Can GANs Learn the Stylized Facts of Financial Time Series?

Sohyeon Kwon
NCSOFT
Seongnam, Republic of Korea
sohyeonk@ncsoft.com

Yongjae Lee
Ulsan National Institute of Science and Technology
(UNIST)
Ulsan, Republic of Korea
yongjaelee@unist.ac.kr

ABSTRACT

In the financial sector, a sophisticated financial time series simulator is essential for evaluating financial products and investment strategies. Traditional back-testing methods have mainly relied on historical data-driven approaches or mathematical model-driven approaches, such as various stochastic processes. However, in the current era of AI, data-driven approaches, where models learn the intrinsic characteristics of data directly, have emerged as promising techniques. Generative Adversarial Networks (GANs) have surfaced as promising generative models, capturing data distributions through adversarial learning. Financial time series, characterized "stylized facts" such as random walks, mean-reverting patterns, unexpected jumps, and time-varying volatility, present significant challenges for deep neural networks to learn their intrinsic characteristics. This study examines the ability of GANs to learn diverse and complex temporal patterns (i.e., stylized facts) of both univariate and multivariate financial time series. Our extensive experiments revealed that GANs can capture various stylized facts of financial time series, but their performance varies significantly depending on the choice of generator architecture. This suggests that naively applying GANs might not effectively capture the intricate characteristics inherent in financial time series, highlighting the importance of carefully considering and validating the modeling choices.

KEYWORDS

Generative Adversarial Networks (GANs); financial time series; stylized facts

1 INTRODUCTION

The simulation of financial time series is crucial for designing investment strategies and financial products. Two approaches are commonly used in practice. A naïve approach involves testing strategies or products solely on historical data. However, as noted by [Assefa et al. 2020], this approach suffers from a lack of data. An alternative approach is to develop mathematical models of financial time series based on both academic research and practical observations of their properties. This model-based approach began with the use of simple Brownian motion to represent the unpredictable nature of stock prices. Over time, various models have been developed, including the Ornstein-Uhlenbeck (OU) process for mean-reverting behaviors, Jump Diffusion processes to account for occasional jumps, and the Heston model to capture time-varying volatilities. While these models are effective at reflecting known properties of financial time series, they can become highly complex when more realistic simulations are sought. Consequently,

estimating the parameters of these sophisticated models becomes increasingly challenging.

Deep generative models have recently emerged as strong alternatives for simulating financial time series. Unlike model-driven approach, these models adopt a data-driven approach, learning various intrinsic characteristics of financial time series directly from the data without requiring specific modeling of certain properties. Among the various models, Generative Adversarial Networks (GANs) proposed by [Goodfellow et al. 2020] have gained significant attention by employing an adversarial training mechanism between two distinct neural networks: a generator and a discriminator. GANs have demonstrated exceptional performance in image and text generation (e.g. [Karras et al. 2019]; [Ramesh et al. 2021]). Furthermore, there have been attempts to utilize GANs for time series generation, including financial time series (e.g., [Yu et al. 2017]; [Yoon et al. 2019]). Notable examples of finance-specific GANs include QuantGANs [Wiese et al. 2020], Tail-GAN [Cont et al. 2022] and Fin-GAN [Vuletić et al. 2024].

Most previous studies have proposed GANs that generate log return distributions of financial time series, closely matching the shape of these distributions. **However, can financial time series generated by these GAN models be used to develop robust hedging strategies or investment models that cover both occasional and unanticipated situations and the sequence of such events?** Practically applicable GAN models must be able to mirror various patterns seen in real data, including random walks, mean reversion, jumps, and time-varying volatilities.

In this paper, we aim to verify the ability of GANs to learn various stylized facts inherent in financial time series: random walks, mean reversions, jumps, and time-varying volatilities. Each of these properties is represented by a specific stochastic process. For instance, the random walk property is modeled by Brownian motion or Geometric Brownian motion, while the Ornstein-Uhlenbeck process characterizes mean reversion. Jump diffusion models capture jump behaviors, and the Heston model depicts time-varying volatilities. Our goal is to evaluate whether GANs can accurately approximate these models, considering both return distributions and stylized facts.

2 RELATED WORK

2.1 Model-driven financial time series simulator

Traditionally, financial time series have been modeled using various stochastic processes, which are collections of random variables with a few parameters that distill the various characteristics of financial time series. Bachelier was the pioneer in modeling the prevalent randomness of financial time series by introducing the concept of Brownian motion. The Geometric Brownian motion,

which simulates a lognormal distribution, models random walks with geometric growth and eliminates the possibility of stock prices being negative. Stock prices exhibit a tendency to revert to their mean, and their distributions are typically leptokurtic, deviating from the normal distribution. This behavior is best captured by the mean-reverting Ornstein-Uhlenbeck process, which is used in option pricing model (e.g., [Stein and Stein 1991]). Real-world financial time series often exhibit heavy-tailed distributions, where extreme price movements occur more frequently than would be expected under a normal distribution, posing challenges for traditional models. In this regard, [Merton 1976] developed the Jump Diffusion model, which incorporates the Poisson process for price jumps and successfully simulates both normal and abnormal events [Kou 2007]. Additionally, financial markets are characterized by volatility clustering, where periods of high volatility are followed by high volatility and low volatility by low volatility, which reflects the time-varying nature of risk in financial markets.

2.2 GANs in finance

In recent years, the remarkable performance of GANs in various fields, including text and image generation, has spurred interest in their applicability within the financial sector. [Wiese et al. 2020] proposed QuantGAN, specifically tailored for financial time series generation, which uses a stacked temporal convolutional network to capture long-range temporal dependencies. [Koshiyama et al. 2021] used conditional GANs for calibrating trading strategies, integrating the generated samples into ensemble modeling. Meanwhile, [Pardo and López 2019] trained a Wasserstein GAN with gradient penalty for data augmentation, demonstrating that synthetic data can mitigate overfitting and enhance trading strategies. [Cont et al. 2022] developed Tail-GAN by refining the loss function using the joint elicitability of tail risk information, such as value-at-risk and expected shortfall. The Fin-GAN, introduced by [Vuletić et al. 2024], employs a novel economics-driven loss function to facilitate the forecasting of financial time series. On the other hand, [Kim et al. 2023] utilized the discriminator part of GANs to detect anomalies in financial time series, and [Kim et al. 2024] applied it to the mean-variance model to enable dynamic control of model robustness. For more examples of application of ML/AI models (including GANs) in finance, see [Lee et al. 2023]

3 MODELS

In this section, we examine the capability of GANs to capture the temporal structures of financial time series, as represented by the stochastic processes discussed in Section 2.1. While GANs do not require a specific architecture for the neural network, their performance varies depending on the chosen architecture. Additionally, GANs are known to be highly sensitive to hyperparameter configurations, particularly the learning rates for both the generator and discriminator. Therefore, the primary focus of our experiments is: (1) experimenting with various neural network architectures, and (2) formulating distribution distance-based objective functions for hyperparameter optimization.

3.1 Model architectures

GANs operate as a data generation framework via adversarial training and do not assume any specific network architecture. While the discriminator usually employs a simple architecture, the generator varies widely depending on the research objectives and the characteristics of the data. For instance, [Esteban et al. 2017] in RGAN and [Wiese et al. 2020] in QuantGAN, employed recurrent neural networks (RNN) and temporal convolutional networks (TCN) respectively, to encapsulate the temporal structure of time series data. Conversely, [Takahashi et al. 2019] experimented with multiple architectures, including multi-layer perceptrons (MLP), convolutional neural networks (CNN), and a hybrid of the two, MLP-CNN.
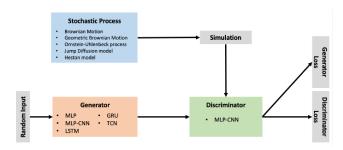

Figure 1: Experiment overview

Therefore, we tested the following five different architectures for the generator: Multi-Layer Perceptron (MLP), Multi-Layer Perceptron – Convolutional Neural Network (MLP-CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Temporal Convolutional Network (TCN). For simplicity, We fixed the architecture of the discriminator as MLP-CNN. An overview of the experiments in this study is shown in **Figure 1**.

3.2 Hyperparameter optimization

GAN is extremely sensitive to hyperparameter settings, especially the learning rates of the generator and discriminator. Consequently, a hyperparameter search is essential to improve the model's performance. In image generation, the quality of the generated images can be evaluated using the bias and variance of Fréchet Inception Distance, which measures the similarity between distribution of real and generated image features. However, there is currently no widely accepted metric for evaluating the performance of GANs in generating time series, nor is there an established objective for minimizing hyperparameter optimization in this context.

To measure the distance between the real and generated distributions in the context of hyperparameter optimization, we can use a representative metric such as the Kullback-Leibler (KL) divergence. KL divergence is a common measure of the difference between two probability distributions and can be employed as an objective function for hyperparameter optimization in GAN-based time series generation. This approach allows us to quantify how well the generated data approximates the real data, thereby guiding the optimization process to improve the quality of the generated time series. $D_{KL}(P||Q)$ measures the average information entropy required for approximating distribution Q to model the actual distribution P and is defined as

$$D_{KL}(P||Q) = -\int P(x) \ln \left\{ \frac{q(x)}{p(x)} \right\} dx \qquad (1)$$

However, KL-divergence cannot measure how far the two probability distributions are due to its asymmetricity. Alternatively, a symmetrized and smoothed version of KL-divergence, Jensen-Shannon divergence, can be used to calculate the distance between the two distributions. Jensen-Shannon divergence lets M be an average distribution of P and D, and it quantifies the average of $D_{KL}(P||M)$ and $D_{KL}(Q||M)$.

$$JSD(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M) \qquad (2)$$

$$where \ M = \frac{1}{2}(P + Q)$$

To define the objective function for hyperparameter search, we use a linear combination of the Jensen-Shannon divergence between the log return distributions of the real and generated data, and the divergence between their final value distribution. The log return distribution reflects the volatility of the financial time series, which is a crucial feature when evaluating a financial time series simulator. Including the final value distribution in the objective function helps to prevent model collapse and retain the scale range of the generated data. Empirical weights are assigned to the log return and final value distributions to balance their contributions to the objective function effectively. Let R and F be the distribution of log return and final value, respectively, and the objective function is

$$min(4 * JSD(R_{real}||R_{generated})) + (1 * JSD(F_{real}||F_{generated})) \quad (3)$$

In the case of multivariate time series, it is essential to capture not only the return and final value distributions but also the dependencies between variables during GAN training. Accurately modeling these dependencies ensures that the generated time series reflects the complex interactions present in the real data, which is crucial for applications such as portfolio optimization and risk management. To address this, the objective function for hyperparameter search should include metrics that assess the quality of these dependencies, ensuring that the generated data maintains the inherent correlations and relationships between variables. Let N_0 and N_1 be the independent two normal distributions with the same dimension k, then KL-divergence for multivariate normal distributions is defined as

$$D_{KL}(N_0||N_1) = \qquad (4)$$

$$\frac{1}{2}(tr(\Sigma_1^{-1}\Sigma_0)) - k + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) + \ln\left(\frac{det\Sigma_1}{det\Sigma_0}\right)$$

Including the marginal and bivariate distribution information, the hyperparameter is searched by the following function.

$$min(2 * D_{KLmultivariate}) \qquad (5)$$

$$+(4 * (JSD(R_{realx1}||R_{generatedx1}) + JSD(R_{realx2}||R_{generatedx2})))$$

$$+JSD(F_{realx1}||F_{generatedx1}) + JSD(F_{realx2}||F_{generatedx2})$$

4 EXPERIMENT

4.1 Data description

For modeling temporal structures of financial time series, this study employs five stochastic processes as introduced in Section 2.1: Brownian motion (BM), Geometric Brownian motion (GBM), Ornstein-Uhlenbeck process (OU), Jump diffusion model (JD), and Heston model (HT). While historical real financial time series contain multiple inherent features, these features may not occur frequently, leading to a lack of data. However, with the availability of parameters, stochastic processes can be simulated consistently, providing a valuable tool for generating synthetic financial time series data that can capture the characteristics of the real-world data.

For explicit evaluation of the learning ability of GANs in financial time series, we generate each stochastic process with different parameter settings. For Brownian Motion, we vary the rate of change and the scale factor of the Wiener process. In the case of Geometric Brownian Motion (GBM), we adjust the drift and volatility parameters. For the Ornstein-Uhlenbeck (OU) process, the reverting speed and volatility are varied. The Jump Diffusion model is configured with different mean and standard deviation values for the normal distribution to describe jumps. Lastly, for the Heston (HT) model, we modify the volatility of volatility. Each stochastic process, except for the HT model, has six different parameter settings, while the HT model has three parameter settings. To assess the model's performance based on sequence length, we create each stochastic process with three different sequence lengths: 25, 50, and 100. For the multivariate case, a bivariate Geometric Brownian motion is generated with two different correlation settings, high and low. The correlation between the variables is inherently generated within the multivariate normal distribution for the Wiener process of Geometric Brownian motion, with correlations set to approximately 0.9 and 0. The sequence length for the multivariate case is set to 50, and the drift and volatility for both variables (x1 and x2) are set to be different, 0.5 and 0.1, respectively, resulting in distinct return distribution shapes for each variable.

4.2 Evaluation method

We assess the quality of the synthesized financial time series from two perspectives: distribution evaluation and the characteristics of each stochastic process. Unfortunately, a widely accepted metric for evaluating the performance of time series generation models is unavailable. However, as Goodfellow et al., 2014 emphasize, GANs are successful in learning the underlying distribution of the training data. Therefore, we utilize the Jensen-Shannon divergence, as described in **Eq. 2**, as a distance measure between the distributions of the real and generated data. This metric is estimated for the overall evaluation of the model's performance.

As the length of the Wiener process used as training data in this experiment is comparatively short, directly estimating the values of its parameters from the generated data is challenging. Therefore, we indirectly approximate the features of each stochastic from the generated data. For Brownian motion and Geometric Brownian motion, which both describe the randomness of the stock prices, we approximate their statistical properties, such as mean, standard deviation, maximum, and minimum of log return values. To approximate the reverting speed and count to the long-term
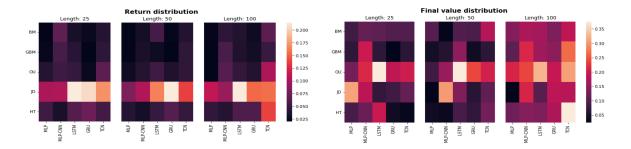
Figure 2: Jensen-Shannon divergence heatmap of log return and final value distribution. Each value is the mean of every parameter type of each stochastic process.
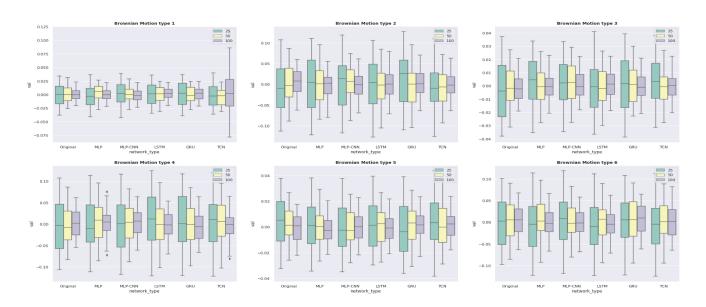

Figure 3: Box plots of 6 types of the Brownian motion. The first three boxes in each type frame refer to the statistical properties of the original case corresponding length. The following three boxplots are the statistical properties of synthetic data generated by each network. The types vary in terms of (mean, std) of Wiener process, with the parameters set to (0, 10), (0, 30), (2.5, 10), (2.5, 30), (5, 10), (5, 30).

mean of the Ornstein-Uhlenbeck process, we define the range of the long-term mean as $100 \pm margin$ and quantify the reverting speed by measuring the timesteps between two reverting events. We calculate the count as the number of reverting events in each time series. For the Jump Diffusion model, we estimate the size and count of jump events in each time series. Since classifying whether the value of each time step is in a jump or normal status is ambiguous, we define the value as a jump if $r_t \geq \mu_{\text{jump distribution}} - \sigma_{\text{jump distribution}}$ where r_t refers to the return at time step t. Lastly for the Heston model, we estimate the volatility by rolling every five days and quantify its mean, standard deviation, maximum, and minimum values.

For the evaluation of multivariate time series, it is significant to learn the intrinsic characteristics of each variable and the dependencies between variables. The Jensen-Shannon divergence is estimated separately for each variable, x1 and x2, for independent variable evaluation. Variable dependency of multivariate time series is revealed in the bivariate distribution and correlation between variables. Therefore, the shape of the bivariate distribution is compared, and the correlations are quantified for comparison.

4.3 Results

4.3.1 Univariate time series generation. As GAN learns the distribution of the training data, the Jensen-Shannon divergence is used to estimate the distances between the log return and final value distributions, as shown in **Figure 2**. The length dependency is evident in the final value distribution, whereas there is little difference in return distribution between lengths. The Jensen-Shannon divergence of lengths 25 and 50 in the final value distribution of **Figure**
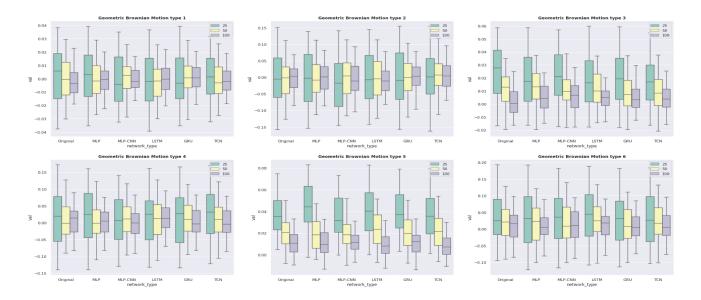
Figure 4: Box plots of 6 types of the Geometric Brownian motion. The first three boxes in each type frame refer to the statistical properties of the original case corresponding length. The following three boxplots are the statistical properties of synthetic data generated by each network. The types vary in terms of (μ, σ) of Wiener process, with the parameters set to (0, 0.1), (0, 0.4), (0.5, 0.1), (0.5, 0.4), (1.0, 0.1), (1.0, 0.4).

2 generally remains low, but the values of length 100 are comparatively high. This indicates that as the length increases, there is a greater possibility of the generated final values being outside of the original data range, even though the return distributions resemble each other. This may occur because a small difference in return value between generated and original data results in a more significant difference as the error accumulates. Among the stochastic process types, Geometric Brownian motion shows the best learning performance both in log return distribution and final value distribution, with average values of 0.12913 and 0.46003, respectively, across the different lengths of the time series. As clearly shown in **Figure 2**, the log return distribution of generated data in the case of the Jump Diffusion model shows the most prominent distribution distance to that of the training data, with a value of 0.52243, regardless of the time series length. The final values of the Ornstein-Uhlenbeck process form a distribution that is the most dissimilar to the training data, quantified with an average distance value of 0.68728.

The results suggest that the GAN framework effectively learns both the distributions and features of Brownian Motion and Geometric Brownian Motion, which represent the randomness of stock behavior. However, the performance of the LSTM and TCN generators on the Ornstein-Uhlenbeck process and Jump Diffusion requires further analysis. The poorer performance of these generators on the Ornstein-Uhlenbeck process is particularly evident in the final value distribution in **Figure 2**. This process's reverting characteristics rely more on its past values than other features, and the final value strongly depends on the sequence of log returns. Although LSTM is a specialized neural network for sequential data,

empirical studies have shown that it performs poorly on the log return data. Therefore, the absence of accurate sequential information may lead to generated final values that fall outside the range of the original data.

Figure 3 shows the statistical properties of the log return of both original and generated data by all generators. The overall box plots of generated data are similar to those of the original data. However, in some cases, such as type 1 generated by the TCN generator, the length 100 cases are not generated as accurately as the original data, producing log return data with large maximum, minimum, and standard deviation values. Additionally, some outliers are detected in type 4 generated by the MLP and TCN generators.

The characteristics of the Geometric Brownian motion are shown in **Figure 4**. Similar to Brownian motion, the statistical properties estimate different parameter settings for Geometric Brownian motion. Although Geometric Brownian motion is also based on the randomness of its movement, the box plots of generated data are exceptionally similar to those of the original data. The training data for Geometric Brownian motion features strong drift in its up-phase trends, especially in types 3 and 5, and this characteristic is well captured in the generated data.

Table 1 summarizes the results for the Ornstein-Uhlenbeck process, including the estimated reverting speed and the number of reverting cases. Among the various generator architectures, MLP and LSTM are the most effective at reproducing a reverting speed close to that of the original data, while TCN shows limited performance in learning this feature. However, the results of the LSTM generator are somewhat contradictory since its Jensen-Shannon

Length	Type	Reverting Speed						Reverting Count					
		Original	MLP	MLP-CNN	LSTM	GRU	TCN	Original	MLP	MLP-CNN	LSTM	GRU	TCN
25	1	3.103	1.820	1.939	**2.464**	1.943	1.835	4.850	**3.892**	3.053	2.232	3.703	3.644
	2	3.481	**3.409**	2.858	2.683	3.050	3.088	2.303	1.250	1.322	**1.682**	1.276	1.298
	3	3.359	**2.487**	2.069	1.767	1.757	2.058	5.530	**3.794**	3.414	3.177	3.680	3.001
	4	4.145	3.514	3.071	2.980	**3.565**	3.354	2.587	1.393	1.443	1.714	1.474	**1.729**
	5	3.423	2.241	**2.269**	1.619	2.167	2.155	6.225	3.624	2.758	**5.112**	3.628	3.537
	6	4.662	**4.420**	3.829	4.188	3.968	13.882	2.982	1.782	**1.802**	1.033	1.522	1.583
100	1	4.301	2.326	**2.443**	2.289	2.019	2.170	17.894	12.670	12.410	6.602	**14.405**	13.891
	2	8.830	5.565	4.912	**7.088**	4.839	4.374	6.752	5.372	5.186	3.601	5.980	**6.617**
	3	4.395	2.909	2.721	**3.056**	1.859	2.819	20.971	15.481	11.590	4.858	**19.259**	14.424
	4	9.750	**6.400**	5.069	5.420	4.539	4.053	8.049	5.702	4.996	5.672	7.170	**7.471**
	5	4.185	2.105	2.257	**3.099**	2.142	2.006	24.076	14.875	**14.930**	4.822	13.874	9.387
	6	9.474	**5.418**	4.915	4.944	4.850	5.291	9.831	6.548	6.670	8.772	**8.797**	4.796

Table 1: The speed and count of reversion of the Ornstein-Uhlenbeck process. The closest values to the original cases are bold. The types differ in terms of (speed, volatility), with the parameters set to (1.0, 5.0), (1.0, 15.0), (2.5, 5.0), (2.5, 15.0), (5.0, 5.0), (5.0, 15.0).

Length	Type	Jump Count						Jump Size					
		Original	MLP	MLP-CNN	LSTM	GRU	TCN	Original	MLP	MLP-CNN	LSTM	GRU	TCN
25	1	2.430	3.242	**2.270**	1.499	1.816	1.498	0.012	0.016	**0.011**	0.008	0.009	0.008
	2	2.886	**2.827**	3.188	2.264	2.499	2.529	0.014	**0.013**	0.015	0.012	0.012	**0.013**
	3	2.394	1.613	**2.246**	1.619	1.474	2.127	0.017	0.012	**0.015**	0.011	0.012	**0.015**
	4	2.984	**2.999**	2.336	2.093	2.849	2.857	0.021	0.020	0.017	0.016	**0.021**	0.022
	5	2.276	3.700	**2.126**	1.686	2.123	2.897	0.023	0.034	0.019	0.019	**0.023**	0.029
	6	2.508	2.645	3.096	**2.485**	1.949	2.587	0.027	**0.027**	0.028	0.028	0.022	0.025
100	1	8.637	10.210	**7.356**	5.708	5.201	5.624	0.010	0.022	**0.008**	**0.008**	0.007	0.007
	2	9.791	8.720	7.980	6.504	**9.831**	8.312	0.011	0.009	0.009	0.009	**0.012**	0.010
	3	6.072	12.663	7.587	**5.509**	7.351	9.938	0.014	0.022	**0.014**	0.011	0.010	0.016
	4	6.751	9.961	8.629	**7.209**	7.966	6.180	0.017	**0.018**	0.015	0.014	0.015	0.011
	5	2.751	6.336	7.330	**5.154**	9.256	7.731	0.019	0.015	**0.017**	0.014	0.024	0.015
	6	2.657	9.691	8.969	6.301	**6.156**	6.582	0.022	0.024	**0.020**	0.016	0.016	0.017

Table 2: The count and size of jumps of the Jump diffusion model. The closest values to the original cases in each type and length are highlighted in bold. The parameters settings of the Jump Diffusion are mean and std of jump size and detailed settings are as follows: (0.1, 0.02), (0.1, 0.03), (0.15, 0.03), (0.15, 0.05), (0.2, 0.04), (0.2, 0.06)

divergence values are greater than those of other generator architectures. It's important to note that reverting speed is estimated only for the reverting case, without providing any information about the divergence or reversion of the time series. Therefore, the results of learning distributions do not precisely reflect the learning performance of each temporal structure. Although there is no clear tendency in learning reverting count between the network types, MLP and GRU perform best at generating time series with a reverting count closest to the original data. Interestingly, while TCN tends to produce reverting cases that occur by concentration, its reverting speed results are much smaller than those of the original data. Moreover, as the time series length increases, the error between the reverting speed of the original data and the generated

time series also increases, highlighting the limitations of the GAN framework in learning inherent reverting features.

We present the results of the generated Jump Diffusion model data, comparing the jump count and size to the original data in **Table 2**. For shorter lengths, MLP and MLP-CNN architectures show the best performance in training the number of jump cases within one time series, while LSTM and GRU networks perform better for longer lengths. In terms of jump size, there is no clear trend in learning across the different generator architectures, but the MLP-CNN architecture produces the closest jump sizes to the original data.

The statistical properties of the volatility of volatility in the Heston model are estimated and described in the box plot in **Figure 5**. The volatility of volatility parameter settings for the Heston
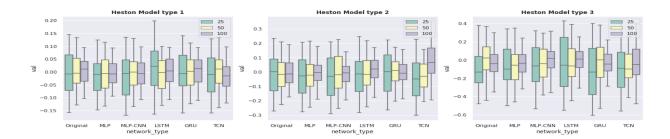
Figure 5: Box plots of 3 types of the Heston model. The first three box plots are statistical properties of original cases, and the following three boxes are the results of generated data corresponding networks. The types differ in terms of volatility of volatility, with the parameters set to be 0.01, 0.1, 1.0.

(a)   Highly correlated data

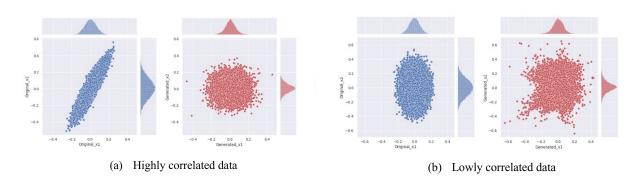
(b)   Lowly correlated data

Figure 6: Scatter plots of bivariate distribution and log return distribution of each variable. Blue plots refer to the original data, whereas red plots are generated data.

model are 0.01, 0.1, and 1.0 for types 1, 2, and 3, respectively. As the volatility of volatility increases, the box plots of the generated data deviate significantly from the box plot horizon of the corresponding length of original data. In Heston model type 1, each generator's general performance shows remarkable results except for one case: the maximum and minimum values generated by the LSTM generator with length 25 are far from those of the original data. The box plots of type 2 are within the acceptable range, except for the TCN generator with a length of 100. For this length, all components of the box plot are shifted upwards, and the distribution of volatility of volatility is more platykurtic than the original data. In contrast to types 1 and 2, the performance in type 3 is slightly poorer, especially for LSTM and GRU generators with shorter lengths and the TCN generators with longer time series. Although the mean of LSTM and GRU cases with a length of 25 is similar to that of the original data, the standard deviation is significant. Additionally, the box of the TCN generator is relatively larger than those of the other networks and the original data.

4.3.2   Multivariate time series generation. We also conducted GAN training on multivariate Geometric Brownian motion. The generator and discriminator are composed of simple MLP layers and trained with the Binary Cross Entropy loss function. The evaluation for multivariate data is conducted from two perspectives: for each independent variable and for variable dependency. Firstly,

	High correlated data	Lowly correlated data
x1	0.0702	0.0874
x2	0.306	.185

Table 3: The Jensen-Shannon divergence of the log return distribution.

the Jensen-Shannon divergence for each variable is estimated and presented in **Table 3**.

In the context of multivariate data learning, the performance of the plain GAN is limited in learning each variable and variable dependency, as shown in **Figure 6**. The generated distribution shapes of x1 and x2 do not resemble those of the original distribution. Additionally, as summarized in **Table 3**, the Jensen-Shannon divergence for variables x1 and x2 between the original and generated data is also imbalanced, indicating that the GAN tends to generate data dependent on only one variable, x1, which has smaller volatility than x2. One possible reason for this is that the discriminator does not separately distinguish between the two variables but instead provides feedback on both variables together. Consequently, the information on x2 with larger volatility may have been offset. The bivariate distribution between x1 and x2 is also very different from the original. While the high correlation is well described in

the original bivariate distribution that extends in one direction, the generated bivariate distribution forms a circular shape. In the low correlation setting, the bivariate distribution of the generated multivariate data is randomly spread, unlike the original, more concentrated data. The essential variable dependency feature, the correlation between x1 and x2, is also compared. For highly correlated data, the correlation between x1 and x2 is 0.8993 for the original data and 0.8249 for the generated data. Meanwhile, for the low correlation setting, the correlation between x1 and x2 is 0.001062 for the original data and -0.05633 for the generated data. Although the generated correlations are not exactly the same as the original correlations, the differences remain relatively low, 0.0744 and 0.057395, for high and low correlation cases, respectively. These results indicate that while the GAN can capture some aspects of the correlations between variables, it struggles to accurately reproduce the individual distributions and dependencies in the multivariate setting. This limitation suggests that further refinement of the GAN architecture or the training process may be necessary to better model multivariate financial time series.

5 CONCLUSION

In this paper, a vanilla GAN framework is investigated for its capability to learn significant temporal structures in financial time series rather than solely evaluating the GAN's learning performance. We have concluded several findings by conducting extensive experiments with various network architectures on five stochastic processes representing different temporal structures. In univariate data generation, the GAN models generally capture the different temporal structures but struggle with detailed features. The performance of GAN in learning the distributions and temporal structures of Brownian Motion and Geometric Brownian Motion is superior, yet some generator architectures, particularly the TCN-based generator, struggle to learn the reverting speed of the Ornstein-Uhlenbeck process. Additionally, only MLP and MLP-CNN-based generators can capture the two-hump-shaped distributions, whereas others create right-skewed distributions. For the Heston Model, the distributions of volatility of volatility are sometimes more platykurtic than the original data, and this trend worsens as the volatility of volatility increases. When it comes to multivariate data generation, the vanilla GAN framework is very limited in capturing the dependencies between multivariate time series. The generated distributions of x1 and x2 do not resemble the original distributions and look alike, indicating that the GAN tends to generate data dependent on only one variable, x1, which has smaller volatility than x2.

The cause of the limited replication of detailed features of the temporal structure remains for further study. It is premature to conclude that the adversarial framework of GAN is ineffective, as many other potential factors could affect the results, such as superficially constructed networks or sequence length. Although the GAN models in this experiment do not include complex neural network architectures, incorporating additional architectures, such as attention mechanisms, may result in better performance in learning time series dependencies. In summary, GAN models have shown potential as financial simulators without mathematical

assumptions, but the detailed replication of temporal structures or multivariate time series still requires additional attention.

REFERENCES

Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. 2020. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*. 1–8.
Rama Cont, Mihai Cucuringu, Renyuan Xu, and Chao Zhang. 2022. Tail-gan: Nonparametric scenario generation for tail risk estimation. *arXiv preprint arXiv:2203.01664* (2022).
Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. 2017. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633* (2017).
Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.
Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4401–4410.
Jang Ho Kim, Seyoung Kim, Yongjae Lee, Woo Chang Kim, and Frank J Fabozzi. 2024. Enhancing mean–variance portfolio optimization through GANs-based anomaly detection. *Annals of Operations Research* (2024), 1–28.
Seyoung Kim, Joohwan Hong, and Yongjae Lee. 2023. A GANs-Based Approach for Stock Price Anomaly Detection and Investment Risk Management. In *Proceedings of the Fourth ACM International Conference on AI in Finance*. 1–9.
Adriano Koshiyama, Nick Firoozye, and Philip Treleaven. 2021. Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quantitative Finance* 21, 5 (2021), 797–813.
Steve G Kou. 2007. Jump-diffusion models for asset pricing in financial engineering. *Handbooks in operations research and management science* 15 (2007), 73–116.
Yongjae Lee, John RJ Thompson, Jang Ho Kim, Woo Chang Kim, and Francesco A Fabozzi. 2023. An Overview of Machine Learning for Asset Management. *Journal of Portfolio Management* 49, 9 (2023).
Robert C Merton. 1976. Option pricing when underlying stock returns are discontinuous. *Journal of financial economics* 3, 1-2 (1976), 125–144.
Fernando De Meer Pardo and Rafael Cobo López. 2019. Mitigating overfitting on financial datasets with generative adversarial networks. *The Journal of Financial Data Science* (2019).
Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. (2021), 8821–8831.
Elias M Stein and Jeremy C Stein. 1991. Stock price distributions with stochastic volatility: an analytic approach. *The review of financial studies* 4, 4 (1991), 727–752.
Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. 2019. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications* 527 (2019), 121261.
Milena Vuletić, Felix Prenzel, and Mihai Cucuringu. 2024. Fin-gan: Forecasting and classifying financial time series via generative adversarial networks. *Quantitative Finance* (2024), 1–25.
Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. 2020. Quant GANs: deep generation of financial time series. *Quantitative Finance* 20, 9 (2020), 1419–1440.
Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. 2019. Time-series generative adversarial networks. *Advances in neural information processing systems* 32 (2019).
Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.