Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

AN EMPIRICAL STUDY OF GENERATIVE MODELS AS INPUT MODELS FOR SIMULATION

Zhou Miao[1], Zhiyuan Huang[2], Zhaolin Hu[2]

[1]Dept. of Aeronautical and Aviation Eng., The Hong Kong Polytechnic University, Hong Kong, CHINA
[2]Dept. of Management Science and Eng., Tongji University, Shanghai, CHINA

ABSTRACT

Input modeling is pivotal for generating realistic data that mirrors real-world variables for simulation, yet traditional parametric methods often fail to capture complex dependencies. This study investigates the efficacy of modern generative models—such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Normalizing Flows, and Diffusion Models—in addressing these challenges. Through systematic experiments on synthetic and real-world datasets, we evaluate their performance using metrics like Wasserstein distance and quantile loss. Our findings reveal that VAEs and Denoising Diffusion Probabilistic Models (DDPMs) consistently outperform other models, particularly in capturing nonlinear relationships, while GAN-based approaches exhibit instability. These results provide practical insights for practitioners, highlighting models that deliver reliable performance without extensive customization, and outline promising directions for future research in simulation input modeling.

1 INTRODUCTION

Input modeling is necessary in simulation modeling when random variables influence the system being modeled. The goal is to create a model that generates input data mimicking the real-world data of these variables. This step is critical because well-represented variables and captured behaviors enable the simulation to produce meaningful results, directly impacting downstream tasks such as decision-making, risk management, and system optimization. This topic has been extensively studied, and detailed discussions can be found in various textbooks (e.g. Chapter 5 in Henderson and Nelson (2006), Chapter 6 in Nelson (2013), Chapter 6 in Law (2014)) and tutorials (e.g. Kuhl et al. (2006); Biller and Gunes (2010)).

Despite its importance, input modeling poses significant challenges, particularly when dealing with multivariate data. Classical approaches often rely on parametric distributions with explicit density functions, which are chosen based on assumptions about the data's underlying structure. While these methods are mathematically tractable and computationally efficient, they struggle to capture the complexity of real-world systems, where variables often exhibit intricate dependencies, such as correlations, nonlinear relationships.

The rise of deep learning and neural networks has revolutionized many fields by significantly improving performance, often at the cost of sacrificing explicit mathematical formulations. Unlike classical methods that rely on predefined parametric models with clear interpretability, neural networks learn complex, data-driven representations through layers of nonlinear transformations. This shift from explicit, parametric modeling to implicit, data-driven approaches has enabled the capture of intricate patterns and dependencies in high-dimensional data. In the context of probabilistic modeling, this shift is reflected in the development of modern generative models. These models, including Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), leverage neural networks to learn and generate data from complex, high-dimensional distributions, without relying on restrictive assumptions or explicit density functions.

In simulation input modeling, generative models have been increasingly adopted to address diverse challenges, demonstrating superior performance over traditional approaches. For instance, GANs have shown promise in synthesizing input distributions (Montevechi et al. 2021) and augmenting datasets for Urban Air Mobility evacuation scenarios (Alsaheal et al. 2024). Similarly, VAEs have been applied to

simulate vehicle fault scenarios (Kuiper et al. 2024) and to generate time series data (Cen et al. 2020). However, with the growing variety of generative frameworks, each with distinct strengths and limitations, practitioners face ambiguity in selecting the most suitable model for their specific tasks.

In this paper, we empirically evaluate the performance of unmodified, off-the-shelf generative models in generating diverse simulation input data. By benchmarking these models without architectural adjustments or domain-specific tuning, our study highlights relative strengths and weaknesses across frameworks. For practitioners, these results provide evidence-based comparisons to inform model selection, while researchers can identify architectures that serve as solid starting points for further refinement.

The remainder of the paper is structured as follows: Section 2 introduces generative models and their architectures. Section 3 details the experimental setup, datasets, and evaluation metrics. Section 4 presents empirical results, and Section 5 discusses implications and future work.

2 OVERVIEW OF GENERATIVE MODELS

Modern generative models leverage neural networks to effectively learn intricate data distributions, overcoming the limitations of classical parametric models, which struggle to capture complex dependencies. Numerous approaches have been proposed to integrate neural networks with probabilistic frameworks, leading to a variety of generative model classes. These models are primarily categorized based on how they combine standard probability distributions with neural networks to produce realistic data. Additionally, subtle differences may emerge depending on the specific training criterion, which, together with variations in model structure, determines how the parameters of the neural networks are optimized during training.

Before delving into generative models, we introduce the following notation. Let $X \in \mathscr{X} \subseteq \mathbb{R}^d$ and $Y \in \mathscr{Y} \subseteq \mathbb{R}^m$ denote jointly distributed random vectors representing uncertain variables in simulation. We distinguish between X and Y to reflect practical scenarios in which X is observed before the simulation, thereby making the simulation's randomness dependent primarily on Y. Our goal is to construct an input model for Y given an observed X. For simplicity, we refer to X as the conditioning variable and Y as the conditional variable. The training dataset for the models is denoted as $(X_i, Y_i)_{i=1}^n$. Additionally, since all generative models incorporate randomness from basic distributions (e.g., standard Gaussian), we use $Z \in \mathscr{Z} \subseteq \mathbb{R}^l$ to represent this random source. Neural network parameters are denoted by θ, and the generated data given a feature $X = x$ from a specific generative model is represented by $\tilde{Y}_\theta(x)$.

Table 1 summarizes all generative models considered in this paper. Since the structure of data generation naturally provides a useful framework for classifying these models, we adopt this viewpoint to organize our subsequent introduction.

The first type of generative models directly maps the random source Z to the data space using neural networks. Let $f_\theta(\cdot, x) : \mathscr{Z} \to \mathscr{Y}$ denote the neural network parameterized by θ. We generate data as $\tilde{Y}_\theta(x) = f_\theta(Z, x)$. Given this structure, the goal of training is to ensure the distribution of $\tilde{Y}_\theta(x)$ closely approximates the conditional distribution $Y|X = x$ for all x.

For directly mapped models, we first consider normalizing flow models (Dinh et al. 2016; Winkler et al. 2019). An arbitrary choice of f_θ would make it difficult to explicitly calculate the density of $\tilde{Y}_\theta(x)$, thus preventing training via any criterion that requires the density function, such as maximum likelihood estimation (MLE). Normalizing flow models overcome this issue by restricting f_θ to be invertible and differentiable with an efficiently computable Jacobian determinant. This structural constraint allows direct computation of the likelihood through the Jacobian transformation, enabling training via MLE or equivalently minimizing the KL divergence relative to the conditional distribution of $Y|X = x$.

However, imposing structural constraints on the neural network f_θ can restrict the model's flexibility. To address the challenge of explicitly evaluating likelihoods when using an arbitrary neural network f_θ, an auxiliary neural network can be introduced to provide an approximation. Generative Adversarial Networks (GANs) represent a prominent implementation of this strategy (Goodfellow et al. 2014; Mirza and Osindero 2014). GANs employ an additional neural network, called the discriminator, which differentiates between real and generated data. The discriminator implicitly learns features of the underlying data

distribution, effectively providing an approximation of the Jensen-Shannon (JS) divergence between the real and generated data distributions. This approximation facilitates assessing the quality of generated data, enabling parameter updates during training. Wasserstein GANs (Arjovsky et al. 2017; Gulrajani et al. 2017; Engelmann and Lessmann 2021) extend this approach by training the discriminator to estimate and minimize the Wasserstein distance between real and generated data distributions, enhancing training stability and the quality of generated samples.

The second type of generative model leverages neural networks to map simple random sources into the parameter space of parametric distributions or stochastic processes, enhancing model flexibility by allowing parameters themselves to be randomly generated. Specifically, let $p(x|\gamma)$ denote the density function of a parametric distribution with parameter $\gamma \in \Gamma$. In this setting, a neural network maps the random vector Z into the parameter space Γ, providing parameters $\gamma_\theta(Z)$. Subsequently, generated data are obtained by sampling from the distribution with these parameters: $\tilde{X}_\theta \sim p(x|\gamma_\theta(Z))$.

VAEs (Kingma and Welling 2013; Sohn et al. 2015) use this structure to generate samples. This data generation structure is referred to as the decoder, and here Z is considered as the latent variable of the data. In this framework, training is conducted via likelihood maximization; however, the latent variable Z is unobserved and makes the likelihood challenging to compute directly. This issue is addressed through variational inference from Bayesian literature, which constructs a tractable approximation of the posterior distribution $p(z|x)$. Specifically, this approximation utilizes another neural network, referred to as the encoder, which approximates $q_\phi(z|x) \approx p(z|x)$. This approximation establishes an Evidence Lower Bound (ELBO) of the likelihood function, provides an effective proxy for the true log-likelihood.

Although it may not be strictly categorized here, we also include diffusion models (Ho et al. 2020; Song et al. 2020a) under this type. For diffusion models, $p(x|\gamma_\theta(Z))$ denotes the final-state distribution of a diffusion process that evolves from an initial state Z through a fixed sequence of steps discretized in time. Specifically, let X_t denote the state of the diffusion process at time step t, the sampling process uses a backward notation form $t = T$ to $t = 0$. Starting from a random sample $X_T = Z$. This diffusion process is governed by the transition distribution $p(x_{t-1}|\gamma_\theta(X_t,t))$, where the drift (mean direction) of the kernel is determined by a neural network, $\gamma_\theta(X_t,t)$, mapping from the previous state and the time step t. Then we have $\tilde{X}_\theta \sim p(x_0|\gamma_\theta(X_1,1))$ in the last step. As this discretized diffusion process can be viewed as a hierarchical latent model, the training can follow variational inference like the VAE. Alternatively, one could adopt a continuous diffusion process perspective and use score function matching for training (Song et al. 2020b). However, due to practical simplicity in training, the variational approach is more generally used.

Table 1: Data generation structures and training objective function for different types of generative models.

Model Name	Data Generation Structure	Target Criterion
Normalizing Flow	Draw $Z \sim p$, then $\tilde{X}_\theta = f_\theta(Z)$	KL-Exact
GAN	Draw $Z \sim p$, then $\tilde{X}_\theta = f_\theta(Z)$	JS-Approximation
WGAN	Draw $Z \sim p$, then $\tilde{X}_\theta = f_\theta(Z)$	Wasserstein-Approximation
VAE	Draw $Z \sim p$, then $\tilde{X}_\theta \sim p(x\|\gamma_\theta(Z))$	KL-ELBO
Diffusion Models	Draw $Z \sim p$, then $\tilde{X}_\theta \sim p(x\|\gamma_\theta(Z))$	KL-ELBO/Score Function Matching

3 EXPERIMENT SETUP

In this section, we detail the components of our numerical experiments.Note that these experiments assume sufficient data with a stable distribution. We describe the dataset, evaluation metrics, and benchmark methods used for comparative analysis.

3.1 Datasets

Our numerical experiments employ three distinct datasets to compare the performance of generative models. Each dataset includes a multidimensional conditioning variable X and a single-dimensional conditional variable Y. Using a single-dimensional Y simplifies metric estimation, enabling clearer insights into the comparative performance of different models. The first two datasets are synthetically generated using linear and nonlinear models, respectively, while the third dataset is drawn from real-world data. Details are as follows:

Dataset 1 (Linear Model) In the linear model, we generate the conditioning variable $X \sim N(0, I_d)$, where I_d is the $d \times d$ identity matrix. The conditional variable Y is then generated as $Y = \beta^\top X + \varepsilon$, where $\beta \in \mathbb{R}^d$ is the coefficient vector with each entry independently drawn as $\beta_j \sim U[-1,1]$, and $\varepsilon \sim N(0,1)$ represents independent Gaussian noise.

Dataset 2 (Nonlinear Model) In the nonlinear model, we generate mutually uncorrelated $X \sim N(0,1)$ and $y = \sum_{j=1}^{d} g_j(x_j) + \varepsilon$ with $\varepsilon \sim N(0,1)$, where each g_j is a nonlinear function of x_j and x_j denotes the jth element of $x \in \mathbb{R}^d$. Specifically, each g_j is selected from $g(x) = e^x$, $g(x) = \sin(x)$, and $g(x) = x^2$ randomly.

In our experiments, we evaluate the performance of different models on the generated datasets and analyze how performance changes with data richness. Rather than varying the dataset size, we adjust the dimensionality of the conditioning variable X to represent richer (lower-dimensional) or sparser (higher-dimensional) data conditions. Specifically, we fix the dataset sizes as follows: 11,583 for training data; 1,287 for validation data; 5,791 for test data (used for metric computation).

Dataset 3 (Real-world Data) In addition to the synthetic datasets, we also consider a real-world dataset containing Walmart's historical weekly sales data. Besides weekly sales, this dataset includes features such as Store, Date, Holiday Flag, Temperature, Fuel Price, CPI, and Unemployment, covering the period from 2010-02-05 to 2012-11-01. In our analysis, we treat the weekly sales as the conditional variable and the remaining attributes as conditioning variables. The dataset is publicly available at Kaggle (https://www.kaggle.com/datasets/yasserh/walmart-dataset).

This real-world dataset contains 6435 data points. For evaluation, we employ leave-one-out cross-validation to compute performance metrics. In each iteration, the data is partitioned into training, validation, and test sets at an 8:1:1 ratio.

3.2 Generative Model Architectures and Training

In the experiment, we select several generative models for comparative studies. These models include the Real NVP model by Dinh et al. (2016), representing the normalizing flow models; GAN by Goodfellow et al. (2014); WGAN (Arjovsky et al. 2017), along with its variants WGAN-GP, which features an alternative loss function (Gulrajani et al. 2017). Additionally, the Variational Autoencoder (VAE) by Kingma and Welling (2013), and finally the Denoising Diffusion Probabilistic Model (DDPM) by Ho et al. (2020) serve as examples of diffusion models. The structures and training details of these models are introduced in this section.

3.2.1 Model Structures

In generative models, the architecture of the neural network—including its design, connectivity, and chosen activation functions—is critical to the model's expressiveness and its capacity to learn effectively. Therefore, to ensure a fair comparison across different generative models, we must carefully control the structure and size of the neural networks to maintain similar levels in each experiment.

For the network structures, the models—including $f_\theta(Z)$ in GANs, $\gamma_\theta(Z)$ in VAE, and $\gamma_\theta(X_t, t)$ in DDPM—employ a Multilayer Perceptron (MLP), which is a fully connected neural network, with ReLU activation functions. However, there is an exception: the Real NVP model, which uses a specialized network structure to ensure explicit likelihood calculation.

For the size of the neural networks, we control the total number of parameters to ensure they are of similar magnitude across experiments, despite varying dimensions of the conditioning variable. Table 2 presents the specific numbers of parameters used in each experiment. Column d represents the whole dimension of conditioning variables and conditional variable.

Table 2: Number of parameters for different generative models used in the experiments.

d	VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM
5	5451	6978	6978	6978	2420	7265
10	5771	8011	8011	8011	3920	7585
15	6091	8816	8816	8816	5420	7905
20	6411	9621	9621	9621	6920	8225
25	6731	10426	10426	10426	8420	8545

3.2.2 Training of the Models

Training is also a critical factor in model performance. In the experiments, we use the RMSProp optimizer (Zaheer and Shaziya 2019) for GAN-type models, and ADAM optimizer (Kingma and Ba 2014) for training remaining models. To ensure a fair comparison, we standardize the training process across all models by using consistent hyperparameters within their respective optimization algorithms.

The key hyperparameters in these optimizers include the learning rate, which adjusts the step size during optimization; the batch size, determining the number of samples processed before the model is updated; the number of training epochs, defining the total number of times the model is updated through the entire training; and an early stopping mechanism, designed to prevent overfitting by ceasing training if the loss estimated using the validation dataset does not decrease after a specified number of iterations.

In our experiments, we maintain consistent training conditions with a fixed learning rate of 0.001 and a batch size of 64. All models undergo training for a maximum of 500 epochs, with early stopping implemented to halt training if the validation loss fails to improve for 100 consecutive epochs. This configuration ensures a fair comparison across models while preventing overfitting through the early stopping mechanism.

3.3 Performance Metrics

For input models, the likelihood of a test dataset is the most prevalent criterion for performance evaluation. However, this likelihood is not available for models without an explicit density function, making it challenging to apply to generative models (except for normalizing flow models). Therefore, we consider two alternative metrics to compare the model performance.

We note that the two metrics are designed to accommodate the challenges presented by the real-world dataset experiment, ensuring consistent metrics across all three experiments. In particular, the real-world dataset consists of (X_i, Y_i) pairs, where most X_i values are unique. This uniqueness makes it challenging to estimate the conditional distributions $p(y|x)$ of the real-world data, complicating direct comparisons with the generative models.

3.3.1 Approximated Wasserstein Distance on Marginal Distribution

The first metric in our experiments is the approximate Wasserstein distance, used to assess the marginal distribution of the conditional variable Y. In particular, for each test data point (X_i, Y_i), we generate a corresponding sample $\tilde{Y}_i$ using the generative model conditioned on X_i. The first-order Wasserstein distance is then computed between the empirical distributions of the true $\{Y_i\}$ and the generated $\{\tilde{Y}_i\}$, providing a statistical measure of how well the model captures the marginal behavior of Y.

We emphasize that strong performance on the marginal distribution does not imply a high-quality model; this metric can only reliably identify poor models. While a small Wasserstein distance indicates

that the generative model captures the marginal behavior of Y, it provides no assurance that the model has learned the conditional relationship between X and Y. Thus, this metric serves as a necessary but insufficient sanity check—additional evaluations are required to assess the conditional models.

3.3.2 Average Quantile Loss

To evaluate the conditional models $p(y|x)$, we employ a quantile calibration metric based on the quantile check function (Koenker and Bassett Jr 1978). For each test data point (X_i, Y_i) and the corresponding conditional model $p(y|X_i)$, we estimate the τ-quantile, denoted by $q_\tau(Y|X_i)$, of the conditional distribution. We then compute the quantile residual $u_i = Y_i - q_\tau(Y|X_i)$, and evaluate the quantile loss using the quantile check function at level τ:

$$\rho_\tau(u) = u\,(\tau - \mathbb{I}\{u < 0\})\,.$$

The overall metric on the test dataset $\{(X_i, Y_i)\}_{i=1,\ldots,n}$ is computed as $\frac{\sum_{i=1}^{n} \rho_\tau(u_i)}{n}$.

As $n \to \infty$, this metric is minimized exactly when the predicted quantile matches the true conditional τ-quantiles of $Y|X$, so lower values indicate better calibration at the τ-quantile. An effective conditional model must produce well-calibrated quantile estimates across all values of τ. Therefore, in our experiments, we assess model performance across multiple values of τ, using the overall quality of quantile calibration to interpret the quality of the conditional models.

3.4 Benchmark Models

To evaluate the generative models' performance, we introduce a set of simple yet interpretable benchmark models. These baselines allow us to assess whether the more complex models yield meaningful improvements over traditional input modeling methods.

The simplest approach we consider is kernel density estimation (KDE). However, applying a naive KDE to conditional distributions introduces computational inefficiency: for each conditional variable X_i, we must sample from a mixture distribution with as many components as training samples. Additionally, computing the weight of each component requires evaluating density functions across all training points, leading to a heavy computational burden. Since most weights are negligible, we adopt the Nearest-Neighbor Kernel Density Estimation (NNKDE) method in (Chen 2017), which leverages the k nearest neighbors in KDE, to reduce computational costs.

In addition, we use an additional approach for each metric: for the Wasserstein distance, we compare against the train data empirical distribution of Y; for the average quantile loss, we use linear quantile regression as a baseline (Koenker and Hallock 2001). The quantile regression is referred to as QR in the tables.

4 EXPERIMENTAL RESULTS

We present the results separately for each dataset. We evaluate model performance using the two metrics introduced in Section 3.3.

For each dataset, our analysis begins with the Wasserstein distance, where we classify models as underperforming if they fail to achieve comparable results to both reference baselines: the empirical distribution and NNKDE. We then examine average quantile loss. Since quantile regression is specifically designed for quantile estimation, we expect it to achieve strong performance on the linear dataset and relatively good performance on the nonlinear and real-world datasets. Consequently, we consider a model to perform well if it either matches quantile regression's performance or surpasses NNKDE's results.

We present our results using both tables and figures. Boxplots are employed to illustrate the variation of the results. In each plot, the median is indicated by a central line, while the top and bottom edges of the box represent the 75th and 25th percentiles, respectively. The whiskers extend beyond the box to the

most extreme data points within 1.5 times the interquartile range (IQR). Data points outside this range are classified as outliers and marked with individual red points.

4.1 Method Comparison on Linear Datasets

Table 3 and figure 1 present the experimental results based on average Wasserstein distances. Notably, GAN and WGAN exhibit the poorest performance among all generative models. For example, in 5-dimensional settings, GAN and WGAN achieve Wasserstein distances of 96.2 and 41.15, respectively—significantly higher than the benchmark values of 13.08 and 11.5. This trend persists in higher dimensions (e.g., $d = 25$), where GAN (237.86) and WGAN (236.27) substantially underperform compared to benchmarks (20.88 and 10.76). Figure 1 further confirms that this gap is systematic rather than anomalous. These findings demonstrate that GAN and WGAN struggle to generate target data effectively regarless to the richness of data. In contrast, other models (VAE, Real NVP, DDPM, and WGAN-GP) perform competitively, closely matching the benchmark methods.

Table 3: Average Wasserstein distances between test data and model-generated samples across 10 replications for the linear synthetic dataset.

d	VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM	NNKDE	Empirical
5	14.37	96.02	41.15	22.07	15.02	16.65	13.08	**11.5**
10	16.62	108.55	122.01	20.07	15.88	18.84	20.22	**12.11**
15	14.65	224.95	184.67	19.34	12.19	15.67	17.29	**9.16**
20	21.37	214.06	269.21	22.75	16.45	18.17	14.34	**13.48**
25	19.08	237.86	236.37	25.6	16.6	21.11	20.88	**10.76**

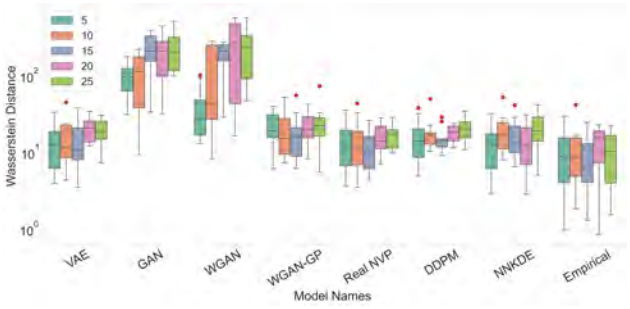

Figure 1: Boxplot of Wasserstein distances between test data and model-generated samples across 10 replications for the linear synthetic dataset.

Next, we analyze the average quantile loss metric results shown in Table 4 and Figure 2. Table 4 reveals that VAE, Real NVP, and DDPM consistently outperform both benchmarks and other generative models across dimensions $d = 10$ to 25. In particular, VAE achieves the lowest scores at smaller dimensions ($d = 5$: 2.27×10^2, $d = 10$: 3.12×10^2), while Real NVP dominates at higher dimensions ($d = 15$: 3.82×10^2, $d = 25$: 4.71×10^2). DDPM demonstrates strong performance, particularly at $d = 20$ (4.10×10^2). Figure 2 highlights the stability of DDPM, Real NVP, and VAE, which exhibit performance comparable to the benchmark quantile regression. This is reflected in their similarly compact IQR and significantly lower variability compared to NNKDE, whose boxplot displays a larger IQR and greater dispersion.

Table 4: Average quantile loss across 10 replications for the linear synthetic dataset.

d	VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM	QR	NNKDE
5	2.27×10^2	5.87×10^2	2.80×10^2	2.66×10^2	$\mathbf{2.27 \times 10^2}$	2.38×10^2	2.30×10^2	2.29×10^2
10	$\mathbf{3.12 \times 10^2}$	8.29×10^2	9.07×10^2	3.39×10^2	3.27×10^2	3.21×10^2	3.32×10^2	3.82×10^2
15	3.93×10^2	1.63×10^3	1.03×10^3	4.47×10^2	$\mathbf{3.82 \times 10^2}$	3.86×10^2	4.06×10^2	4.97×10^2
20	4.58×10^2	1.54×10^3	1.90×10^3	4.79×10^2	4.14×10^2	$\mathbf{4.10 \times 10^2}$	4.71×10^2	5.71×10^2
25	4.87×10^2	1.65×10^3	1.56×10^3	5.33×10^2	$\mathbf{4.71 \times 10^2}$	4.80×10^2	5.12×10^2	6.95×10^2

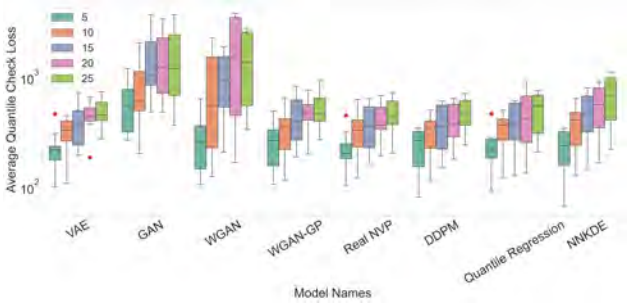

Figure 2: Boxplot of average quantile loss across 10 replications for the linear synthetic dataset.

The results on the linear dataset indicate that DDPM, Real NVP, and VAE perform competitively on both metrics. In contrast, GAN-based models exhibit poor performance, though GAN-GP shows some improvement—albeit still moderate compared to the benchmark and other models.

4.2 Method Comparison on Nonlinear Datasets

Table 5 and figure 3 show Wasserstein distance results. GAN and WGAN perform worst among generative models, e.g., at $d = 5$, scoring 3.12 and 2.57 vs. benchmarks 0.24 and 0.15. At $d = 25$, GAN (7.09) and WGAN (7.76) lag behind benchmarks (20.88 and 10.76). WGAN-GP demonstrates improvements over GAN-based models in lower dimensional settings but underperforms at higher dimensions, such as $d = 25$, where it scores 9.8. Figure 3 confirms this consistent performance gap. GAN-based models generally struggle with data generation, while VAE, Real NVP, and DDPM closely align with benchmark performance across dimensions.

Table 5: Average Wasserstein distances between test data and model-generated samples across 10 replications for the nonlinear synthetic dataset.

d	VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM	NNKDE	Empirical
5	0.16	3.12	2.57	1.23	0.17	0.17	0.24	**0.15**
10	**0.22**	7.07	7.32	3.52	0.28	**0.22**	0.86	0.23
15	0.26	7.48	7.69	2.71	0.38	0.28	1.59	**0.25**
20	**0.23**	7.21	5.5	5.3	0.48	0.24	2.16	**0.23**
25	0.29	7.09	7.76	9.8	0.47	**0.22**	2.49	0.28

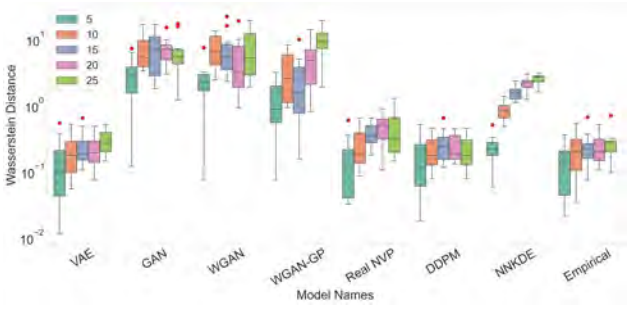

Figure 3: Boxplot of Wasserstein distances between test data and model-generated samples across 10 replications for the nonlinear synthetic dataset.

We next analyze the average quantile loss metric results presented in table 6 and figure 4. Table 6 indicates that VAE, Real NVP, and DDPM consistently outperform both benchmark methods and other generative models across dimensions $d = 5$ to 25. Specifically, DDPM achieves the best performance at dimensions 5, 15, 20, and 25, with scores of 1.75, 3.54, 3.93, and 4.87, respectively, while VAE outperforms other generative models at dimension 10, achieving a score of 3.31. Figure 4 illustrates the stability of VAE and DDPM compared to benchmark methods and other generative models.

Table 6: Average quantile loss across 10 replications for the nonlinear synthetic dataset.

d	VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM	QR	NNKDE
5	1.96	1.91×10^1	1.29×10^1	4.59	1.83	**1.75**	5.43	4.66
10	**3.31**	4.18×10^1	4.74×10^1	1.11×10^1	3.75	3.32	1.10×10^1	1.20×10^1
15	3.78	4.77×10^1	5.02×10^1	1.41×10^1	4.65	**3.54**	1.51×10^1	1.79×10^1
20	4.31	4.77×10^1	4.30×10^1	1.80×10^1	6.08	**3.93**	1.76×10^1	2.19×10^1
25	5.56	4.18×10^1	5.62×10^1	3.72×10^1	8.86	**4.87**	1.99×10^1	2.49×10^1

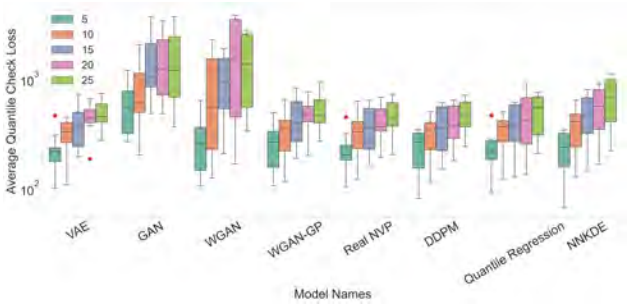

Figure 4: Boxplot of average quantile loss across 10 replications for the nonlinear synthetic dataset.

The results on the nonlinear dataset indicate that DDPM outperforms other generative models and benchmark methods. In contrast, Real NVP does not perform as well as it does on the linear dataset. Meanwhile, GAN-based models exhibit poor performance, and the gradient penalty method fails to effectively improve their performance under complex relational conditions.

4.3 Method Comparison on Real-world Dataset

Table 7 and figure 5 evaluate generative models using the Wasserstein distance metric. We can observe that the GAN (8.43×10^5) performs the worst among all models. However, replacing the original GAN loss function with the Wasserstein distance loss effectively improves the model's performance, proved by

WGAN (2.67×10^5) and WGAN-GP (2.71×10^5). This is also evident in Figure 5, where WGAN and WGAN-GP consistently outperform the standard GAN model.

Table 7: Average Wasserstein distances between test data and model-generated samples for the real-world dataset.

VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM	NNKDE	Empirical
7.47×10^4	8.43×10^5	2.67×10^5	2.71×10^5	7.47×10^4	5.22×10^4	4.00×10^4	$\mathbf{2.71 \times 10^4}$

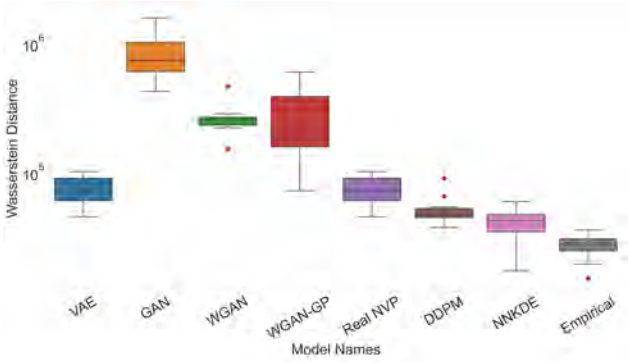

Figure 5: Boxplot of Wasserstein distances between test data and model-generated samples for the real-world dataset.

Next, we analyze the average quantile loss metric results in table 8 and figure 6. We can observe that the VAE (8.73×10^5) and the DDPM (8.68×10^5) significantly outperform other generative models and benchmarks, with DDPM being the best-performing model. Figure 6 shows the stability of the performance of VAE and DDPM.

Table 8: Average quantile loss for the real-world dataset.

VAE	GAN	WGAN	WGAN-GP	Real NVP	DDPM	QR	NNKDE
8.73×10^5	5.87×10^6	2.64×10^6	1.71×10^6	1.19×10^6	$\mathbf{8.68 \times 10^5}$	2.40×10^6	2.60×10^6

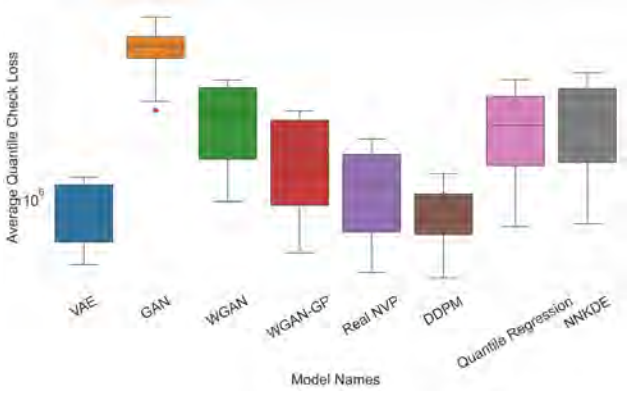

Figure 6: Boxplot of average quantile loss for the real-world dataset.

In conclusion, we can determine that VAE and DDPM exhibit strong performance in generating real-world data with complex relationships, which is consistent with the conclusions we obtain on the nonlinear dataset.

5 DISCUSSION

In our experiments, models leveraging neural networks for parameterization (e.g., VAEs and DDPMs) not only demonstrated consistent performance across all tasks but also outperformed the benchmark methods. The Real-NVP normalizing flow model achieved moderate results, while GAN-type models exhibited less stability. This hierarchy can be interpreted through architectural differences: VAEs and DDPMs employ less restrictive frameworks compared to normalizing flows (which require exact invertibility constraints). These models also benefit from optimizing a tractable lower bound on the marginal likelihood (ELBO), enabling stable training through variational approximation. This formulation simplifies optimization to a single minimization objective, avoiding the adversarial minmax game inherent to GAN training. However, we emphasize that our experimental scope remains limited, and broader evaluations would help confirm these observations.

Additionally, practitioners often face data-scarce scenarios and distribution drift. Several works in machine learning address these challenges, such as research on small datasets (Alpaydin 2023) and GAN performance in drifting environments (Guzy et al. 2021). Future work will further investigate these issues.

ACKNOWLEDGMENTS

We gratefully acknowledge support from the National Natural Science Foundation of China under Grant No. 72301195.

REFERENCES

Alpaydin, E. 2023. "A Survey on Deep Learning Tools Dealing with Data Scarcity". *Machine Learning*.

Alsaheal, A., W. Hoy, P. Haro, A. Rehman, and N. Celik. 2024. "GenAir: Generative AI for Resilient Urban Air Mobility with VTOLs in Disaster Evacuation". In *2024 Winter Simulation Conference (WSC)*, 537–548 https://doi.org/10.1109/WSC63780.2024.10838767.

Arjovsky, M., S. Chintala, and L. Bottou. 2017. "Wasserstein Generative Adversarial Networks". In *International conference on machine learning*, 214–223. PMLR.

Biller, B., and C. Gunes. 2010. "Introduction to Simulation Input Modeling". In *proceedings of the 2010 Winter Simulation Conference*, 49–58 https://doi.org/10.1109/WSC.2010.5679176.

Cen, W., E. A. Herbert, and P. J. Haas. 2020. "Nim: Modeling and Generation of Simulation Inputs via Generative Neural Networks". In *2020 Winter Simulation Conference (WSC)*, 584–595 https://doi.org/10.1109/WSC48552.2020.9383966.

Chen, Y.-C. 2017. "A Tutorial on Kernel Density Estimation and Recent Advances". *Biostatistics & Epidemiology* 1(1):161–187.

Dinh, L., J. Sohl-Dickstein, and S. Bengio. 2016. "Density Estimation Using Real Nvp". *arXiv preprint arXiv:1605.08803*.

Engelmann, J., and S. Lessmann. 2021. "Conditional Wasserstein GAN-based Oersampling of Tabular Data for Imbalanced Learning". *Expert Systems with Applications* 174:114582.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, *et al*. 2014. "Generative Adversarial Nets". *Advances in neural information processing systems* 27.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. 2017. "Improved Training of Wasserstein Gans". *Advances in neural information processing systems* 30.

Guzy, F., M. Woźniak, and B. Krawczyk. 2021. "Evaluating and Explaining Generative Adversarial Networks for Continual Learning Under Concept Drift". In *2021 International Conference on Data Mining Workshops (ICDMW)*, 295–303.

Henderson, S. G., and B. L. Nelson. 2006. *Handbooks in Operations Research and Management Science: Simulation*. Elsevier.

Ho, J., A. Jain, and P. Abbeel. 2020. "Denoising Diffusion Probabilistic Models". *Advances in neural information processing systems* 33:6840–6851.

Kingma, D. P., and J. Ba. 2014. "Adam: A Method for Stochastic Optimization". *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., and M. Welling. 2013. "Auto-encoding Variational Bayes". *arXiv preprint arXiv:1312.6114*.

Koenker, R., and G. Bassett Jr. 1978. "Regression Quantiles". *Econometrica: journal of the Econometric Society*:33–50.

Koenker, R., and K. F. Hallock. 2001. "Quantile regression". *Journal of economic perspectives* 15(4):143–156.

Kuhl, M. E., N. M. Steiger, E. K. Lada, M. A. Wagner, and J. R. Wilson. 2006. "Introduction to Modeling and Generating Probabilistic Input Processes for Simulation". In *Proceedings of the 2006 Winter simulation Conference*, 19–35 https://doi.org/10.1109/WSC.2006.323035.

Kuiper, P., S. Lin, J. Blanchet, and V. Tarokh. 2024. "Generative Learning for Simulation of Vehicle Faults". In *2024 Winter Simulation Conference (WSC)*, 2106–2117 https://doi.org/10.1109/WSC63780.2024.10838724.

Law, A. M. 2014. *Simulation Modeling and Analysis*. 5th ed. New York, NY: McGraw-Hill Education.
Mirza, M., and S. Osindero. 2014. "Conditional Generative Adversarial Nets". *arXiv preprint arXiv:1411.1784*.
Montevechi, J. A. B., A. T. Campos, G. T. Gabriel, and C. H. dos Santos. 2021. "Input Data Modeling: An Approach Using Generative Adversarial Networks". In *2021 Winter Simulation Conference (WSC)*, 1–12 https://doi.org/10.1109/WSC52266.2021.9715407.
Nelson, B. 2013. *Foundations and Methods of Stochastic Simulation: A First Course*. Springer Science & Business Media.
Sohn, K., H. Lee, and X. Yan. 2015. "Learning Structured Output Representation Using Deep Conditional Generative Models". *Advances in neural information processing systems* 28.
Song, J., C. Meng, and S. Ermon. 2020a. "Denoising Diffusion Implicit Models". *arXiv preprint arXiv:2010.02502*.
Song, Y., J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. 2020b. "Score-based Generative Modeling Through Stochastic Differential Equations". *arXiv preprint arXiv:2011.13456*.
Winkler, C., D. Worrall, E. Hoogeboom, and M. Welling. 2019. "Learning Likelihoods with Conditional Normalizing Flows". *arXiv preprint arXiv:1912.00042*.
Zaheer, R., and H. Shaziya. 2019. "A Study of the Optimization Algorithms in Deep Learning". In *2019 third international conference on inventive systems and control (ICISC)*, 536–539.

AUTHOR BIOGRAPHIES

ZHOU MIAO is currently a graduate student in the Department of Aeronautical and Aviation Engineering, The Hong Kong Polytechnic University, Hong Kong. His research interests include mixed integer programming, generative models and network flow modeling. His email address is 23123805r@connect.polyu.hk.

ZHIYUAN HUANG is an assistant professor the Department of Management Science and Engineering at the Tongji University. His research interests include simulation and stochastic optimization. His email address is huangzy@tongji.edu.cn.

ZHAOLIN HU is a professor in the School of Economics and Management at Tongji University. His research interests include stochastic optimization, simulation theory and practice, machine learning, and risk management. He is currently an associate editor of *Journal of Management Science and Engineering* and *Journal of the Operations Research Society of China*. His email address is russell@tongji.edu.cn.