in this work, WGAN-GP is chosen for training, which we will refer to as WGAN for simplicity. Also, the application of the framed generator part in Figure 6 is identical as discussed in Section 4.1.

Table 3. Network topology of the WGAN model using convolutional neural networks.

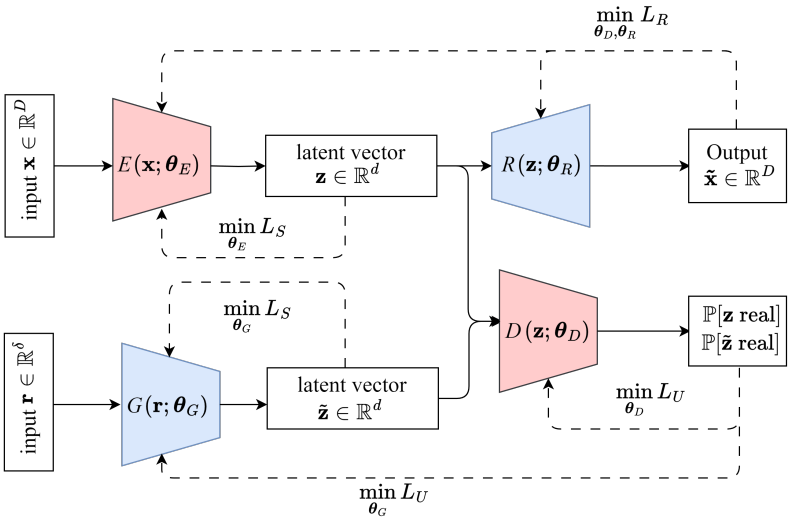
Layer	**Output**
(a) Generator.	
Input	10
Dense, ReLU	800
Reshape	25×32
DeConv, ReLU	50×64
DeConv, ReLU	100×32
Conv, tanh	100×3
(b) Critic.	
Input	100×3
Conv, LeakyReLU	50×64
Conv, LeakyReLU	25×128
Flatten	3200
Dense	1

5. Time-Series Generative Adversarial Network

Time-Series Generative Adversarial Network (TimeGAN) is a generative model especially designed for sequential data, integrating both supervised and unsupervised learning to capture complex temporal dependencies [14]. Unlike traditional GANs generating independent samples, TimeGAN explicitly preserves the underlying temporal structure, making it particularly effective and exciting for generating realistic time-series sequences. The model consists of four key components: an embedder E, a recovery network R, a generator G, and a discriminator D. See Figure 7a. The embedder encodes real input trajectories $\mathbf{x}_i \in \mathbb{R}^D, D = N_t$, into a structured latent representation $\mathbf{z}_i = E(\mathbf{x}_i) \in \mathbb{R}^d$, while the recovery network reconstructs the original trajectories $\tilde{\mathbf{x}}_i = R(\mathbf{z}_i) \in \mathbb{R}^D$ from the latent space to ensure that the temporal relationships are preserved. The generator synthesizes new latent representations $\tilde{\mathbf{z}}_j = G(\mathbf{r}_j) \in \mathbb{R}^d$ from random input $\mathbf{r}_j \in \mathbb{R}^\delta$ taken from a prior distribution $\mathbf{r}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ which is chosen as standard Gaussian distribution. The

discriminator distinguishes between real and generated latent representations, enforcing realism in the generated latent vectors $\mathbf{z}_j$. By combining these components, TimeGAN jointly learns an embedding space for sequential data while simultaneously training a generator and discriminator in an adversarial setting. This hybrid approach allows the model to retain long-term dependencies while ensuring that generated sequences follow realistic temporal patterns.

(**a**)



(**b**)

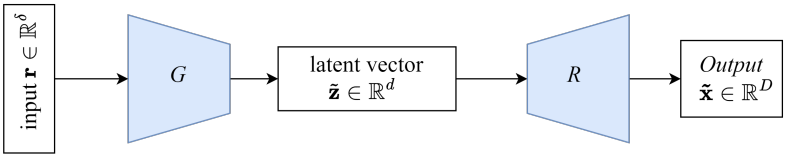


Figure 7. Architecture of TimeGAN for (**a**) training and (**b**) inference.

The training process consists of two phases: embedding learning and adversarial training. In the first phase, the embedder and recovery network are optimized to learn a structured latent representation. Given real measured trajectories $\mathbf{x} \sim p_x(\mathbf{x})$, the embedder maps them to latent representations $\mathbf{z} = E(\mathbf{x})$, and the recovery network reconstructs the original sequence as $\tilde{\mathbf{x}} = R(\mathbf{z})$. Minimization of the reconstruction loss [14]

$$L_R(\theta_E, \theta_R) = \mathbb{E}[\|\mathbf{x} - \tilde{\mathbf{x}}\|^2]. \tag{20}$$

ensures that the recovered sequences closely match the original ones. To enforce temporal structure in the generated latent representations, TimeGAN introduces a supervised loss L_s, which aligns the latent space $\mathbf{z} = E(\mathbf{x})$ learned by the embedder with the latent space $\tilde{\mathbf{z}} = G(\mathbf{r})$ learned by the generator. This is performed by minimizing the loss expectation

$$L_s(\theta_E, \theta_G) = \mathbb{E}[\|\mathbf{z} - \tilde{\mathbf{z}}\|^2]. \tag{21}$$

Both losses are combined and minimized together with respect to the embedder, recovery, and generator network parameters:

$$\min_{\theta_E, \theta_R, \theta_G} (\lambda L_s + L_r), \tag{22}$$

where λ is a hyperparameter that controls the balance between supervised and reconstruction loss. This step ensures that the latent space remains structured, facilitating stable adversarial training.

In the second phase, adversarial training is applied to improve the quality of generated trajectories. The discriminator attempts to classify whether a given latent representation is real $\mathbf{z}$ or generated $\tilde{\mathbf{z}}$. The adversarial loss for improving the discriminator is defined similar to Equation (10), however, with the latent vector $\mathbf{z}$ as input:

$$L_u(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G) = -\mathbb{E}[\log D(\mathbf{z})] - \mathbb{E}[\log(1 - D(\tilde{\mathbf{z}}))], \tag{23}$$

where $\mathbf{z} = E(\mathbf{x})$ represents the real latent representation, and $\tilde{\mathbf{z}} = G(\mathbf{r})$ represents the generated latent representation. The generator G is trained to minimize a combination of the supervised loss (21) and the adversarial loss (23), i.e.,

$$\min_{\boldsymbol{\theta}_G}(\eta L_s + L_u), \tag{24}$$

where η is a hyperparameter that controls the trade-off between supervised learning and adversarial learning. Meanwhile, the discriminator D is trained to maximize the adversarial loss (23) with respect to discriminator parameters $\boldsymbol{\theta}_D$, i.e.,

$$\max_{\boldsymbol{\theta}_D} L_u. \tag{25}$$

Unlike traditional GANs that rely purely on adversarial feedback, TimeGAN explicitly aligns the latent representations learned from real and generated data through L_s. This ensures that the generator produces meaningful latent sequences rather than simply fooling the discriminator. The hyperparameters λ and η in Equations (22) and (24) control the trade-offs between supervised, reconstruction, and adversarial components but are generally robust in practice. Typically, they are set to $\lambda = 1$ and $\eta = 10$, also used here.

TimeGAN was originally implemented with Recurrent Neural Network (RNN) layers. However, here it is adapted to convolutional neural network-based architectures for consistency with the other models. The detailed structure of the generator, discriminator, embedder, and reconstructor networks is given in Table 4. For generating new cut-in maneuvers, random vectors $\mathbf{r}_j \in \mathbb{R}^\delta, \delta = 50$ are picked from a standard Gaussian distribution, transformed into the latent space by the generator G, and finally transformed into synthetic maneuvers $\tilde{\mathbf{x}}_j$ by the recovery network R; see Figure 7b.

Table 4. Network topology of the TimeGAN model using convolutional neural networks.

Layer	**Output**
(a) Generator.	
Input	50
Dense, ReLU	800
Reshape	25×32
DeConv, ReLU	50×64
DeConv, ReLU	100×32
Conv, tanh	100×10
(b) Discriminator.	
Input	100×10
Conv, ReLU	50×32
Conv, ReLU	25×64

Table 4. *Cont.*

Layer	Output
Flatten	1600
Dense, ReLU	32
Dense, sigmoid	1
(c) Embedder.	
Input	100×3
Conv, ReLU	100×100
Conv, ReLU	100×50
Conv, ReLU	100×25
Dense, ReLU	100×10
(d) Reconstructor.	
Input	100×10
Dense, ReLU	100×25
DeConv, ReLU	100×50
DeConv, ReLU	100×100
DeConv, ReLU	100×200
Dense, tanh	100×3

6. Parameter Training and Computation Time

All models are trained on a workstation with a 6-core CPU and a Quadro T2000 GPU (8GB VRAM). Training configurations are adjusted based on model complexity and convergence behavior to ensure stable learning. Due to their adversarial nature, GAN-based models require more training epochs than VAE. Therefore, the total training duration varies across the models. VAE is trained for 1000 epochs, while GAN and WGAN require 1500 epochs to achieve stable loss values. Due to its higher iteration requirements, TimeGAN is trained for 2000 epochs. To prevent overfitting, early stopping is applied, halting training once validation loss plateaus.

Hyperparameters, including batch size, learning rate, and optimizer, are tuned using grid search across all models. Additionally, model-specific parameters are optimized: the update ratio and gradient penalty weight (λ) for WGAN, and the KL divergence loss weight for VAE. A batch size of 32 is used for all models. ADAM optimization is applied to VAE with a learning rate of 0.00001 ($\beta_1 =$ default, $\beta_2 =$ default) and a KL divergence loss weight of 0.001, GAN with a learning rate of 0.0002 ($\beta_1 = 0.5, \beta_2 =$ default), and TimeGAN with a learning rate of 0.0001 ($\beta_1 =$ default, $\beta_2 =$ default). RMSprop is used for WGAN with a learning rate of 0.00005 to improve stability. For WGAN, we use a generator-to-critic training ratio of 1:3 and apply a gradient penalty with $\lambda = 10$ to enforce the Lipschitz constraint.

As shown in Table 5, VAE has the shortest training time, completing its 500 epochs significantly faster than the other models. However, its inference time is more than twice that of GAN-based models. In contrast, GAN and WGAN require much longer training durations, taking approximately 17 times longer than VAE, but achieve inference speeds over 60% faster. TimeGAN is the least efficient model, with a training time over six times longer than GAN and WGAN and an inference time more than eleven times that of VAE.

Table 5. Comparison of model training time and inference efficiency.

Model	Training Time	Inference Time
VAE	**10m (500 epochs)**	11.54 ms
GAN	2 h 48 m (1500 epochs)	4.64 ms
WGAN	2 h 55 m (1500 epochs)	**4.48 ms**
TimeGAN	19 h 45 m (2000 epochs)	129 ms

7. Qualitative Analysis of Generated Cut-In Maneuvers

A qualitative evaluation may be performed by using heatmaps, t-Distributed Stochastic Neighbor Embedding (t-SNE), or kernel density estimation (KDE). The goal is to gain insight into the data distribution and the model's ability to capture underlying patterns. Heatmaps are used to examine structural similarities, capture spatial and temporal variations, and to highlight alignment and divergence between synthetic and real maneuvers. t-SNE is a nonlinear dimension reduction technique that maps high-dimensional data into a lower-dimensional space, typically two or three dimensions, while preserving neighborhood relationships. This mapping enables the representation of the N_t-dimensional data $\mathbf{d}_i$ and $\mathbf{v}_i$ in 2D, allowing for a visual assessment of how well the generated samples resemble real data. To complement these visual analyses, KDE plots provide a statistical comparison of data distributions. KDE estimates the probability density function of a dataset, offering a smooth representation that helps to evaluate how closely generated samples match the statistical properties of the real dataset. By jointly analyzing heatmaps, t-SNE plots, and KDE plots, we obtain a comprehensive qualitative and statistical evaluation of the generated data, and thus a first assessment of the underlying generative models.

7.1. Heatmaps

The full dataset of over 8000 measured trajectories for lateral position $\mathbf{d}_i$ and longitudinal velocity $\mathbf{v}_i$ is visualized in Figure 2, while their equal number of generated counterparts denoted as $\tilde{\mathbf{d}}_i$ and $\tilde{\mathbf{v}}_i$ are shown in Figure 8. A visual comparison of heatmaps already allows to clearly assess how well the models preserve the underlying temporal structure and dynamics.

Among the evaluated models, VAE demonstrates the best performance, generating trajectories that closely align with the measured data. As shown in Figure 8a, its trajectories exhibit smooth transitions and accurately capture underlying patterns, ensuring high trajectory realism. The basic GAN in Figure 8b performs reasonably well, producing less noisy outputs compared to the other GAN-based models in Figure 8c,d. However, its trajectories remain less smooth and structured than those of VAE, particularly in velocity. The WGAN exhibits the greatest irregularity with high noise level, reducing the overall trajectory realism. TimeGAN also generates noisy trajectories but performs slightly better than WGAN. Due to the noisy trajectories, all three models except the VAE require post-processing to ensure stability before being used in simulation testing, whereas the trajectories generated by the VAE are sufficiently smooth for direct use. The noise also makes it difficult to clearly distinguish different probabilistic modes through direct trajectory observation, necessitating deeper analysis using techniques such as t-SNE and kernel density estimation (KDE) to properly assess the diversity and structural coverage of the generated scenarios.

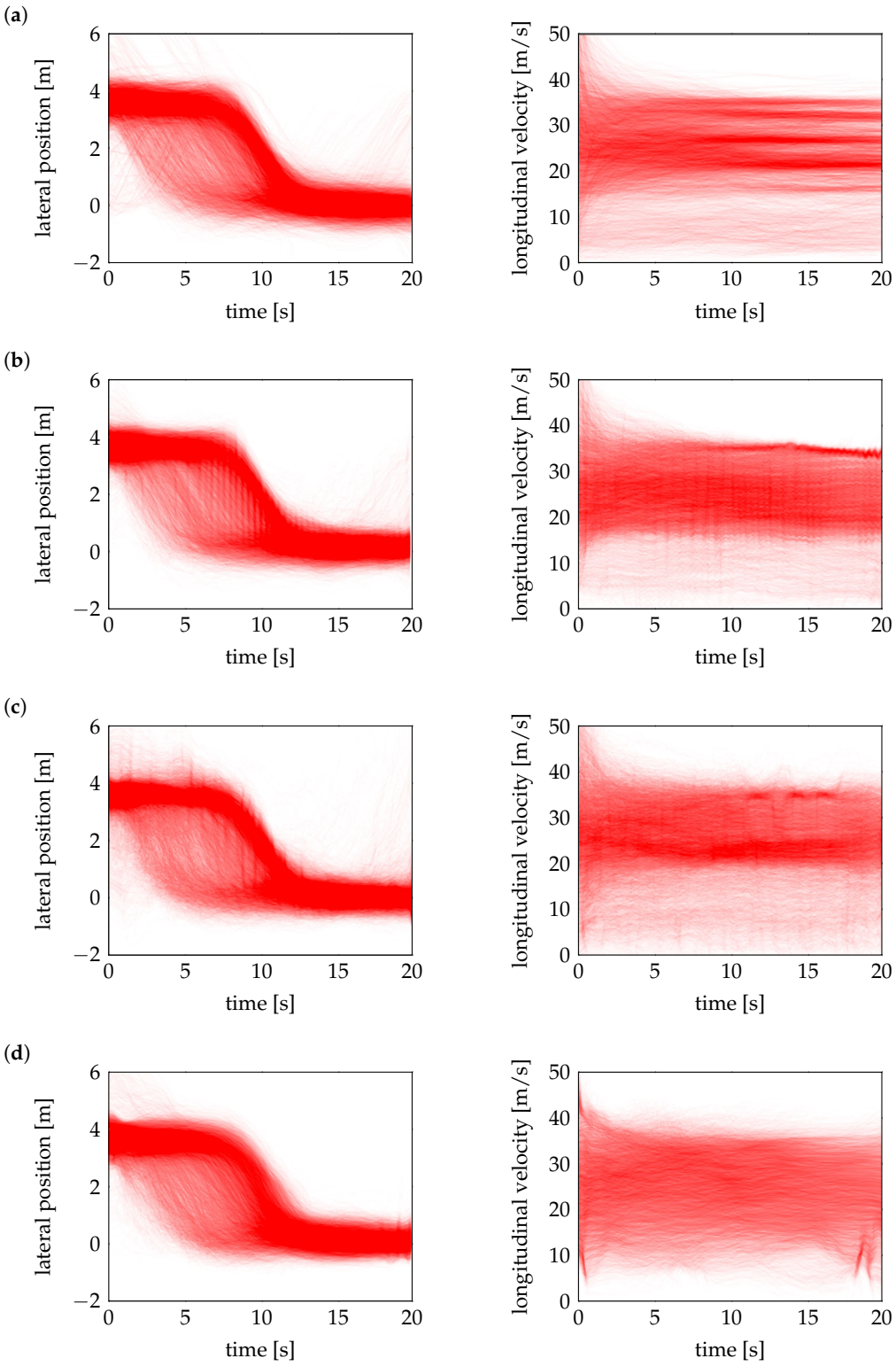


Figure 8. Heatmaps of lateral position (left) and longitudinal velocity (right) of cut-in maneuvers generated by (**a**) VAE, (**b**) GAN, (**c**) WGAN, and (**d**) TimeGAN.

7.2. Distributed Stochastic Neighbor Embedding

To assess the similarity between different trajectories, we compare their distances in a structured manner. By introducing the feature vector (2), each cut-in trajectory is represented as a point in the N_t-dimensional feature space, where its relative distance

$$\Delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\sum_{k=1}^{N_t} (x_k^{(i)} - x_k^{(j)})^2}, \quad i, j = 1 \dots N_S, \tag{26}$$

to other trajectories $\mathbf{x}_j$ provides insight into its similarity. If two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are close in the feature space, their associated maneuvers $x_k^{(i)}$ and $x_k^{(j)}$ exhibit similar movement patterns. However, the direct visualization of these neighborhood relationships in the high-dimensional space is not feasible (as here N_t = 100), making it necessary to map the data into a two-dimensional representation while preserving neighborhood structures.

Such a mapping can be achieved by using t-SNE [24], which is an improved version of SNE [25]. While SNE is assuming Gaussian distribution in both spaces, i.e, the original **x**-space and the mapped **y**-space, t-SNE assumes a t-distribution in the **y**-space. The transformation

$$\gamma : \mathbb{R}^{N_t} \to \mathbb{R}^2, \quad \mathbf{y} = \gamma(\mathbf{x}), \tag{27}$$

is applied independently to real and generated datasets, mapping their high-dimensional representations into a two-dimensional space, which then allows to visualize trajectory distributions for each dataset separately and compare their overall patterns.

The t-SNE computes a probability distribution of the distances (26) using a Gaussian kernel:

$$p_{ij} = \frac{\exp(-\Delta_{ij}^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\Delta_{ik}^2 / 2\sigma_i^2)}, \tag{28}$$

where higher values of p_{ij} indicate stronger similarity between trajectories $\mathbf{x}_i$ and $\mathbf{x}_j$. The variance σ_i^2 is determined individually for each point, adapting to local density variations. Acting as a bandwidth parameter for the Gaussian kernel, it dynamically scales distances to control how local or global the similarity assessment is. In dense regions, a smaller σ_i focuses on immediate neighbors, while in sparse regions, a larger σ_i broadens the influence to distant points. The value of σ_i is adjusted such that the effective number of neighbors each point considers remains approximately constant.

In the two-dimensional visualization space, the same concept is applied, but instead of a Gaussian kernel, a Student's t-distribution with one degree of freedom is used, which is equivalent to a Cauchy distribution and provides better results:

$$q_{ij} = \frac{(1 + \delta_{ij}^2)^{-1}}{\sum_{k \neq i} (1 + \delta_{ik}^2)^{-1}}, \tag{29}$$

where $\delta_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$ represents the Euclidean distance in the two-dimensional **y**-space. Due to normalization, both quantities p_{ij} and q_{ij} can be interpreted as observations of associated probability distributions P and Q in both spaces. The two point clouds $\mathbf{x}_i$ and $\mathbf{y}_i$ have the same neighbor characteristics if their corresponding distributions (28) and (29) are identical. A possible measure of the similarity of the two probability distributions is the Kullback–Leibler (KL) divergence defined as

$$C = D_{\mathrm{KL}}(P||Q) = \sum p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{30}$$

Similarity is enforced by moving the mapped points $\mathbf{y}_i$ such that C is minimized where a gradient search with a momentum term is used as the training procedure to finally obtain the mapping (27).

The application of t-SNE to the measured maneuvers (2) results in the point clouds $\mathbf{y}_i$ illustrated in Figure 9. Similarly, t-SNE is applied separately to the generated maneuvers $\tilde{\mathbf{x}}_i$ in Figure 8 yielding the point clouds $\tilde{\mathbf{y}}_i$ shown in Figure 10. By comparing them with Figure 9, the performance of the generative models may be assessed. If the generated clustering patterns align closely with those of the measured data in the two-dimensional space, this indicates that the model has successfully captured the underlying structure of the measured cut-in maneuvers.

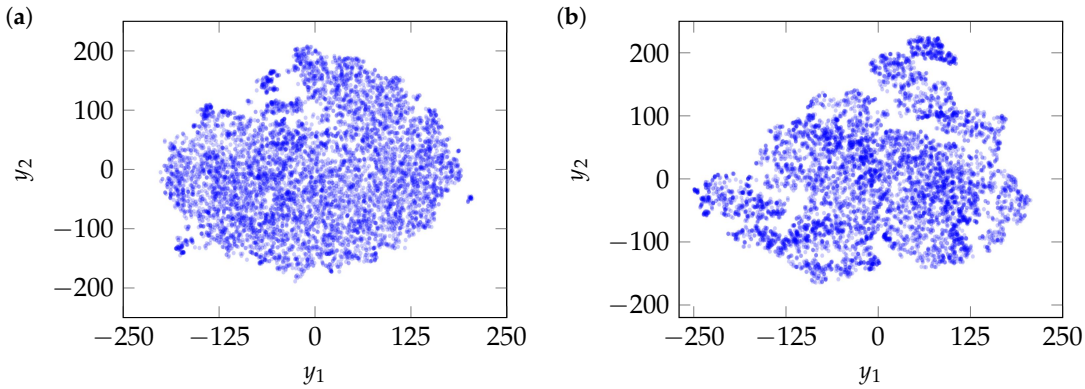


Figure 9. Two-dimensional t-SNE maps of (**a**) lateral position and (**b**) longitudinal velocity of measured cut-in maneuvers.

VAE and WGAN demonstrate comparable performance, effectively covering the entire mapped measured data space and preserving the cluster structure as shown in Figure 10a,c. The former suggests a strong capture of diversity, while the latter indicates similarity in maneuver distributions. GAN and TimeGAN also perform well. However, both models miss certain regions and show minor inconsistencies in clustering. Even though these models exhibit less noise in the heatmap comparison (compared to WGAN), their reduced diversity becomes evident when analyzed with t-SNE. The improved coverage of WGAN is attributed to its use of the Wasserstein loss, which encourages alignment with the full data distribution and mitigates mode collapse despite noisier trajectory outputs. TimeGAN, in particular, exhibits more pronounced mode issues, identified through missing regions, uneven cluster densities, and the over-concentration of trajectories, likely due to its latent space training strategy that simplifies the learning task but increases the risk of overfitting to dominant patterns and reduces overall diversity.

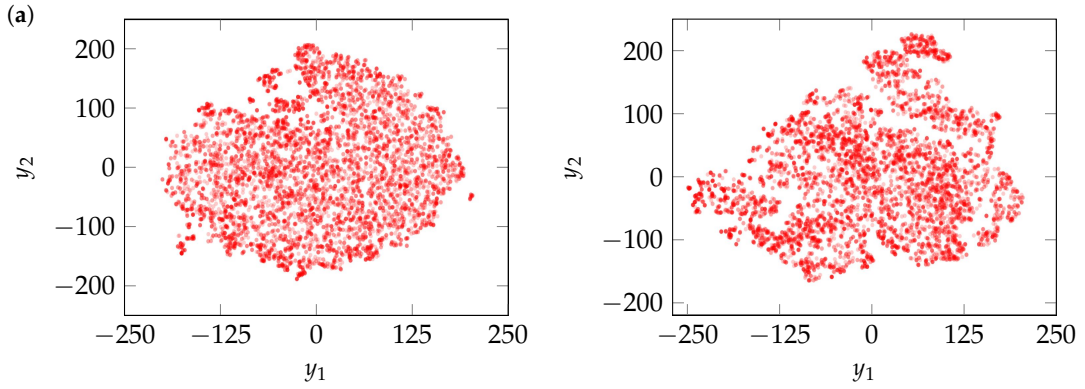


Figure 10. *Cont.*

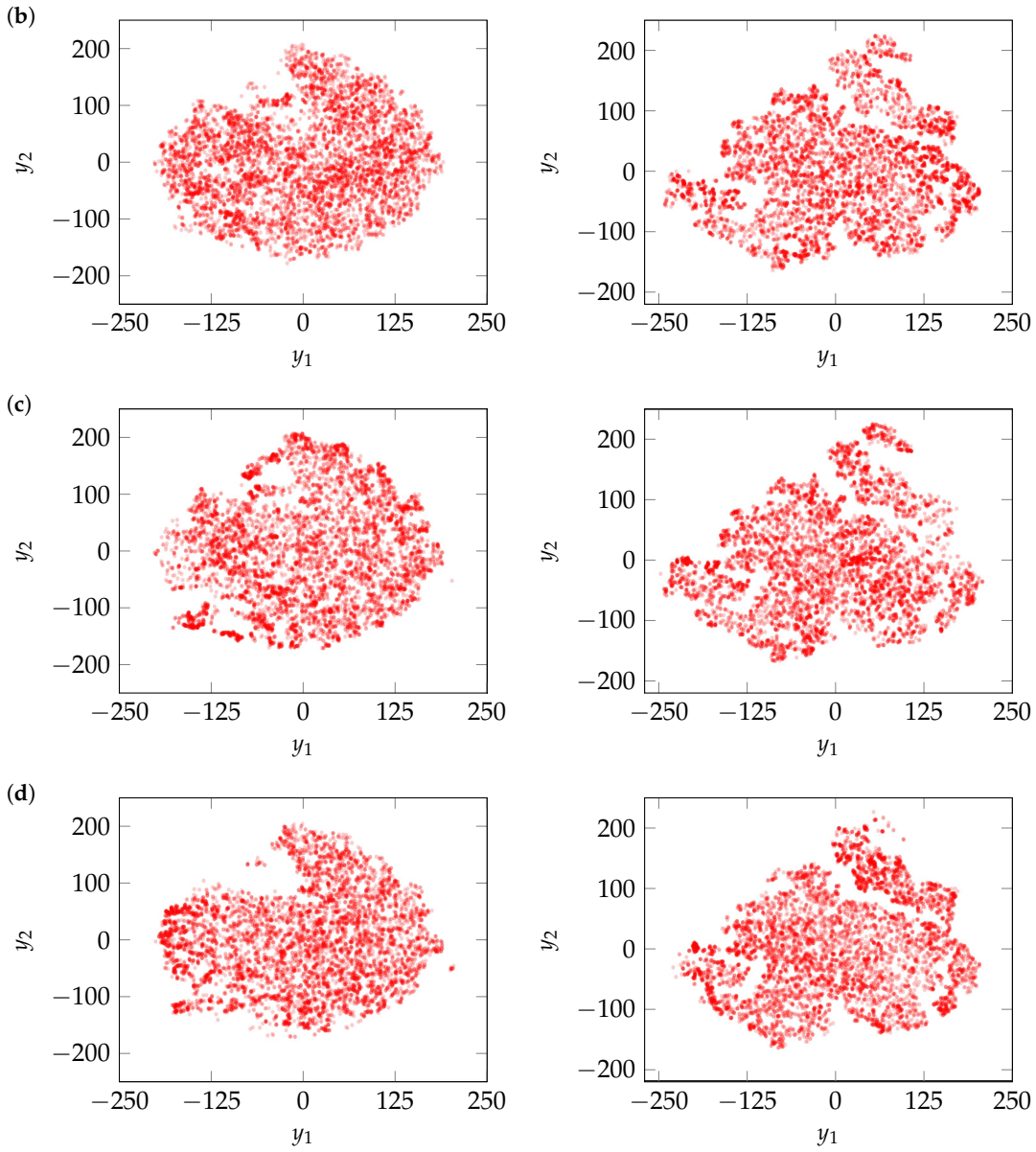


Figure 10. Two-dimensional t-SNE maps for lateral position (left) and longitudinal velocity (right) of cut-in maneuvers generated by (**a**) VAE, (**b**) GAN, (**c**) WGAN, and (**d**) TimeGAN.

7.3. Statistical Analysis Using KDE

While heatmaps and t-SNE plots reveal structural similarities between real and generated data, kernel density estimation [26,27] provides a statistical perspective on distribution alignment. The conformity of probability distributions is essential to obtain a correct risk assessment from the simulation of generated cut-in scenarios [10].

KDE is applied to the data $\mathbf{x}_i$ and $\tilde{\mathbf{x}}_i$ in the $\mathbf{x}$-space to compute smooth estimates of probability density functions by superposing Gaussian kernel densities. Given a set of observations $x_1, x_2, \ldots, x_n$, the estimated density at any value x of a random variable X is approximately

$$f_x(x) \approx \frac{1}{nh} \sum_{l=1}^{n} K\left(\frac{x - x_l}{h}\right), \tag{31}$$

where $K(x)$ is a kernel function (typically Gaussian, i.e., $K(x) = \exp(-x^2/2)/\sqrt{2\pi}$), h is the bandwidth parameter controlling smoothness, and n is the number of samples [26]. This non-parametric method ensures a smooth approximation of the data distribution without as-

suming any predefined shape. To facilitate better comparability, KDE is applied separately to $\mathbf{d}_i$ and $\mathbf{v}_i$, disregarding the potential correlation effects, where the measurements $\mathbf{x}_k^{(i)}$ of all time points t_k and the total set of cut-in maneuvers are summarized in a singe dataset, i.e., $x_l \in \{d_k^{(i)}, i = 1, \ldots, N_S,\ k = 1, \ldots, N_t\}$ or $x_l \in \{v_k^{(i)}, i = 1, \ldots, N_S,\ k = 1, \ldots, N_t\}$, resulting in $n = N_s N_t$.

Figure 11 presents KDE comparisons across models, where blue curves represent the probability density of measured data, and red curves correspond to generated data, respectively. All models closely match the distributions in the lateral position. However, discrepancies emerge in the longitudinal velocity, where VAE performs best, capturing the full distribution. WGAN follows, though it fails to represent all modes, leading to partial coverage only. In contrast, both TimeGAN and GAN struggle significantly, failing to recover key modes of the velocity distribution. It should be noted that variations in peak probabilities highlight associated cut-in maneuvers with a significantly higher or lower proportion of such cases in the generated test scenarios compared to real-world driving data. Consequently, this leads to an over- or underestimation of the failure probability of Advanced Driver Assistance Systems (ADASs) [10], depending on how critical these maneuvers are.

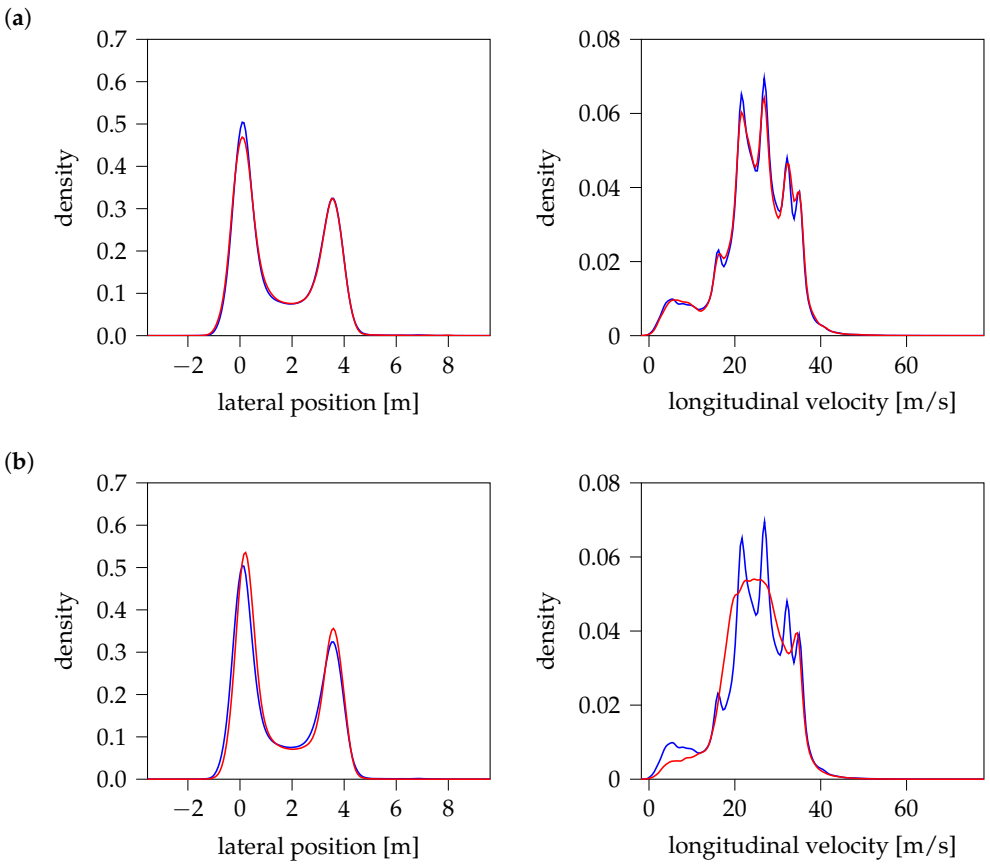


Figure 11. *Cont.*

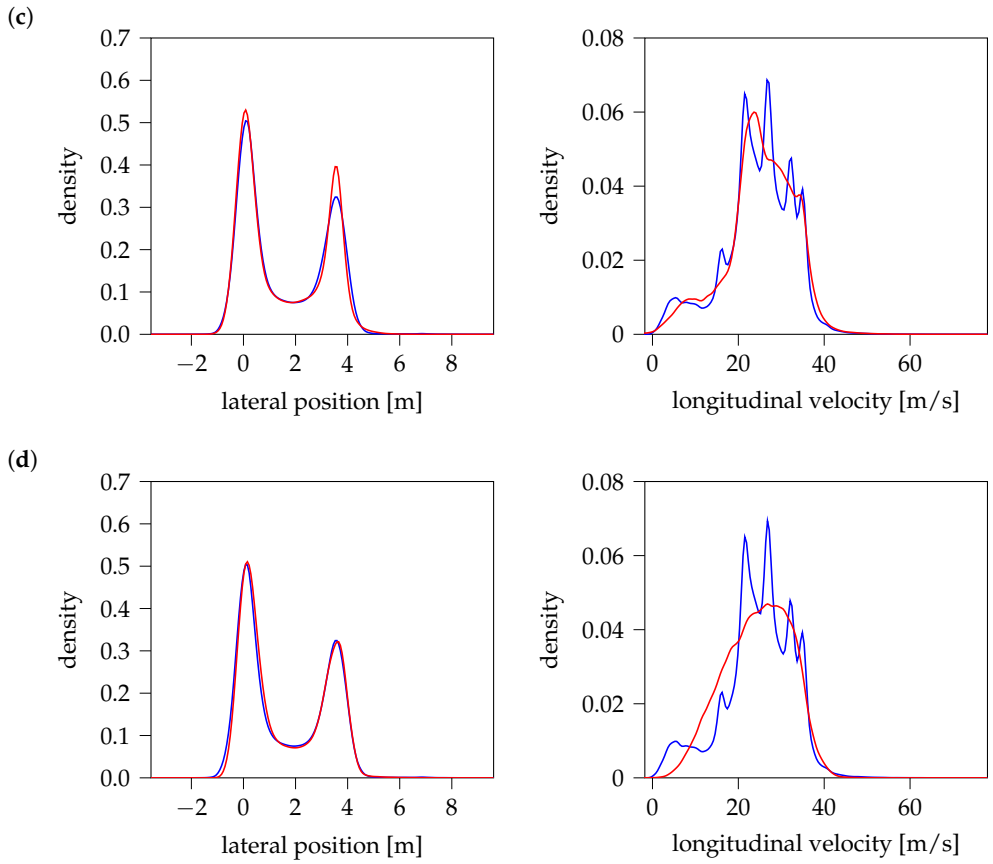


Figure 11. Kernel density estimation (KDE) for lateral position (left) and longitudinal velocity (right), comparing the probability density function of measured data (blue) with those generated (red) by (**a**) VAE, (**b**) GAN, (**c**) WGAN, and (**d**) TimeGAN.

8. Quantitative Metrics for the Comparison of Generated Results

Quantitative metrics are an essential complement for evaluating model performance, especially when visual methods such as t-SNE plots or KDE analyses suggest closely aligned results. While our qualitative evaluation highlights distinct variations across different aspects, such as noise and diversity, quantitative metrics help to unify these insights, providing a more comprehensive and objective assessment of realism and overall data representation. To assess the models quantitatively, two primary metrics are utilized: Mean of Incoming Variance of Outgoing (MiVo) [28] and the Hungarian distance [7].

Given the large dataset size of more than 8000 trajectories, the computational efficiency of these metrics is optimized by replacing Dynamic Time Warping (DTW) with Euclidean distance as the trajectory similarity measure. This adaptation is validated through mixing and discriminability tests [28] to ensure that the modified metrics remains reliable and robust. Additionally, a 1:1 ratio of real to generated samples is maintained across all evaluations to enable balanced comparisons. A sample efficiency test [28] reveals that the 1:1 ratio is optimal, and even when the proportion of generated samples is increased, the metrics remains consistent, indicating that higher ratios do not significantly impact reliability.

8.1. Mean of Incoming Variance of Outgoing (MiVo)

Similar to (26), MiVo [28] measures the differences in between generated and real trajectories, i.e.,

$$\Delta_{ji} = \|\tilde{\mathbf{x}}_j - \mathbf{x}_i\| = \sqrt{\sum_{k=1}^{N_t} \left(\tilde{x}_k^{(j)} - x_k^{(i)}\right)^2}, \quad i,j = 1,\ldots,N_s. \tag{32}$$

If for example $\Delta_{ji} = 0$, this means that there exists a generated counterpart $\tilde{\mathbf{x}}_j$ that is identical to a measured maneuver $\mathbf{x}_i$. By comparing each measured maneuver $\mathbf{x}_i$ (outgoing) with all generated maneuvers $\tilde{\mathbf{x}}_j$ (incoming), we obtain N_s^2 distances, summarized in a distance matrix

$$\boldsymbol{\Delta} = \begin{pmatrix} \Delta_{11} & \Delta_{12} & \cdots & \Delta_{1,N_s} \\ \Delta_{21} & \Delta_{22} & \cdots & \Delta_{2,N_s} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_{N_s,1} & \Delta_{N_s,2} & \cdots & \Delta_{N_s,N_s} \end{pmatrix}. \tag{33}$$

In order to find the most similar real maneuver $\mathbf{x}_i$ to a specific generated maneuver $\tilde{\mathbf{x}}_j$, we minimize the elements of the distance matrix along its j-th row, i.e.,

$$i = \arg\min_k \Delta_{jk}. \tag{34}$$

Performing this for all $\tilde{\mathbf{x}}_j$ results in an incoming distance vector of nearest neighbors

$$\boldsymbol{\delta} = \left[\min_k \Delta_{1,k}, \min_k \Delta_{2,k}, \ldots, \min_k \Delta_{N_s,k}\right]^\top. \tag{35}$$

The values δ_j in vector $\boldsymbol{\delta}$ represent the distances to the closest real counterpart $\mathbf{x}_i$ for each generated maneuver $\tilde{\mathbf{x}}_j$, respectively. Lower values suggest that generated maneuvers are highly realistic, meaning they closely resemble real-world maneuvers. Since realism should be evaluated based on the overall proximity of generated data to real data, the mean of $\boldsymbol{\delta}$ is taken. A low mean value indicates that, on average, generated samples are well aligned with real data.

Similarly, to find the most similar generated maneuver $\tilde{\mathbf{x}}_j$ to a specific real maneuver $\mathbf{x}_i$, the elements of the distance matrix may be minimized along its i-th column, i.e.,

$$j = \arg\min_k \Delta_{ki}. \tag{36}$$

Performing this for all $\mathbf{x}_i$ results in an outgoing distance vector of nearest neighbors

$$\widetilde{\boldsymbol{\delta}} = \left[\min_k \Delta_{k,1}, \min_k \Delta_{k,2}, \ldots, \min_k \Delta_{k,N_s}\right]^\top. \tag{37}$$

The values $\tilde{\delta}_i$ in vector $\widetilde{\boldsymbol{\delta}}$ indicate how well the generated data points $\tilde{\mathbf{x}}_j$ cover the real data $\mathbf{x}_i$. If all real maneuvers have close generated counterparts, this means that the generative model provides diverse and well-distributed outputs. However, if some real maneuvers have very close generated matches while others are not covered by close counterparts, this suggests an uneven distribution of generated samples. To capture this variation, the variance of $\widetilde{\boldsymbol{\delta}}$ is taken, which quantifies how consistently the real maneuvers are matched by generated ones. A low variance indicates that generated data points are

evenly distributed across the real dataset, avoiding cases where certain real maneuvers are well represented, while others are ignored. Both quantities may be summarized in a metric

$$\rho = \text{mean}(\delta) + \text{variance}(\widetilde{\delta}), \tag{38}$$

where a low value indicates a well-performing generative model producing synthetic maneuvers with high realism capturing the full variety of potential cut-in maneuvers. An example in the Appendix A demonstrates the computation of these quantities.

The application of MiVo to the synthetic maneuvers in Figure 8 yields the values presented in the second column of Table 6. As expected, VAE performs the best, with a noticeable gap to the other GAN-based models, which show relatively similar values. Among them, TimeGAN has the highest MiVo value, likely due to its limited diversity as already seen in the t-SNE and KDE plots, where it misses certain regions. While MiVo effectively identifies major differences, it is less sensitive to minor differences, such as those between GAN and WGAN, where global structure improvements are offset by increased local noise. This limitation highlights the need for a more precise evaluation, which is why we also apply the Hungarian distance [7], offering an additional measure of alignment by finding the optimal one-to-one matching between generated and real samples.

8.2. Hungarian Distance

The Hungarian distance [7] utilizes combinatorial optimization to find an optimal match between the generated sample and the real sample such that $\tilde{\mathbf{x}}_j$ and $\mathbf{x}_i$ are paired one to one. Formally, the Hungarian distance is given as the minimal total pairwise distance

$$D_H = \min_{\pi_k} \sum_{j=1}^{N_s} \Delta_{j,\pi_k(j)}, \tag{39}$$

where $i = \pi_k(j)$ is the jth element of the permutation π_k, $k = 1, \ldots, N_s!$, resulting in distance Δ_{ji}. This metric provides a precise measure of how well the generated data points match the real data points, although it can be influenced by outliers in the distribution. Directly computing all $N_s!$ permutations π_k is infeasible. Instead, the Hungarian Algorithm [29] finds the optimal assignment by solving a minimum-cost bipartite matching problem [30]. Given a cost matrix $\mathbf{\Delta} \in \mathbb{R}^{N_s \times N_s}$ according to Equation (33), the optimal assignment is obtained by

$$\pi^* = \arg\min_{\pi_k} \sum_{j=1}^{N_s} \Delta_{j,\pi_k(j)}. \tag{40}$$

The algorithm iteratively modifies the cost matrix by subtracting row/column minima, covering zeros with the fewest lines, and adjusting uncovered elements until an optimal assignment is found. This avoids brute-force computation while ensuring the minimum transport cost.

The Hungarian distance establishes a one-to-one correspondence between real and generated examples while minimizing the total pairwise assignment cost. If multiple generated examples have identical distances to a single real example, ambiguity arises in the assignment process. The Hungarian algorithm resolves this by considering the global transport cost across all possible permutations. Appendix A includes an example that demonstrates the calculation of the Hungarian distance (39).

According to the third column in Table 6, VAE once again achieves the best performance with Hungarian distance values at least 20% lower than those of the other models. The ordering of the D_H values aligns with the qualitative observations, where WGAN performs slightly better than GAN, while both significantly outperform TimeGAN in terms

of diversity. In the qualitative evaluation, the GAN-based models exhibit different strengths and weaknesses, making direct comparisons challenging. However, the quantitative metrics helps to identify clearer distinctions in the overall performance of generative models.

Table 6. Comparison of generative AI models with the VAE model.

Model	MiVo	Hungarian Distance	Difference to VAE	
			MiVo	Hungarian
VAE	**0.155297**	**0.246246**	–	–
GAN	0.252633	0.325716	+62.68%	+32.27%
WGAN	0.253929	0.298249	+63.51%	+21.12%
TimeGAN	0.265483	0.344059	+70.95%	+39.72%

9. Conclusions

Risk assessment based on the simulation of safety-critical scenarios requires an algorithm capable of generating synthetic maneuvers with high realism, sufficient diversity, and accurate statistical properties. The present study evaluates and compares four potential AI models: VAE, GAN, WGAN, and TimeGAN for their ability to generate realistic and diverse critical scenarios. The results show that VAE performs best, outperforming GAN-based models in producing smooth and realistic trajectories while effectively capturing the full variety of real-world cut-in scenarios. Consequently, VAE proves to be a robust tool for supporting the development and validation of automated driving systems. By producing an arbitrary number of safety-critical scenarios with correct statistical properties, it supports simulation-based risk assessment of these systems, facilitating safer integration into real-world traffic environments.

Building on these findings, future research should aim at extending the capabilities of VAEs by enabling the generation of multiple trajectory variations for a given scenario, increasing its utility in complex, multi-agent environments. Additionally, incorporating semi-supervised learning techniques could prioritize the generation of rare high-risk scenarios by leveraging partially labeled data. Future work should also focus on expanding the generative models to simulate more complex driving maneuvers, such as sudden braking or pedestrian crossing, which present additional challenges for realistic scenario generation. These advancements would enable more precise, flexible, and scalable scenario generation, supporting the continued progress of automated driving technologies and ensuring their safety and reliability in increasingly dynamic environments.

Author Contributions: Conceptualization, Z.K. and M.M.M.; methodology, Z.K.; software, Z.K. and M.M.M.; investigation, Z.K. and M.M.M.; writing—original draft, Z.K., M.M.M. and D.B.; supervision, Z.K. and D.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets presented in this article are not available due to the privacy restrictions of the OEM. Requests to access the datasets should be directed to the corresponding author.

Conflicts of Interest: Author Manasa Mariam Mammen and Zafer Kayatas were employed by the company Mercedes-Benz AG. The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

AD	Autonomous Driving
ADAS	Advanced Driver Assistance System
AI	Artificial Intelligence
ECU	Electronic Control Unit
GAN	Generative Adversarial Network
HAD	Highly Automated Driving
KDE	Kernel Density Estimation
KL	Kullback-Leibler divergence
MiVo	Mean of Incoming Variance of Outgoing
RNN	Recurrent Neural Network
t-SNE	t-Distributed Stochastic Neighbor Embedding
TimeGAN	Time-series Generative Adversarial Network
TSF	Time-Series Forest
VAE	Variational Autoencoder
WGAN	Wasserstein Generative Adversarial Network
WGAN-GP	Wasserstein Generative Adversarial Network with Gradient Penalty

Appendix A. Demonstrating Example for MiVo and Hungarian Distance

To illustrate the metric computation in Section 8, we consider a case of three generated samples $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_3$ for three real samples $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. The distance matrix may be given as

$$\boldsymbol{\Delta} = \begin{bmatrix} 8 & 4 & 7 \\ 5 & 2 & 3 \\ 3 & 4 & 8 \end{bmatrix}. \tag{A1}$$

MiVo calculations using Equations (35) and (37) result in the incoming vector

$$\delta = \begin{bmatrix} \min(8,4,7), & \min(5,2,3), & \min(3,4,8) \end{bmatrix}^\top = \begin{bmatrix} 4 & 2 & 3 \end{bmatrix}^\top \tag{A2}$$

associated with the pairings $\tilde{\mathbf{x}}_1 \to \mathbf{x}_2, \tilde{\mathbf{x}}_2 \to \mathbf{x}_2$ and $\tilde{\mathbf{x}}_3 \to \mathbf{x}_1$, and the outgoing distance vector

$$\widetilde{\delta} = \begin{bmatrix} \min(8,5,3), & \min(4,2,4), & \min(7,3,8) \end{bmatrix}^\top = \begin{bmatrix} 3 & 2 & 3 \end{bmatrix}^\top \tag{A3}$$

associated with the pairings $\mathbf{x}_1 \to \tilde{\mathbf{x}}_3, \mathbf{x}_2 \to \tilde{\mathbf{x}}_2$ and $\mathbf{x}_3 \to \tilde{\mathbf{x}}_2$. Using these vectors, the MiVo metric (38) yields

$$\text{mean}(\delta) = 3, \quad \text{variance}(\widetilde{\delta}) = \frac{1}{3} \quad \Rightarrow \quad \rho = 3 + \frac{1}{3} = \frac{10}{3}.$$

Obviously, MiVo allows a single real data point $\mathbf{x}_i$ to be mapped to multiple generated data points $\tilde{\mathbf{x}}_j$ as the nearest neighbor (here $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2 \to \mathbf{x}_2$) and vice versa (here $\mathbf{x}_2, \mathbf{x}_3 \to \tilde{\mathbf{x}}_2$). In contrast, the Hungarian distance enforces a one-to-one mapping by generating all $3! = 6$ permutations π_k and computing the associated cost of corresponding distances:

k	π_k	Assignment	Cost
1	$(1,2,3)$	$\tilde{\mathbf{x}}_1 \leftrightarrow \mathbf{x}_1, \tilde{\mathbf{x}}_2 \leftrightarrow \mathbf{x}_2, \tilde{\mathbf{x}}_3 \leftrightarrow \mathbf{x}_3$	$8+2+8=18$
2	$(1,3,2)$	$\tilde{\mathbf{x}}_1 \leftrightarrow \mathbf{x}_1, \tilde{\mathbf{x}}_2 \leftrightarrow \mathbf{x}_3, \tilde{\mathbf{x}}_3 \leftrightarrow \mathbf{x}_2$	$8+3+4=15$
3	$(2,1,3)$	$\tilde{\mathbf{x}}_1 \leftrightarrow \mathbf{x}_2, \tilde{\mathbf{x}}_2 \leftrightarrow \mathbf{x}_1, \tilde{\mathbf{x}}_3 \leftrightarrow \mathbf{x}_3$	$4+5+8=17$
4	$(2,3,1)$	$\tilde{\mathbf{x}}_1 \leftrightarrow \mathbf{x}_2, \tilde{\mathbf{x}}_2 \leftrightarrow \mathbf{x}_3, \tilde{\mathbf{x}}_3 \leftrightarrow \mathbf{x}_1$	$4+3+3=10$
5	$(3,1,2)$	$\tilde{\mathbf{x}}_1 \leftrightarrow \mathbf{x}_3, \tilde{\mathbf{x}}_2 \leftrightarrow \mathbf{x}_1, \tilde{\mathbf{x}}_3 \leftrightarrow \mathbf{x}_2$	$7+5+4=16$
6	$(3,2,1)$	$\tilde{\mathbf{x}}_1 \leftrightarrow \mathbf{x}_3, \tilde{\mathbf{x}}_2 \leftrightarrow \mathbf{x}_2, \tilde{\mathbf{x}}_3 \leftrightarrow \mathbf{x}_1$	$7+2+3=12$

Clearly, the optimal assignment is given by permutation π_4, resulting in the minimum cost and thus Hungarian distance

$$D_H = 10. \tag{A4}$$

References

1. Zhang, X.; Tao, J.; Tan, K.; Törngren, M.; Sánchez, J.M.G.; Ramli, M.R.; Tao, X.; Gyllenhammar, M.; Wotawa, F.; Mohan, N.; et al. Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study. *IEEE Trans. Softw. Eng.* **2022**, *49*, 991–1026. [CrossRef]
2. Society of Automotive Engineers, SAE J3016. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. *SAE Int.* **2018**, *2018*, 1–35.
3. Zhong, Z.; Tang, Y.; Zhou, Y.; Neves, V.d.O.; Liu, Y.; Ray, B. A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation. *arXiv* **2021**, arXiv:2112.00964.
4. Sauerbier, J.; Bock, J.; Weber, H.; Eckstein, L. Definition of Scenarios for Safety Validation of Automated Driving Functions. *ATZ Worldw.* **2019**, *121*, 42–45. [CrossRef]
5. Riedmaier, S.; Ponn, T.; Ludwig, D.; Schick, B.; Diermeyer, F. Survey on Scenario-Based Safety Assessment of Automated Vehicles. *IEEE Access* **2020**, *8*, 87456–87477. [CrossRef]
6. Kayatas, Z. Optimierung der Zuverlässigkeitsanalyse für die Simulative Absicherung Hochautomatisierter Fahrerassistenzsysteme. Master's Thesis, Leibniz Universität Hannover, Hannover, Germany, 2020.
7. Demetriou, A.; Alfsvåg, H.; Rahrovani, S.; Haghir Chehreghani, M. A Deep Learning Framework for Generation and Analysis of Driving Scenario Trajectories. *Comput. Sci.* **2023**, *4*, 251. [CrossRef]
8. Krajewski, R.; Moers, T.; Meister, A.; Eckstein, L. Béziervae: Improved Trajectory Modeling Using Variational Autoencoders for the Safety Validation of Highly Automated Vehicles. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 3788–3795.
9. Ding, W.; Wang, W.; Zhao, D. A New Multi-Vehicle Trajectory Generator to Simulate Vehicle-to-Vehicle Encounters. *arXiv* **2018**, arXiv:1809.05680.
10. Kayatas, Z.; Bestle, D.; Bestle, P.; Reick, R. Generation of Realistic Cut-In Maneuvers to Support Safety Assessment of Advanced Driver Assistance Systems. *Appl. Mech.* **2023**, *4*, 1066–1077. [CrossRef]
11. Donahue, J.; Simonyan, K. Large Scale Adversarial Representation Learning; In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 8 December 2019; Volume 32.
12. Karras, T. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv* **2019**, arXiv:1812.04948.
13. Karras, T.; Laine, S.; Aittala, M.; Hellsten, J.; Lehtinen, J.; Aila, T. Analyzing and Improving the Image Quality of StyleGAN. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 8110–8119.
14. Yoon, J.; Jarrett, D.; Van der Schaar, M. Time-Series Generative Adversarial Networks. Advances in Neural Information Processing Systems. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 8 December 2019; Volume 32.
15. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In *Communications of the ACM;* Advances in Neural Information Processing Systems; Association for Computing Machinery: New York, NY, USA, 2014; Volume 27.
16. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein Generative Adversarial Networks. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; PMLR; pp. 214–223.
17. Kayatas, Z.; Bestle, D. Scenario Identification and Classification to Support the Assessment of Advanced Driver Assistance Systems. *Appl. Mech.* **2024**, *5*, 563–578. [CrossRef]
18. Kingma, D.P.; Welling, M. An Introduction to Variational Autoencoders. In *Foundations and Trends® in Machine Learning*; Mark de Jongh: Dordrecht, The Netherlands, 2019; Volume 12, pp. 307–392.
19. Kingma, D.P.; Salimans, T.; Jozefowicz, R.; Chen, X.; Sutskever, I.; Welling, M. Improved Variational Inference with Inverse Autoregressive Flow. In Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016; Advances in Neural Information Processing Systems; Volume 29.
20. Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.; Glorot, X.; Botvinick, M.; Mohamed, S.; Lerchner, A. Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
21. Wang, Y. A Mathematical Introduction to Generative Adversarial Nets (GAN). *arXiv* **2020**, arXiv:2009.00169.
22. Villani, C. *Optimal Transport: Old and New*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 338.

23. Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A.C. Improved Training of Wasserstein GANs. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Advances in Neural Information Processing Systems; Volume 30.
24. Van der Maaten, L.; Hinton, G. Visualizing Data Using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
25. Hinton, G.E.; Roweis, S.T. Stochastic neighbor embedding. *Adv. Neural Inf. Process. Syst.* **2002**, *15*, 857–864.
26. Parzen, E. On Estimation of a Probability Density Function and Mode. *Ann. Math. Stat.* **1962**, *33*, 1065–1076. [CrossRef]
27. Chen, Y.C. A Tutorial on Kernel Density Estimation and Recent Advances. *Biostat. Epidemiol.* **2017**, *1*, 161–187. [CrossRef]
28. Arnout, H.; Bronner, J.; Runkler, T. Evaluation of Generative Adversarial Networks for Time Series Data. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Virtual, 18–22 July 2021; pp. 1–7.
29. Kuhn, H.W. The Hungarian Method for the Assignment Problem. *Nav. Res. Logist. Q.* **1955**, *2*, 83–97. [CrossRef]
30. Crouse, D.F. On Implementing 2D Rectangular Assignment Algorithms. *IEEE Trans. Aerosp. Electron. Syst.* **2016**, *52*, 1679–1696. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.