GANs for the Conditional Generation of Stochastic Processes in Finance

UCC Department of Mathematics

MSc. Financial & Computational Mathematics

Rudy Schneeweiss
Andreas Amann

November 28, 2023

ucc
Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland

Acknowledgements

I would like to acknowledge some of the people who helped me during my thesis project and throughout the year at UCC. I am extremely grateful to my supervisor, Andreas, for agreeing to take on my project and for his patience and guidance throughout the semester. A special thanks to my friends and family, especially my mom, who have all been unconditionally supportive. I would also like to thank all my fellow students and all the teachers of my various modules; it's been fun!

Summary

Asset price returns possess several properties that, when combined, make their modeling difficult. Through our analysis of these properties and review of past approaches, we introduce a new solution - a convolution-based generative adversarial network (GAN) with fewer layers than comparable models in the current literature. Our GAN can successfully replicate the distributional traits of traditional financial models and of real-world financial time series data. Additionally, we showcase the model's ability to extend into a conditional framework for generating time series under specific conditions.

Contents

1 Introduction

Asset returns in finance are unpredictable. They are subject to exogenous circumstances such as business news, political events, and pandemics. Even so, when examining returns data statistically, it consistently exhibits the same set of non-trivial properties known in the literature as *stylized facts* [1] [2]. Some noteworthy examples of these are:

- **Linear unpredictability**. The autocorrelations found in asset returns time series are insignificant; there is no information to be gained about future returns by looking at those in the past.
- **Heavy tails**. "Extreme" price movements occur more often than expected from a normally distributed random variable, giving asset returns a leptokurtic distribution.
- **Volatility Clustering**. High-volatility events cluster together in time. Generally, this feature means that in asset returns, large changes will follow large changes, and small changes will follow small changes.

Collecting all these properties together results in a hugely challenging process to model, to the vexation of many investors, as the pricing of option contracts relies on an accurate way to replicate the behavior of the underlying asset. The most widely known attempt at such a model is Black-Scholes [3], which uses geometric Brownian motion to produce linear unpredictability in its output. This model allows a closed-form solution to option pricing, yet it fails to reproduce the other listed stylized facts and does not truly reflect the behavior of asset returns [4]. Almgren states: *"The Brownian motion model is extremely popular, not primarily because of statistical evidence, but because it is only with this model that we can determine option prices exactly"* [5]. The search for a more accurate pricing model has led researchers down a series of different routes, such as:

- **Autoregressive Models**. ARCH/GARCH [6] [7] models estimate the change in asset return variance. This method permits the capture of non-constant volatility. Although widely used [8] [9], these struggle to replicate many of the stylized facts without extension [10] and are not without their well-documented practical issues [11].
- **Agent-Based Models**. ABM conducts a large-scale simulation using different agents placing orders at an order book level. This technique is capable of replicating many properties of asset returns [12] [13]; however, simulating at an agent level can be very computationally intensive for large systems. Furthermore, human agents often make subjective evaluations and exhibit irrational behavior, which is difficult to quantify [14].
- **Further Stochastic Processes**. In the vein of Black-Scholes, other models based on stochastic equations have been proposed, each addressing different aspects of market dynamics. Merton's jump-diffusion model [15] simulates large jumps in asset prices, which mimic real-life market shocks. Heston's stochastic volatility model incorporates an SDE that characterizes the asset's fluctuating volatility [16]. Cox, Ingersoll, and Ross provide a mean-reverting model to capture the stochastic nature of interest rates [17]. A multitude of others, such as SABR [18] and local volatility models [19] [20] have been developed, although even these advanced models are not without their inherent assumptions, as acknowledged by their creators.

Here, as a potential solution to the modeling problem, we turn to the field of deep learning and, in particular, GANs. Generative adversarial networks (GANs) [21] undertake a purely data-driven approach

to modeling their training sets and can be summarized as follows: One neural network, the generator, is set up to produce synthetic data. Another neural network, the discriminator, is set up to determine whether the data it receives comes from the real data set or the generator. By pitting these two networks against each other in a zero-sum game, we grow the capability of both, resulting in a generator that can produce outputs mimicking the properties of the training data almost exactly. GANs have shown a remarkable aptitude for modeling complex dependencies in images and videos [22] [23] [24] and have more recently been successfully applied to time series data [25] [26] [27]. Some prominent examples of GANs applied to financial time series are QuantGAN [28] and FIN-GAN [29], although the architecture used in these GANs is complex and computationally intensive to train.

In this dissertation, we examine the stylized facts in real market data, conducting our investigation in Section 2. We introduce GANs in Section 3. We discuss the mathematical motivation behind them, their strengths and weaknesses, and some further developments incorporated within our model design. In Section 4, we apply computationally inexpensive, shallow, convolution-based GANs to financial model data. We show that these GANs can learn the statistical properties inherent in data sets generated by traditional financial models.

Furthermore, we show that these GANs can be extended to a conditional framework in Section 5, thus allowing us to tune model parameters like volatility or interest rate to reflect current market climates. We show that even using (relative to convention) extremely shallow neural networks, the GAN can make inferences about these model parameters and produce statistically appropriate outputs. Finally, in Section 6, we apply our GAN architecture to real-world financial data, showing that it can learn the qualitative properties of real asset return data discussed in Section 2.

2 Characteristics of Financial Returns

Since qualitative properties act as evaluation criteria for asset return simulation, we need an understanding of these properties. This section will investigate the most important of the aforementioned stylized facts of asset returns. We do this by examining the daily closing price of the S&P500 index over 25 years, with data being sourced from *Yahoo! Finance*. This data has been extracted using the yfinance python module with some plotting functions adapted from Lewinson [30].

2.1 Returns vs. Asset Price

The data initially extracted is the closing price of our selected asset. Note that asset prices themselves are subject to scale. If we invest in two stocks that both rise in price by \$10, but the first stock was initially valued at \$5 and the second was initially valued at \$5000, the first stock was clearly a better investment. For this reason, the more relevant quantity for modeling is the asset's returns or log returns. The log return over a time period from $t-1$ to t is defined as $\ln(S_t/S_{t-1})$, and can therefore be derived from the closing prices. Figures 1 and 2 show the closing log price and the log returns over a 25-year period. In this dissertation, we will look at both log returns and asset prices.

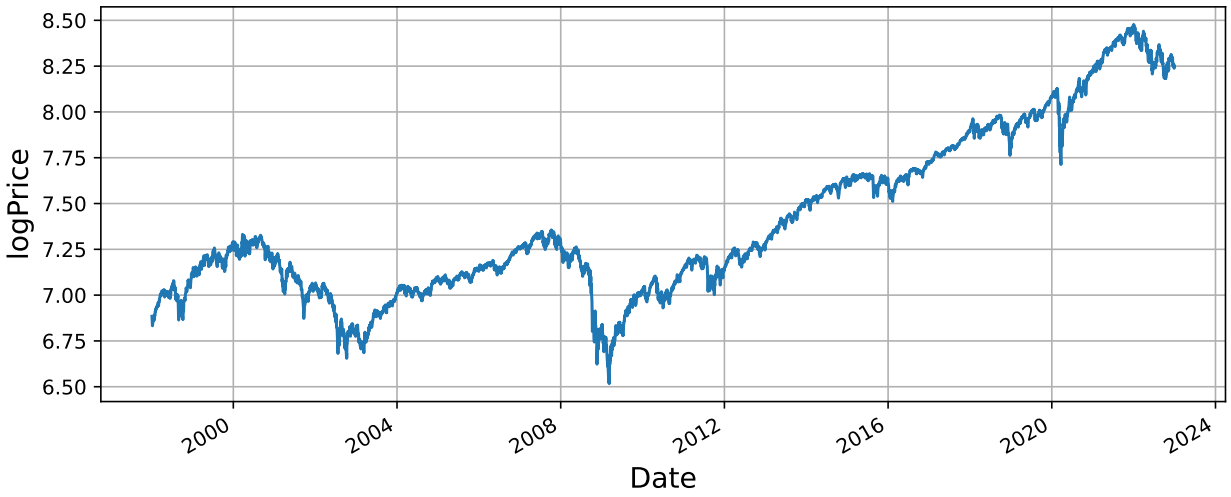


Figure 1: Closing log price of the S&P500 over 25 years

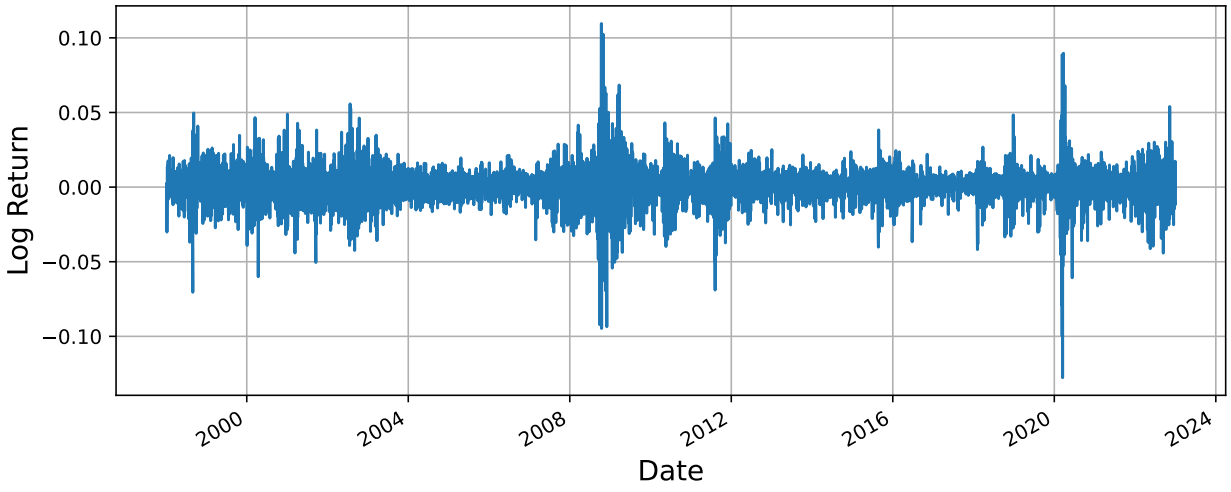


Figure 2: Log returns of the S&P500 over 25 years

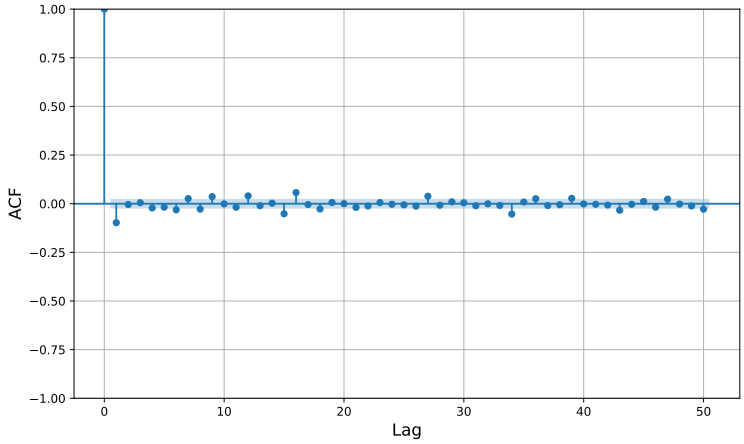


Figure 3: Negligible autocorrelation observed in returns

2.2 Stylized Facts

2.2.1 Absence of Autocorrelation in Returns

The autocorrelation function (ACF) measures the correlation of a time series with a lagged version of itself. Here, we observe that the autocorrelations of this asset return time series are insignificant, meaning that no part of it will look like a lagged version of itself. This aligns with our expectation since any autocorrelation in returns would constitute an obvious arbitrage opportunity. Refer to Figure 3 for the ACF plot for the S&P500 index data.

2.2.2 Heavy Tails

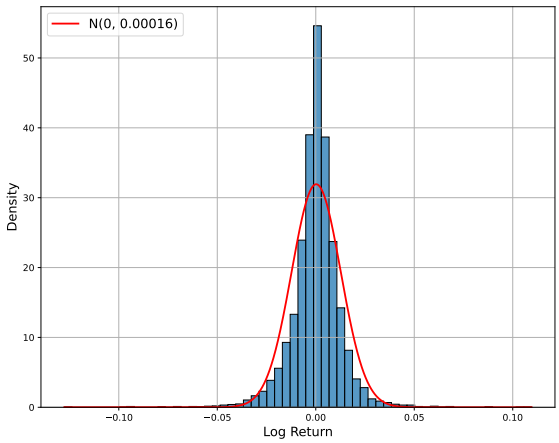
In Figures 4a and 4b, we observe the distribution of asset returns show positive excess kurtosis, i.e. extreme events occur more often than would be expected from a normally distributed random variable. These figures illustrate the "Heavy-tailed" nature of asset price returns as first documented by Mandelbrot [31].

2.2.3 Small and Decreasing Autocorrelation in Squared/Absolute Returns

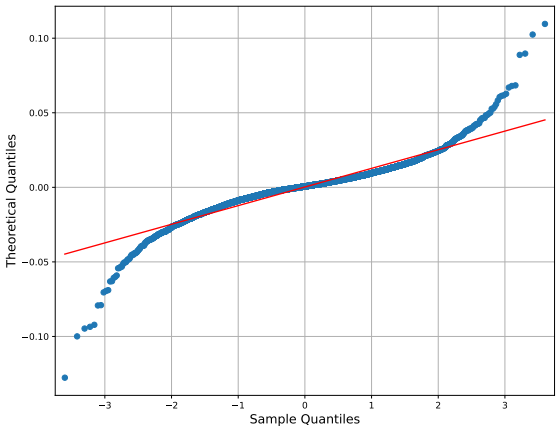
We notice some persistence in the squared and absolute autocorrelations in Figure 5, indicating that price changes in the S&P500 are likely followed by changes of the same magnitude of either sign. This persistence can be seen as a quantitative manifestation of the volatility clustering effect [1].

2.2.4 Leverage Effect

We can see in Figure 6 that large drops in asset price correspond to increases in asset volatility. The bottom graph's rolling volatility rises when the asset price falls. This agrees with the phenomenon dubbed the leverage effect, first studied by Black [32] and expanded upon by numerous others since [33] [34] [35].

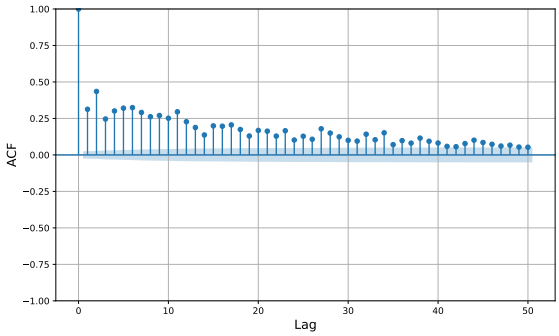


(a) Histogram of S&P500 Returns

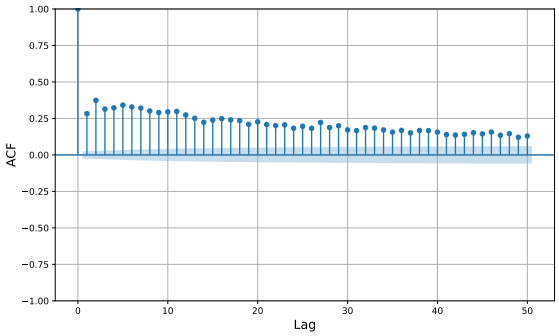


(b) QQ plot for S&P500 Returns

Figure 4: Distribution of S&P500 returns



(a) ACF values for Squared returns



(b) ACF values for Absolute returns

Figure 5: Autocorrelation in squared and absolute returns.

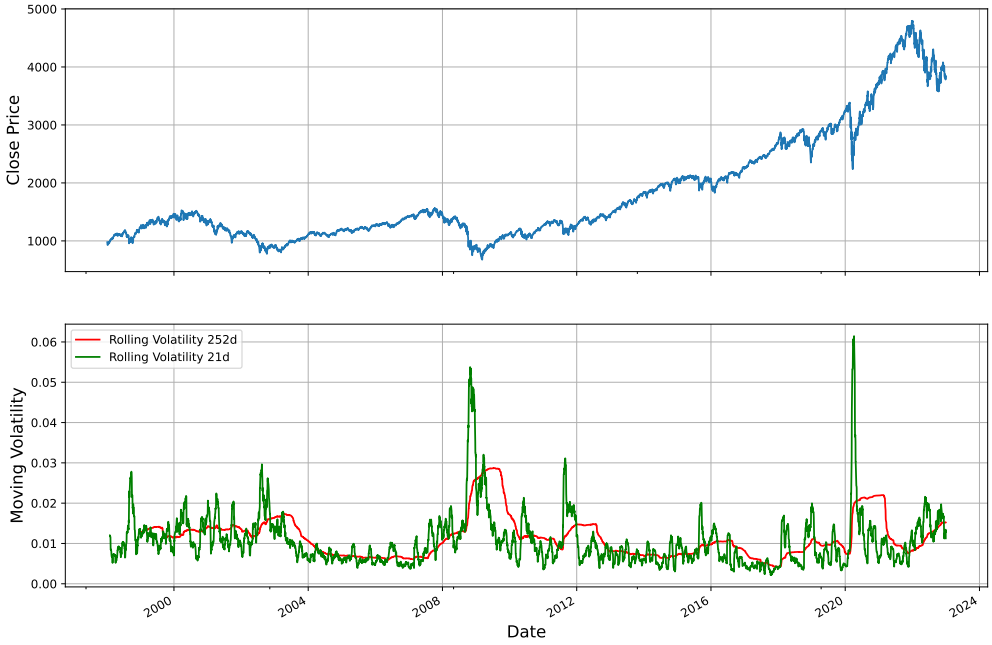


Figure 6: Leverage Effect

2.2.5 Volatility Clustering

We notice from Figure 2 that high volatility events tend to cluster in time. This demonstrates the volatility clustering effect mentioned earlier. Many articles and texts in quantitative research have highlighted the importance of volatility modeling to a range of applications for portfolio management, option pricing, and risk management [36] [37] [38] [39]. In Figure 6, we observe that volatility commonly spikes before gradually reverting to its pre-spike level. For this reason, it can be modeled as a mean reverting process [40].

3 Generative Methods and GANs

GANs are the tools we will use to generate data in this dissertation. As such, a familiarity with the conceptual elements will help. The next section introduces neural networks and GANs, commencing with a brief historical overview of the technology. We will also explain some of the more theoretical details of GAN training. Once we introduce the concepts, we will discuss the common issues faced when training GANs and some further developments that will be used in our models. Finally, we explore implementations of GANs in the context of financial time series data, aligning with our research objectives.

3.1 A Short History

Generative AI has recently burst into the collective conscience due to its novel accessibility, media hype, and the political discourse surrounding recent developments [41] [42] [43]. However, conceptually, generative models have been around for many years. In the early 1900s, Andrey Markov analyzed sequences of dependant random variables [44]. He defined stochastic processes where future outcomes depend on previous outcomes, thus giving us a probabilistic method of generating new data conditioned on previous steps. One could point to the work of Alan Turing as an essential stepping stone. Turing developed the idea that a computer could compute anything a human could compute using logic, describing a precursor to all modern software [45]. He looked at how we could create "intelligent" machines and how those machines could be tested for intelligence [46]. The concept of a neural network, designed as a probabilistic model for how information could be stored in the brain, now marks its eightieth anniversary. The description by McCulloch & Pitts in 1943 [47] and implementation by Rosenblatt in 1958 [48] paved the way for modern neural computing and deep learning. Figure 7 sketches a perceptron, the first neural network model.

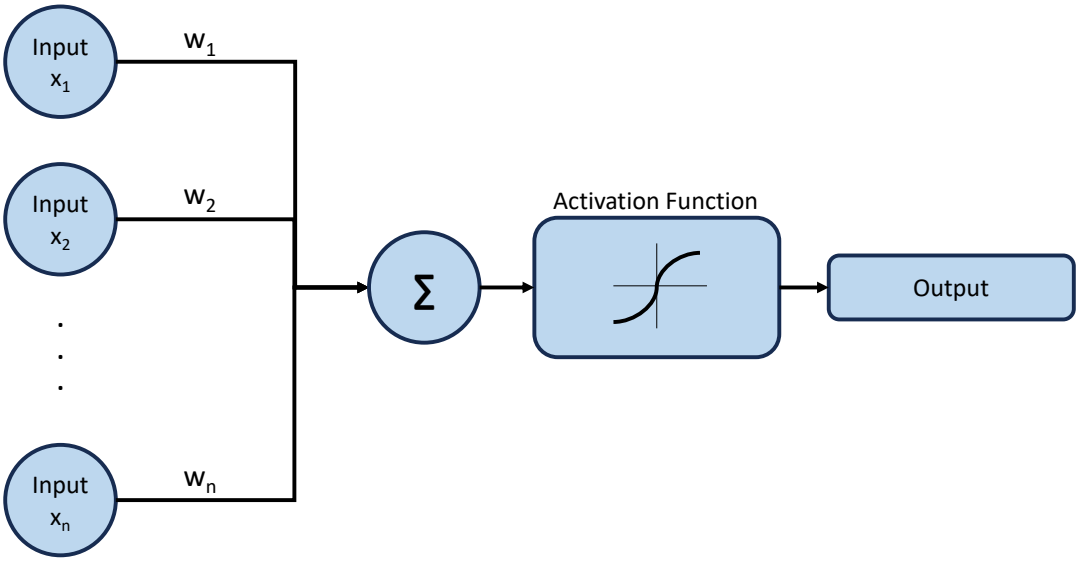


Figure 7: The perceptron, a binary classification algorithm. Inputs on the left are multiplied by their respective weights. A summation is taken and then passed through an activation function (e.g., ReLU, tanh, sigmoid) to receive a numeric output

Since these ideas, technology has come a long way. Years of research and advances in neural computing combined with the democratization of advanced processing machinery make artificial neural networks a widely available, powerful modeling technique [49]. Nowadays, anyone with access to a computer (and some conceptual understanding) can build robust classification systems far beyond the reach of McCulloch and Pitts.

To leverage these advances to generate new data, we must correctly configure our model. Recent years have seen new generative models building on the principles of their predecessors. These powerful probabilistic models include Variational Auto-Encoders [50] and, the primary focus of this dissertation, Generative Adversarial Networks (GANs).

3.2 Generative Adversarial Networks (GANs)

GANs were first proposed in 2014 [21]. In the initial paper, the framework was laid out for what would become a landmark development in the history of generative AI, demonstrating success in fields as varied as audio synthesis [51], drug discovery [52] and high energy physics [53]. The successful applications are all derived from a relatively simple principle, which can be summarized as follows: One neural network's objective, the generator (G), is to produce data that mimics the real data distribution as closely as possible. A separate neural network, the discriminator (D), is tasked with identifying which data came from the generator and which came from the real distribution. Goodfellow uses a cogent analogy, likening the generator to criminal counterfeiters and the discriminator to the police. G attempts

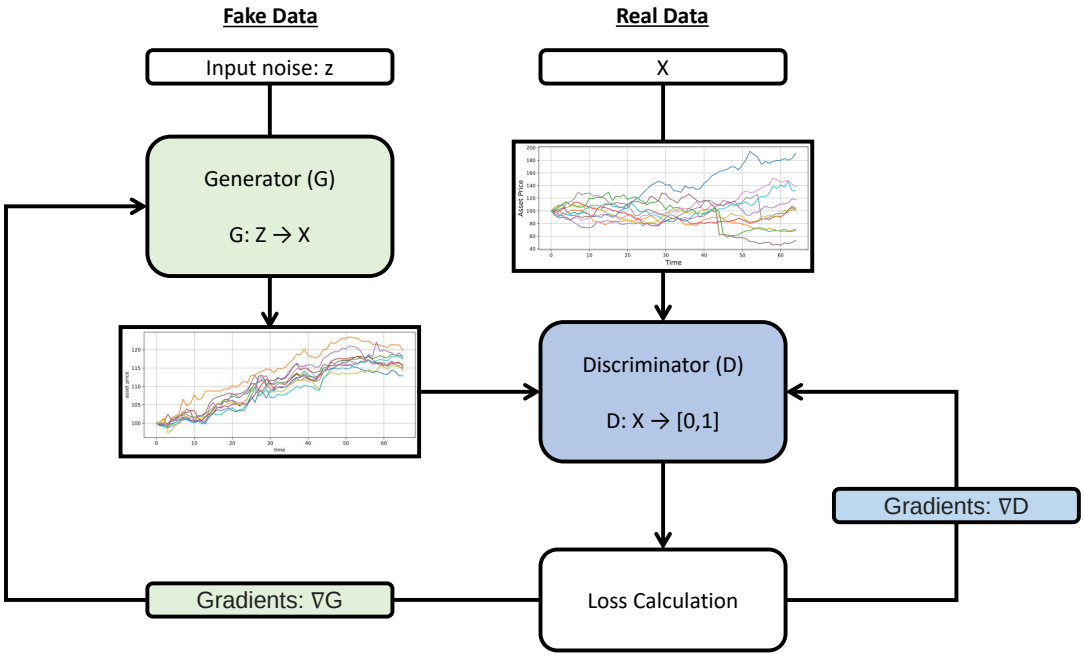


Figure 8: Generative adversarial network layout

to create realistic-looking currency, while D tries to find the forged currency in circulation. Both the generator and discriminator may start with pretty basic skills, but as one improves, the other must catch up, increasing the sophistication of each. The architecture is illustrated in Figure 8.

An exemplary demonstration of the strength of adversarial training comes from StyleGAN [54], which produces lifelike images of human faces based on characteristics it has learned from its training set of facial portraits. The images in Figure 9a are all generated by StyleGAN. These images would be almost indistinguishable from real portraits to a human observer. In mathematical terms, the original paper [21] describes the GAN training process as the generator and the discriminator playing the two-player minimax game in equation (1).

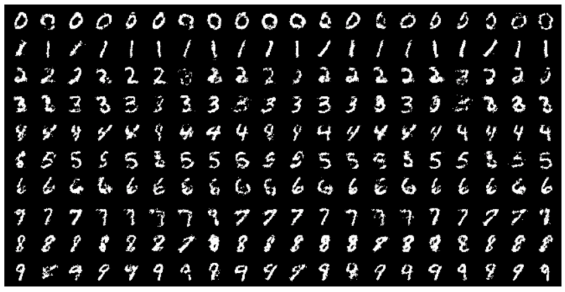
$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim\mathbb{P}_{data}(x)}[\ln D(x)] + \mathbb{E}_{z\sim\mathbb{P}_z(z)}[\ln(1-D(G(z)))] \tag{1}$$

We will look briefly at the mathematical motivation behind this game. Here z represents a vector of input noise from the latent vector space Z, and x is an instance within the data space X. $\mathbb{P}_{data}$ is the distribution of our data and $\mathbb{P}_z$ is the distribution of the noise. The differentiable function G maps the vector z from the latent space into our defined data space.

$$G : Z \to X \tag{2}$$

A second function D is defined to estimate the probability that x comes from the data rather than from G acting on the input noise z. D is trained to maximize the probability of assigning the correct label to samples coming from G or the training data.

(a) Portrait images generated using StyleGAN from Karras et al. [54]



(b) Generated MNIST digits from Mirza [55], each row is conditioned on one label.

Figure 9: Examples of GAN generated image data

$$D : X \to [0, 1] \tag{3}$$

To derive the minimax game in (1), let us look at the optimization goals of G and D. We say that both G and D are optimized using binary cross-entropy. For a real input into D, the correct output is 1, giving us the following binary cross-entropy:

$$\text{BCE}(1, D(x)) = -(1 \cdot \ln(D(x)) + (1 - 1) \cdot \ln(1 - D(x))) = -\ln(D(x)) \tag{4}$$

For a fake input into D, the correct label should be 0, giving the next binary cross-entropy:

$$\text{BCE}(0, D(G(z)) = -(0 \cdot \ln(D(G(z)) + (1 - 0) \cdot \ln(1 - D(G(z))) = -\ln(1 - D(G(z))) \tag{5}$$

Thus, combining the losses for each case gives us the overall optimization objective for the discriminator in the case of a single data point:

$$\min_D \left[-\ln(D(x)) - \ln(1 - D(G(z)))\right] \tag{6}$$

This may equivalently be written as:

$$\max_D \left[\ln(D(x)) + \ln(1 - D(G(z)))\right] \tag{7}$$

Now consider the generator. G tries to fool D into misclassifying fake data as real, corresponding to the following optimization objective:

$$\min_G \left[\ln(1 - D(G(z)))\right] \tag{8}$$

When combined, these two objective functions give us the optimization objective of the GAN for a single data point:

$$\min_G \max_D \left[\ln(D(x)) + \ln(1 - D(G(z)))\right] \tag{9}$$

Extending from a single point into the possible set of points results in the minimax game described by Goodfellow in (1). In practice, the improvement is achieved through the iterative process described in algorithm 1.

Algorithm 1 GAN Training Algorithm

Input: G, D: Functions (neural networks). m: sample size. k: D steps per G step

Output: Weight updates for G and D

1: **for** number of training iterations **do**
2: **for** k steps **do**
3: $z \leftarrow \{z^{(1)}, ..., z^{(m)}; z^{(i)} \sim \mathbb{P}_z(z), i \in [1, ..., m]\}$
4: $x \leftarrow \{x^{(1)}, ..., x^{(m)}; x^{(i)} \sim \mathbb{P}_{data}(x), i \in [1, ..., m]\}$
5: $\nabla_D \leftarrow \frac{1}{m} \sum_{i=1}^{m} \left[\ln D(x^{(i)}) + \ln(1 - D(G(z^{(i)}))) \right]$
6: Update weights in D using ∇_D
7: **end for**
8: $z \leftarrow \{z^{(1)}, ..., z^{(m)}; z^{(i)} \sim \mathbb{P}_z(z)\}$
9: $\nabla_G \leftarrow \frac{1}{m} \sum_{i=1}^{m} \ln(1 - D(G(z^{(i)}))$
10: Update weights in G using ∇_G
11: **end for**

In English, algorithm 1 performs the following actions: In lines 3 and 4, we sample points from our latent space Z and our data space X, respectively. We use these to calculate the gradients for the discriminator, ∇_D in line 5. In line 8, another sample is taken from the latent space, which is then used to calculate the gradients of the generator in line 9.

Theoretically, this training regime corresponds to the minimization of the Jenson-Shannon divergence between the underlying probability distributions of the data ($\mathbb{P}_{data}$) and the distribution of the generator ($\mathbb{P}_g$) [56] [57]. Assuming the distributions are continuous with densities P_{data} and P_g, Jensen-Shannon divergence is defined as follows:

$$JSD(\mathbb{P}_{data}||\mathbb{P}_g) := \frac{1}{2} KL(\mathbb{P}_{data}||\mathbb{P}_{avg}) + \frac{1}{2} KL(\mathbb{P}_g||\mathbb{P}_{avg}) \tag{10}$$

Where $\mathbb{P}_{avg}$ is the average distribution having density $\frac{P_{data} + P_g}{2}$ and where $KL(\mathbb{P}_{data}||\mathbb{P}_g)$ denotes the Kullback-Leibler divergence between $\mathbb{P}_{data}$ and $\mathbb{P}_g$:

$$KL(\mathbb{P}_{data}||\mathbb{P}_g) := \int_X P_{data}(x) \ln \left(\frac{P_{data}(x)}{P_g(x)} \right) dx \tag{11}$$

In plain terms, the Kullback-Leibler divergence is a measure of dissimilarity between probability distributions, and the Jensen-Shannon divergence is a symmetrized and smoothed version of this [58] [59].

3.3 GAN Augmentations

Following the initial proposal in 2014, an extensive array of novel architectures, optimizers, algorithms, and loss functions were suggested for implementation in GANs. In this segment, we will discuss two advances that hold significance for the generative models employed later in this dissertation, offering a rationale for their inclusion in our time-series generation task.

3.3.1 DCGAN

A primary deficiency of densely connected neural networks is their inability to capture the strong correlations between adjacent variables in a time series[60]. Convolutional neural networks solve this problem by convolving the time series with kernels to capture local features. This approach has seen convolutional neural networks applied to a variety of time series classification and forecasting tasks [61] [62] [63]. Convolution-based networks have been shown to compare favorably to recurrent architectures in many sequence-related tasks [64].

DCGAN extends the structure of GANs to incorporate convolutional neural networks, providing a more stable architecture than previous attempts. For this reason, we will start with the DCGAN architecture proposed by Radford [65], adapted to time series and pared back to a minimal depth. Many other GAN architectures accommodating convolutional layers have been proposed since DCGAN [66] [67] [68]; however, these will be outside of the scope of this dissertation.

3.3.2 cGAN

While GANs produce data mimicking their real counterpart, they do not have a way of handling class labels by default. This is a valuable functionality since we often require scenario-specific data rather than data representative of the whole training set. To address this challenge, we may extend the standard GAN into a conditional framework [55]. Conditional GANs (cGANs) have the desirable property that we may feed labels into our model to create scenario-specific data generation under certain user-defined conditions. This is achieved by feeding our condition $y \in Y$ into the generator and discriminator. An illustrative example from the original paper is included here. In Figure 9b, we can see MNIST digit data produced by cGAN, with each row being conditioned on one label corresponding to the digits 0-9. By taking in labels corresponding to the data, the minimax game devised by Goodfellow now becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\ln D(x|y)] + \mathbb{E}_{z\sim p_z(z)}[\ln(1 - D\left(G(z|y)|y\right))] \tag{12}$$

$$G : Z \times Y \to X \tag{13}$$

$$D : X \times Y \to [0,1] \tag{14}$$

3.4 Unique Challenges in GAN training

The potential of GANs is evident from experimental results, particularly in image-related tasks. However, they are notoriously difficult to train [69]. In this section, we list some known challenges when training GAN-based models. Awareness of these issues can guide the user towards more fruitful training techniques.

3.4.1 Relative Complexity

Generative models require a greater capacity to capture complex dependencies. Take the example of facial portraits. A model would capture ideas like "usually two eyes" and "mouth goes below the nose" probabilistically. The summation of all these relationships results in a hugely complex set of inter-dependencies. The number of possible combinations of features is huge, while the number of configurations that constitute a reasonable image is quite small [70]. From this perspective, a generative model must be sophisticated enough, and the training process must be robust enough to capture all

these dependencies. If we swap the spatial dependencies in images for the temporal dependencies in time series, one can see how this is a salient point for our application. One practical remedy to the complexity issue starts by training a GAN on simpler data sets, gradually increasing complexity until the desired level is reached [22].

3.4.2 Optimization

Neural networks are trained through iterative updates of their weights in a direction that decreases a cost function. Since traditional optimization techniques such as gradient descent are designed to find the global minimum of a cost function as opposed to the Nash equilibrium of a minimax game, when applied in GANs, convergence is not guaranteed [71]. In certain cases, the optimization algorithm can produce oscillatory dynamics between the generator and discriminator, which will never converge [72] [73].

3.4.3 Evaluation Criteria

Measuring the success of generative models is difficult. In the case of StyleGAN, producing samples from the model is sufficient as the human observer is a good judge of what is and is not a face. In our chosen subject of financial time series, we must understand the qualitative properties we are looking to reproduce. Even equipped with this, it is not a precise measure. We may need a numerical criterion for our model to judge model success. Examples include log-likelihood, Kullback-Leibler divergence, Jensen-Shannon divergence, or Wasserstein distance. Importantly, success using one criterion does not ensure success using others [56]. We must be careful to select an appropriate evaluation method for our data.

3.4.4 Mode Collapse

Mode Collapse occurs when the generator collapses too broad a region of the latent space into a single output or group of outputs, resulting in the inability of the GAN to exhibit sufficient diversity in its generated data. An example of mode collapse would be a generator trained on MNIST digit data producing only the digit "8" irrespective of the latent vector input. Another example would be a generator trained on time series data producing paths following the same trajectory, as seen in Figure 10. Mode collapse can be described as convergence towards undesirable local minima [68] and has been linked to the problem of catastrophic forgetting in ANNs [74].

3.5 GANs Applied to Asset Return Time Series

The rise in popularity of GANs has seen them applied to many time-series-related tasks, including data imputation, generation, anomaly detection, and forecasting[75]. We now look at two recent implementations pertinent to financial time series and explore some important features from each.

3.5.1 QuantGAN

In QuantGAN [28], Two different neural network architectures are implemented in an adversarial training framework to produce synthetic log return data. The authors prove that when trained in an adversarial framework, neural networks can approximate financial time series in discrete time. The more successful NN architecture for the chosen metrics of Wasserstein distance [76], DY metric [77], and ACF Score

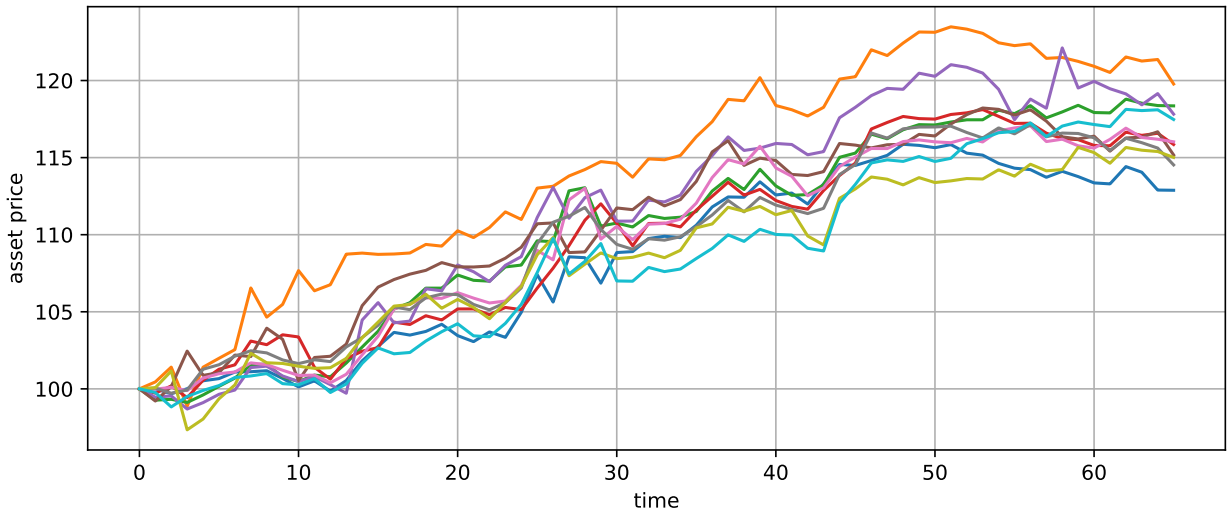


Figure 10: An example of mode collapse in the time series generation task. Outputs follow a similar trajectory irrespective of the latent space input vector

(derived from autocorrelation) is the temporal causal network (TCN) architecture. This architecture had previously demonstrated great aptitude for generating raw audio [51]. The ability of a TCN to map a very large receptive field to a single output (see Figure 11) allows long-range dependencies found in raw audio and financial returns to be modeled by the network. Spectral normalization [78] layers in the network are crucial for training stability. The data preprocessing pipeline is also shared here. The authors employ an inverse Lambert-W transformation [79] to give the generated data a more realistic heavy-tailed distribution. The model outperforms a traditional GARCH(1,1) model in the selected metrics, but the author suggests the need for a unified evaluation criterion that combines distributional metrics and dependence scores.

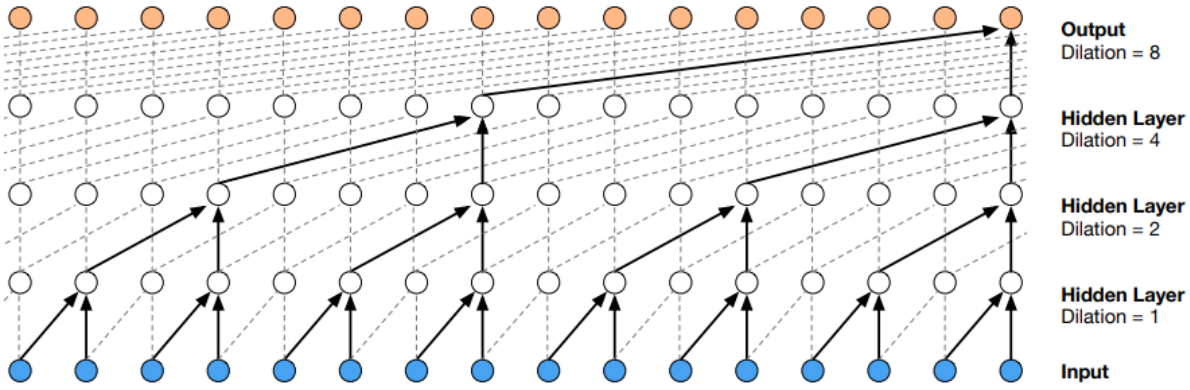


Figure 11: A stack of dilated causal convolutional layers. Figure from Van den Oord et al. [51]

3.5.2 FIN-GAN

FIN-GAN [29] is another commonly cited implementation in recent literature. Here, the author tests various combinations of multi-layer perceptron and convolutional neural networks (CNNs) in the generator to find the architecture that best reproduces the qualitative properties of the financial return data upon which it is trained. Recurrent Neural Network (RNN) based architectures are discounted by the author due to the associated computational cost for long time series.

3.6 Other Financial Applications

Although path simulation is most clearly linked to the focus of this dissertation, there are well-documented further uses for synthetic data in the financial world [80] [81]. These uses may require data taking forms beyond multivariate time series, but the generative methods discussed here still apply. Indeed, synthetic data has been cited as one of the most promising general techniques on the rise in modern deep learning [82]. Here, we briefly summarize some important applications mentioned in the literature [80] [81] to understand other potential uses of GANs in finance.

1. **Data sharing**. Sharing data among financial institutions and within the research community can lead to solutions for common technical challenges these institutions face. Data sharing and privacy restrictions (e.g., GDPR, HIPAA) often prevent this. Synthetic data allows solutions to be inferred from realistic data while keeping within the bounds of data-sharing restrictions.
2. **Testing strategies given limited historical data**. It is often very difficult to study certain behavior regimes due to data scarcity. In financial data sets, market crash situations exemplify this point. Due to the lack of available data, it is useful to have synthetic data for testing strategies and inferences.
3. **Data labeling** In cases where labeled data is required for model training and is not readily available, generative models can be used to produce fully labeled training sets.
4. **Tackling class imbalance**. For use cases such as fraud detection, the datasets are usually heavily skewed, leading to the failure of traditional ML techniques. By applying appropriate data imputation methods, we can use synthetic data to overcome the issue of class imbalance.
5. **Training advanced ML models**. Utilizing advanced machine learning, such as deep learning, often involves cloud-based services that require substantial computing resources and extensive training data. Financial institutions, bound by data security, would be rightly hesitant to upload sensitive training data to external platforms. Synthetic data can be employed to train models externally. These models, once trained, can be deployed internally and used alongside internal, sensitive data. Moreover, training on synthesized data helps safeguard against "membership inference attacks", whereby sensitive training data could be exposed through model parameters.
6. **Causal Modelling** Synthetic data can help explore causal structures with adjusted distributions. This equates to asking the question *"What if?"*. This differs from the earlier point regarding limited historical data since the scenario need not necessarily have occurred before to extrapolate to it via our generative model. Think of regimes of unprecedented interest rates, for example. It should be stated that a degree of model risk is introduced in these situations.

4 Replicating Data from Traditional Financial Models

In the following section, we will examine some notable financial models and the motivation for their development. We will use common discretizations of these models to provide training data for our GAN. The GAN will be trained using this data, and the outputs will be examined statistically compared to their training sets. For more information on the derivation of the discretizations in this section, see Glasserman [83]. Hilpisch [84] provides practical implementations of these in Python, which have been adapted for use here.

Demonstrating the ability of our GAN model to learn the distributions found in these time series lends credence to their application to real-world financial data. Here, we seek to demonstrate that GANs accurately replicate the data distributions produced by their respective "ground truth" SDEs. This will be done by comparing descriptive statistics from each and applying a 2-sample Kolmogorov-Smirnov test [85]. Samples consist of 1000 instances.

The Architecture of the GAN models used in this section can be found in appendix Figures 27a and 27b. Data import and manipulation are performed in Python using functions from the Pandas, Numpy and Scikit-Learn libraries [86] [87] [88]. Tensorflow and Keras are used for the construction of the GAN models [89] [90].

4.1 Black-Scholes Model

Indisputably the most famous model in financial mathematics, the Black-Scholes model incorporates the idea of geometric Brownian motion to capture the stochastic nature of asset prices [3]. To understand why this makes sense, we can look to the efficient market hypothesis espoused by Malkiel [91]. This theory simply states that the current asset price reflects all past information on the asset. Thus, when considering what the price may be tomorrow, we gain no advantage in looking at the asset price at any other time but today. The Black-Scholes model (along with many other models) reflects this by modeling the asset price S as a Markovian process using the following stochastic differential equation:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{15}$$

Here, μ is the drift, σ is the volatility, and W_t is a Wiener process (1-dimensional Brownian motion here). By applying Itô's Lemma [92], we get the solution of the SDE as

$$S_t = S_0 \exp\left[\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma\sqrt{t}Z\right] \tag{16}$$

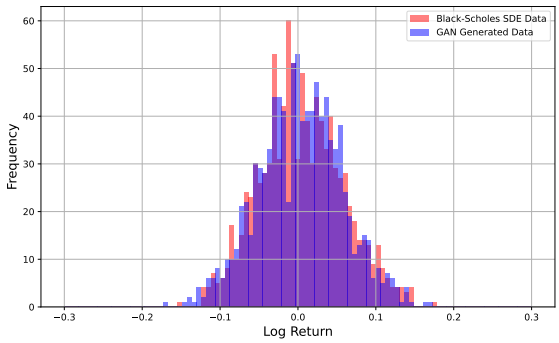
Where Z is a standard normal random variable. While this model makes several assumptions about the behavior of the underlying (e.g., constant volatility, constant interest), it remains fundamental to understanding modern asset price models. It serves as a familiar benchmark with which we can test the conditional generation capabilities of our GAN.

4.1.1 SDE Parameters

```
S0 = 100 #initial asset price
r = 0.1 #constant short rate
sigma = 0.3 #constant volatility
T = 2.0 #time in years
I = 10000 #no. of random draws
M = 64 # no. of time steps
dt = T / M
S = np.zeros((M+1, I))

S[0] = S0
for t in range(1, M+1):
    S[t] = S[t-1] * np.exp((r - 0.5*sigma**2) * dt
                           + sigma * np.sqrt(dt) * npr.standard_normal(I))
```

Listing 1: Black-Scholes parameterization and implementation in python

4.1.2 Results



(a) Log returns generated by GAN and Black-Scholes SDE

(b) Final asset price generated by GAN and Black-Scholes SDE

Figure 12: Black-Scholes: Comparison of distribution between GAN and SDE data

	SDE	**GAN**
Mean	0.0044	0.0028
Variance	0.0029	0.0029
Minimum	−0.1504	−0.1710
Maximum	0.1746	0.1679
Skewness	0.1525	−0.0135
Kurtosis	−0.2311	−0.0709
2 Sample KS Test		
Statistic	0.0320	
p-value	0.6855	

Table 1: Descriptive stats of Black-Scholes log returns. Figure 12a

	SDE	**GAN**
Mean	120.3620	121.2236
Variance	2958.2168	3061.8562
Minimum	32.0749	15.6718
Maximum	414.8591	347.5268
Skewness	1.3448	1.0513
Kurtosis	2.9388	1.2857
2 Sample KS Test		
Statistic	0.0340	
p-value	0.6102	

Table 2: Descriptive stats of Black-Scholes final asset price. Figure 12b

We conclude that the distributions are suitably alike. The 2-sample Kolmogorov-Smirnov test provides p-values exceeding the 0.05 threshold, and we may not reject the null hypothesis that these samples have been drawn from the same distribution. See appendix Figure 24d for example paths.

4.2 Cox-Ingersoll-Ross (CIR) Model

Some processes in finance (e.g., interest rates, volatility) exhibit the property of mean reversion [40]. Although the returns have some stochastic element, when a value high above the long-term mean is attained, there will be an overall downward trend back towards this long-term mean, which may also be a function of time. The Cox-Ingersoll-Ross (CIR) model [17] captures this behavior and is documented as a model for interest rates in many popular texts [93] [94] [39]. It is defined by the following SDE:

$$dx_t = \kappa(\theta - x_t)dt + \sigma\sqrt{x_t}dZ_t \tag{17}$$

The new terms κ and θ are interpreted as the mean-reversion factor and the long-term mean, respectively. Increasing κ will result in faster mean-reversion. Using our GAN to replicate these properties motivates its use for interest rate and volatility processes.

Different discretizations are discussed in the literature for the CIR model [95]. Although the Milstein and Exact schemes are numerically more accurate [96], for the purposes of simulation within this dissertation, the Euler approximation (AKA full truncation [97]) scheme suffices. To implement this scheme, we can discretize the CIR SDE by defining $s = t - \Delta t$ and $x^+ = \max(x, 0)$ to give us the following:

$$\tilde{x}_t = \tilde{x}_s + \kappa(\theta - \tilde{x}_s^+)\Delta t + \sigma\sqrt{\tilde{x}_s^+}\sqrt{\Delta t}Z_t \tag{18}$$

$$x_t = \tilde{x}_t^+ \tag{19}$$

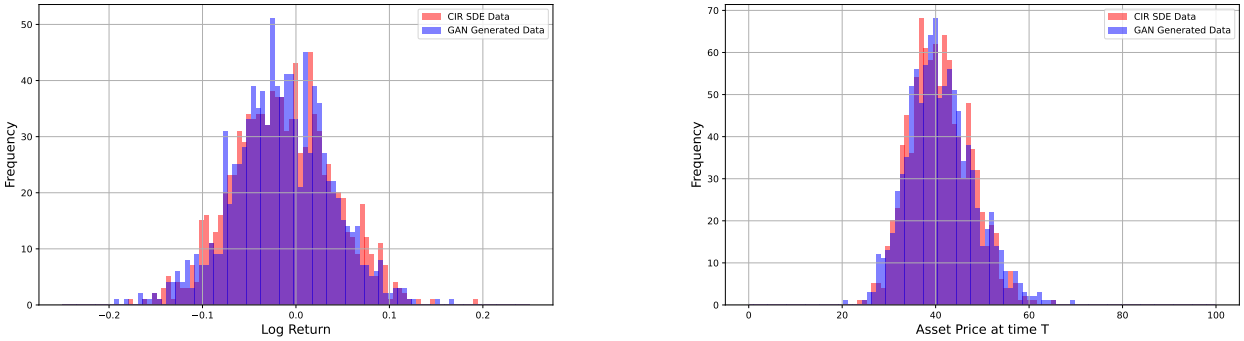
4.2.1 SDE Parameters

```
x0 = 100 #initial x value
kappa = 2.0 #mean-reversion factor (higher => faster reversion to mean)
theta = 40 #long term mean
sigma = 2
T = 2.0
I = 10000
M = 64
dt = T/M

def sqrt_diff():
    xh = np.zeros((M+1,I))
    x1 = np.zeros_like(xh)
    xh[0] = x1[0] = x0
    for t in range(1,M+1):
        xh[t] = (xh[t-1]
                + kappa * (theta - np.maximum(xh[t-1], 0)) * dt
                + sigma * np.sqrt(np.maximum(xh[t-1],0)) * np.sqrt(dt)
                * npr.standard_normal(I))
    x1 = np.maximum(xh,0)
```

```
20      return x1
```

Listing 2: CIR parameterization and implementation in python

4.2.2 Results



(a) Log returns generated by GAN and CIR SDE

(b) Final asset price generated by GAN and SDE

Figure 13: Cox-Ingersoll-Ross: Comparison of distribution between GAN and SDE data.

	SDE	**GAN**
Mean	−0.0130	−0.0147
Variance	0.0028	0.0026
Minimum	−0.1776	−0.1895
Maximum	0.1901	0.1690
Skewness	0.0745	−0.0330
Kurtosis	−0.1067	0.2678
2 Sample KS Test		
Statistic	0.0320	
p-value	0.6855	

Table 3: Descriptive stats of CIR log returns. Figure 13a

	SDE	**GAN**
Mean	41.0014	41.1712
Variance	41.2120	49.1602
Minimum	23.6347	20.7205
Maximum	64.9497	69.3686
Skewness	0.3358	0.4674
Kurtosis	−0.1350	0.2910
2 Sample KS Test		
Statistic	0.0290	
p-value	0.7947	

Table 4: Descriptive stats of CIR final asset price. Figure 13b

We conclude that the distributions are suitably alike. The 2-sample Kolmogorov-Smirnov test provides p-values exceeding the 0.05 threshold, and we may not reject the null hypothesis that these samples have been drawn from the same distribution. See appendix Figure 25d for example paths.

4.3 Merton Jump Diffusion Model

One important feature observed in financial markets is their tendency to exhibit unexpected drops [98] [99] [100]. A normally distributed return model does not account for these essentially discontinuous drops in price. Several explanations have been proposed for this phenomenon. Volatility of returns could be stochastic [16] [18] [101], returns could be drawn from an alternative distribution such as Pareto-Levy [102] [103] or, the assumption which forms the basis of the following model, that asset prices can jump in value. Merton's jump-diffusion model adds a Poisson process to the model to simulate these jump movements [15]. Using our GAN model to recreate this property demonstrates its' ability to recreate these simulated market shocks. The SDE for the Merton Jump Diffusion model is:

$$dS_t = (r - r_j)S_t dt + \sigma S_t dZ_t + J_t S_t dN_t \tag{20}$$

r_j is a drift correction term that is introduced to maintain risk neutrality:

$$r_j = \lambda \left(e^{\mu_j + \delta^2/2} - 1 \right)$$

J_t is a jump at t with the distribution

$$\ln(1 + J_t) \sim N \left(\ln(1 + \mu_t) - \frac{\delta^2}{2}, \delta^2 \right) \tag{21}$$

N_t is a Poisson process with intensity λ. This means that for a timestep dt, the probability of a jump occurring is λdt. Other terms retain their meaning from previous sections. An Euler discretization of this model with z_t^n standard normally distributed and y_t Poisson distributed with intensity λ can be written as:

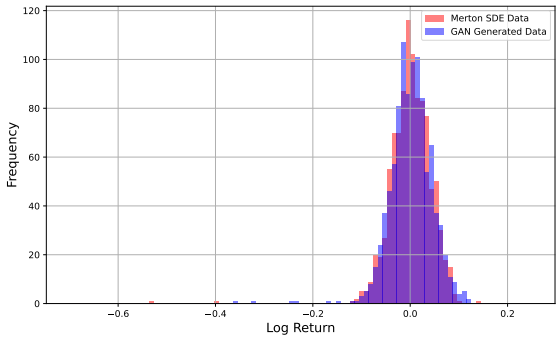
$$S_t = S_{t-\Delta t} \left(e^{\left(r - r_j - \sigma^2/2\right)\Delta t + \sigma\sqrt{\Delta t} z_t^1} + \left(e^{\mu_j + \delta z_t^2} - 1 \right) y_t \right) \tag{22}$$

4.3.1 SDE Parameters

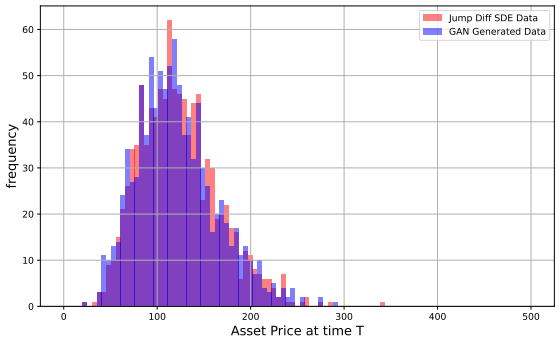
```
S0 = 100
r = 0.2 #constant riskless short rate
sigma = 0.3 #constant vol for S
lamb = 0.2 # poisson process intensity
mu = -0.4 # drift correction mean
delta = 0.1 # drift correction sd
T = 1.0
M = 64
I = 10000
dt = T/M
rj = lamb * (np.exp(mu + 0.5 * delta ** 2) - 1)
S = np.zeros((M+1,I))
S[0] = S0

np.random.seed(42)

sn1 = npr.standard_normal((M+1,I))
sn2 = npr.standard_normal((M+1,I))
poi = npr.poisson(lamb*dt, (M+1,I))

for t in range(1, M+1, 1):
    S[t] = S[t-1] * (np.exp((r - rj - 0.5 * sigma ** 2) * dt
                          + sigma * np.sqrt(dt) * sn1[t])
                          + (np.exp(mu + delta * sn2[t]) - 1)
                          * poi[t])
    S[t] = np.maximum(S[t],0)
```

Listing 3: Jump diffusion parameterization and implementation in python

4.3.2 Results



(a) Log returns generated by GAN and Jump Diff SDE.



(b) Final asset price generated by GAN and Jump-Diff SDE.

Figure 14: Merton Jump-Diffusion: Comparison of distribution between GAN and SDE data

	SDE	GAN
Mean	0.0011	0.0019
Variance	0.0019	0.0019
Minimum	−0.5317	−0.3635
Maximum	0.1385	0.1164
Skewness	−2.6269	−1.2730
Kurtosis	28.3913	9.0566
2 Sample KS Test		
Statistic	0.0230	
p-value	0.9542	

Table 5: Descriptive stats of Jump diffusion log returns. Figure 14a

	SDE	GAN
Mean	123.1812	121.1181
Variance	1818.1957	1820.4042
Minimum	24.1039	24.8669
Maximum	339.7912	289.4449
Skewness	0.6738	0.6316
Kurtosis	0.7448	0.3611
2 Sample KS Test		
Statistic	0.0360	
p-value	0.5363	

Table 6: Descriptive stats of Jump diffusion final asset price. Figure 14b

We conclude that the distributions are suitably alike. The 2-sample Kolmogorov-Smirnov test provides p-values exceeding the 0.05 threshold, and we may not reject the null hypothesis that these samples have been drawn from the same distribution. See appendix Figure 26d for example paths.

4.4 Notes on the GAN Models Used

The GAN models used to replicate SDE processes were intentionally kept shallow for two reasons. Firstly, we have seen that the training set data for each of these has been generated in a manner satisfying the Markov property, meaning no long-term dependencies requiring deeper networks to model. Secondly, training time can increase dramatically for much deeper neural networks. Refer to the appendix for model architecture diagrams. Here, we will examine some model parameters and their observed effects on output variety and quality.

1. **Noise dimension**. Varying the size of the noise vector had minimal effect on the quality of the model outputs, for the most part. The exception appears in cases where the noise dimension is too small (< 20) and outputs fail to exhibit sufficient diversity. Generally, a noise dimension of 50-100 produced satisfactory results.
2. **Number of Filters**. In G, choosing a number too low for the amount of filters restricts the diversity of outputs. Initial tests with between 32-128 filters did not produce satisfactory results. 256 and 512 filters allowed enough diversity in model output, although the increase in filters greatly increased training time. In D, the number of filters did not have the same restriction, and in fact, choosing a larger number induced unstable training and non-convergence in the GAN. For this task, 32 appears to be a sufficient amount of filters.
3. **Batch Normalization**. Batch Normalization in both G and D does improve the stability of training and the quality of outputs at the expense of additional training time.
4. **Kernel Size**. Experimenting with different kernel sizes in G and D appeared inconsequential for the normally distributed Black-Scholes returns. With CIR data, mean-reversion appeared slightly more realistic when using kernel sizes of 5 rather than 3.
5. **Number of Convolutional Layers**. We attempt to limit the number of layers as much as possible. In this task, adding more convolutional layers in G did not produce better results and decreased training stability. Additionally, the increase in training time for additional layers is significant. A single layer with many filters appeared preferable to multiple layers with fewer filters. In D, a second layer produced slightly better results in terms of diversity from the GAN outputs.
6. **Learning Rate**. All models were trained using Adam [104]. The default learning rate for Adam caused some oscillatory behavior in the training losses. More stable training was achieved using a lower learning rate ($\sim 2e^{-5}$).

5 Extension to Conditional Generation

In this section, we take the GANs used in the previous section and extend them to a conditional framework. We first generate data using our SDEs, giving a range of values for the parameter upon which our cGAN will be conditioned. Once the cGAN has been trained, we investigate whether our cGAN has inferred the appropriate effect of each condition by examining its output.

5.1 Black-Scholes cGAN: Conditioned on volatility

For the training set, volatilities are $[0.001, 0.01, 0.1, 1]$. Here, we test the cGANs ability to infer an appropriate distribution between the values provided in the training set. For example, the cGAN has not been trained on any data having a volatility of 0.7. We investigate whether it can produce reasonable data for this volatility.

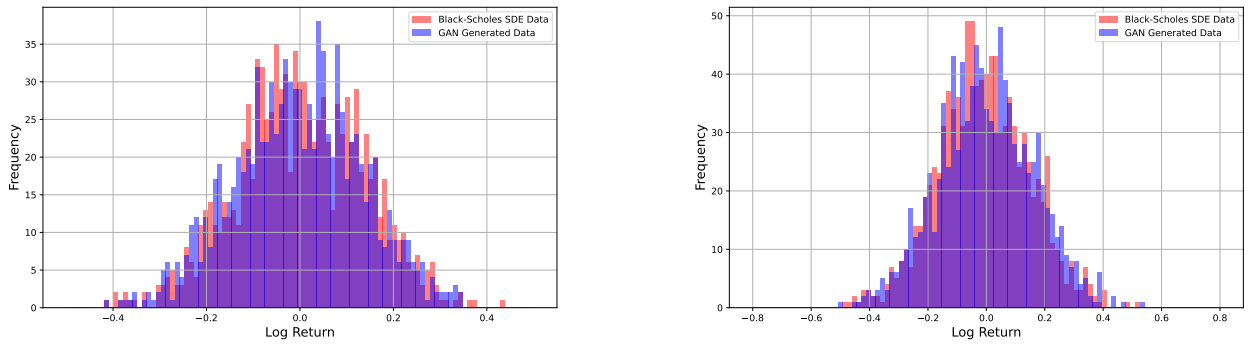
5.1.1 SDE Parameters

```
S0 = 100 #initial asset price
r = 0.2 #constant short rate
T = 2.0 #time in years
```

```
I = 2500 #no. of random draws
M = 64 # no. time steps
dt = T / M

sigmas = [0.001,0.01,0.1,1] #creating different sigma values
Sv = np.zeros((len(sigmas), M+1, I))
Sv[:, 0, :] = S0

for i, sigma in enumerate(sigmas):
    for t in range(1, M+1):
        S_last = Sv[i, t-1]
        S_new = S_last * np.exp((r - 0.5*sigma**2) * dt
                                + sigma * np.sqrt(dt) * npr.standard_normal(I))
        Sv[i, t] = S_new
```

Listing 4: Black Scholes parameterization with varying volatilities

5.1.2 Results



(a) $\sigma = 0.7$: Log returns generated by cGAN and Black-Scholes SDE.

(b) $\sigma = 0.9$: Log returns generated by cGAN and Black-Scholes SDE.

Figure 15: Testing cGAN inferential capabilities: Comparison of log-return distribution between cGAN and SDE data

Based on the p-values obtained, we may not reject the null hypothesis that the samples are drawn from the same distribution for each of the volatilities above. The cGAN model can provide reasonable log return values for these unseen values of σ. We are particularly interested in the variance statistic from these distributions, and increasing σ in the cGAN *does* indeed raise the variance of the log returns, aligning with our expectations.

In contrast to the results above, the cGAN model cannot provide reasonable values in extreme cases where σ is close to 0 or far beyond the extremities of the training set. This can be seen in Figure 16. We can also see evidence of mode collapse when the generator is provided with sigma values far beyond the range within the training set. See appendix Figure 29d for an example of mode collapse. In conclusion, the cGAN can produce reasonable outputs for certain ranges of σ, even previously unseen values. However, it struggles in the edge cases mentioned above.

	SDE	cGAN
Mean	0.0024	−0.0036
Variance	0.0183	0.0182
Minimum	−0.4128	−0.4189
Maximum	0.4389	0.3480
Skewness	−0.0703	−0.0607
Kurtosis	−0.1510	−0.2672
2 Sample KS Test		
Statistic	0.0400	
p-value	0.4006	

Table 7: $\sigma = 0.7$. Figure 17a

	SDE	cGAN
Mean	−0.0106	−0.0026
Variance	0.0246	0.0265
Minimum	−0.4770	−0.5060
Maximum	0.5184	0.5390
Skewness	0.0278	0.0043
Kurtosis	0.0509	−0.1503
2 Sample KS Test		
Statistic	0.0520	
p-value	0.1339	

Table 8: $\sigma = 0.9$. Figure 17b

Table 9: Descriptive stats of log returns for 2 values of σ previously unseen to the cGAN model. Figures 17a and 17b

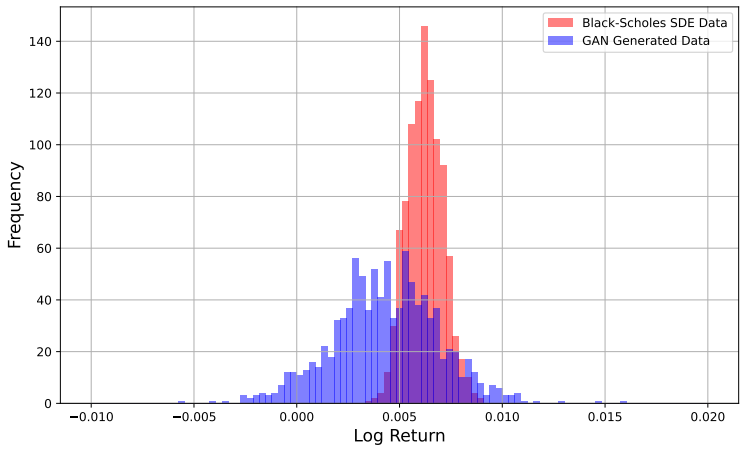
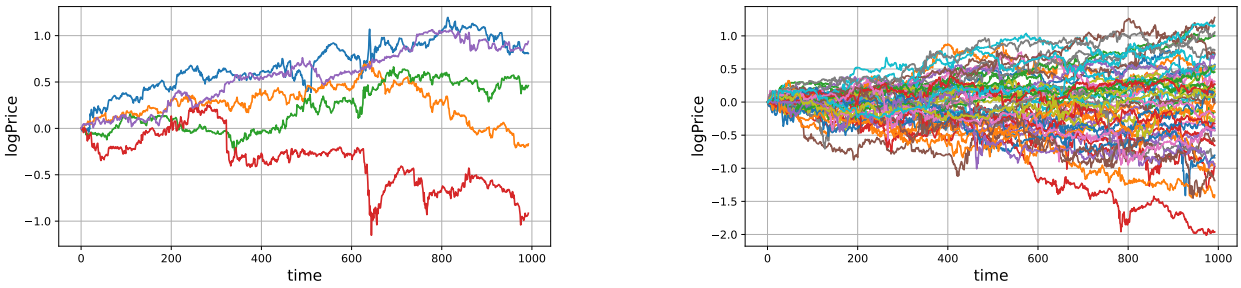


Figure 16: $\sigma = 0.005$: Log returns generated by cGAN and Black-Scholes SDE. GAN is unable to produce a realistic distribution for low values of σ

6 Application to Real Financial Data

Accurately modeling the behavior of asset prices is the ultimate goal of this dissertation. This section will train our GAN model using real asset return data. We will use the GAN to generate asset return series and compare the distributional properties of these series against those found in real data. We will then conduct a qualitative examination of the GAN-generated asset return series, replicating the stylized facts described in Section 2. The time series in this section are of length 992, representing a suitable period over which we can examine their qualitative properties. The Architecture of the GAN models used in this section can be found in appendix Figures 28a and 28b. All training data for this GAN has been extracted from *Yahoo! Finance* using the yfinance module. For detailed training data extraction and preprocessing steps, refer to the appendix.



(a) 5 log paths generated by our GAN

(b) 50 log paths generated by our GAN

Figure 17: Example log paths generated by our shallow convolutional GAN model, trained on real market data

6.1 Results

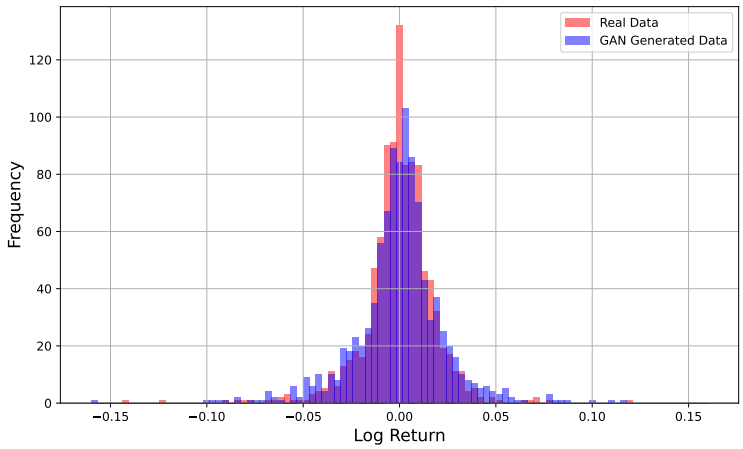


Figure 18: Distributions of real log return data and GAN generated log return data

	Real Data	**GAN**
Mean	0.0000	−0.0001
Variance	0.0005	0.0006
Minimum	−0.1949	−0.1804
Maximum	0.2877	0.1150
Skewness	1.0703	−0.7318
Kurtosis	41.7044	7.5374
2 Sample KS Test		
Statistic	0.051	
p-value	0.1484	

Table 10: Real Data vs. GAN (Figure 18)

Based on the p-value obtained, we may not reject the null hypothesis that the samples are drawn from the same distribution. The GAN has demonstrated the capability to learn the distributional properties of real asset return data in this example. We note the positive excess kurtosis, a distinguishing feature of actual return data, emerging in the GAN-generated distribution. The kurtosis is not of the same magnitude in each of the data samples. A possible solution to this would be to include incorporating an inverse Lambert-W transformation [79] as a data preprocessing step, as suggested by the QuantGAN authors [28].

6.2 Reproducing Stylized Facts Using GAN Data

In this section, we use our GAN trained on real data to produce an asset price return series. We then conduct a qualitative examination of the statistical properties of this series, with reference to the stylized facts discussed in section 2.

6.2.1 Absence of Autocorrelation in Returns

We observe in Figure 19 that the autocorrelations found in the GAN-generated return data are insignificant. The GAN has retained the property of linear unpredictability in the returns.

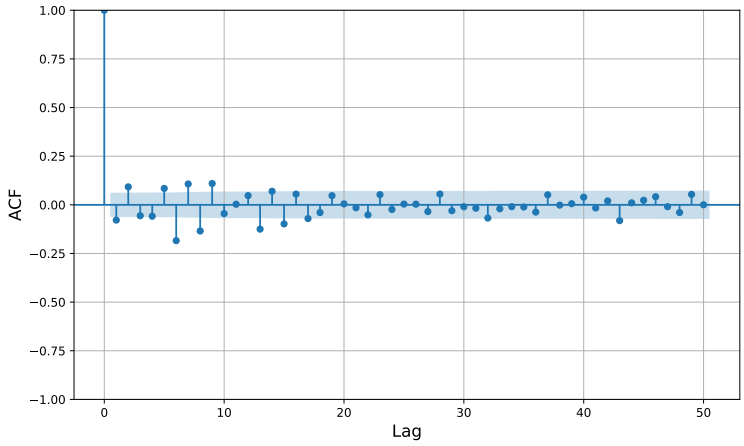


Figure 19: Negligible autocorrelation observed in returns

6.2.2 Heavy Tails

In Figures 20a and 20b we see the replication of the "Heavy-tailed" distribution of returns. The GAN has learned the positive excess kurtosis in the real data distribution.

6.2.3 Small and Decreasing Autocorrelation in Squared/Absolute Returns

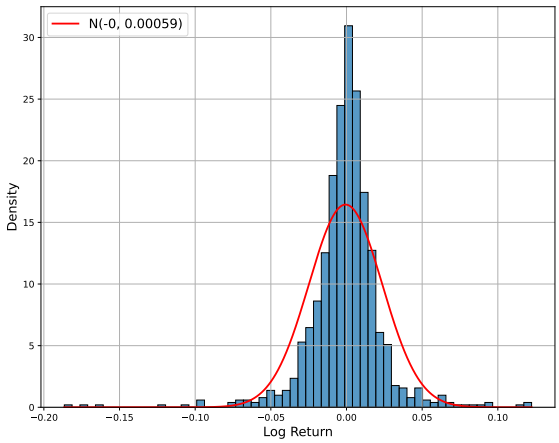
We notice some persistence in the squared and absolute autocorrelations in Figure 21. The GAN has learned and reproduced the volatility clustering effect in the generated time series. However, this effect does not persist in the higher lag values, as we have previously seen in the actual data. This is a quantitative manifestation of volatility clusters having a shorter duration in our GAN-generated data.

6.2.4 Leverage Effect

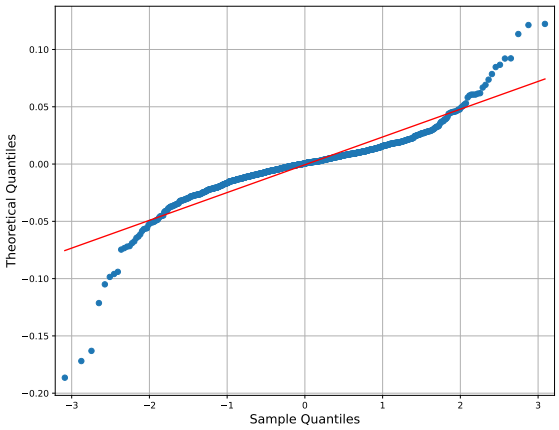
We can see in Figure 22 that significant drops in asset price correspond to increases in asset volatility, as we previously observed in the actual data. The GAN has learned and reproduced the leverage effect.

6.2.5 Volatility Clustering

We observe definite regions of higher volatility in Figure 23. Combined with the plots in Figure 21, we can conclude that the GAN has replicated the volatility clustering effect. It is important to note that

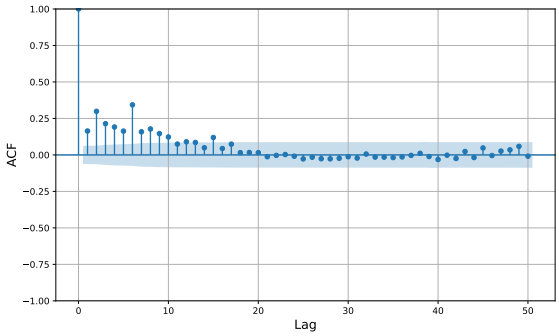


(a) Histogram of GAN generated log returns

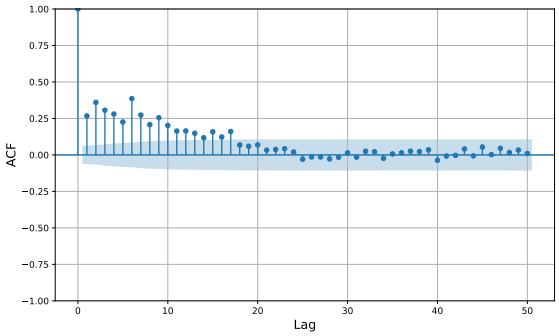


(b) QQ plot for GAN generated log returns

Figure 20: Distribution of GAN generated log returns, replicating the heavy-tails found in empirical data.



(a) ACF values for squared returns in GAN data



(b) ACF values for absolute returns in GAN data

Figure 21: Autocorrelation in squared and absolute returns in GAN generated data

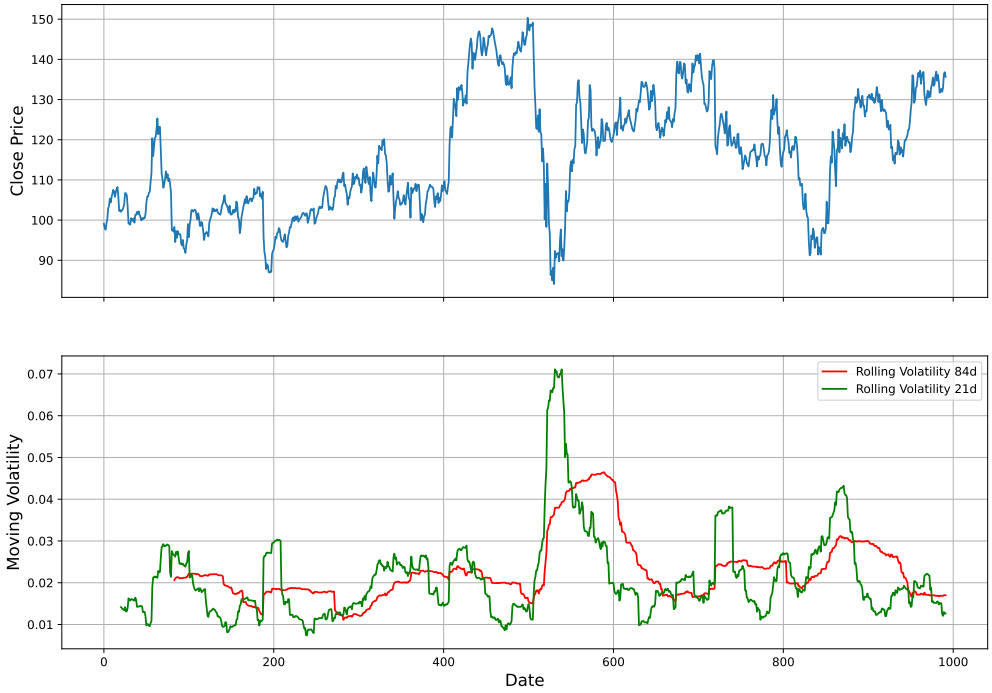


Figure 22: Leverage Effect in GAN generated data

the duration of persistence in volatility observed in the GAN-generated data does not precisely match that of the actual data.

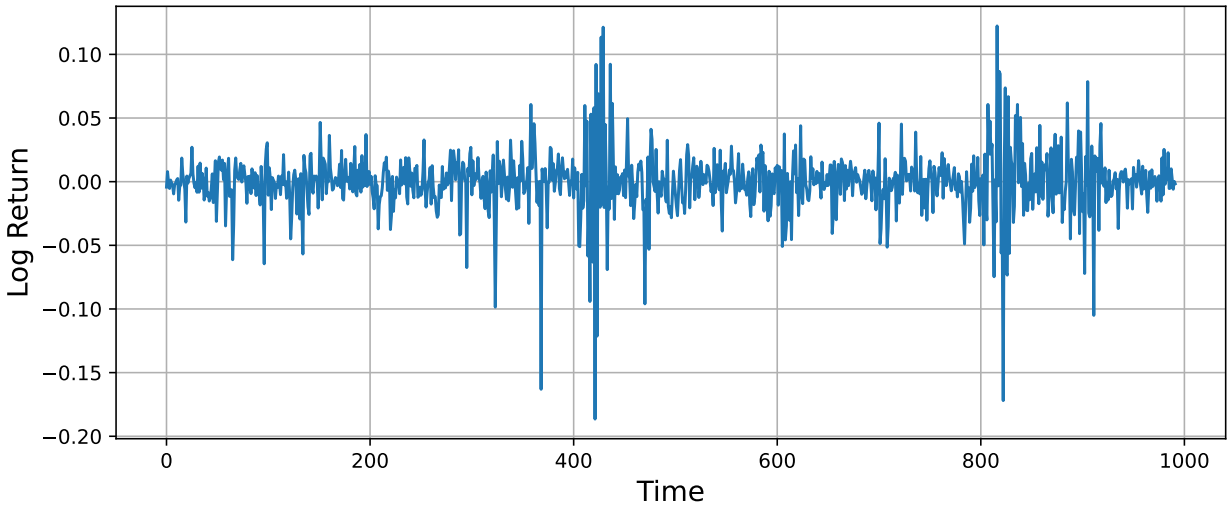


Figure 23: Log returns of the GAN generated data

7 Conclusion

The behavior of asset returns has several complex characteristics, making them challenging to model accurately. Traditional financial models use different techniques to capture various aspects of asset return behavior. GANs are a recently developed technique from the field of deep learning which have shown promise in time-series modeling applications. In this dissertation, we have presented a GAN framework that can reproduce the distributional properties of both traditional financial model data and empirical financial data. Our model's generator and discriminator architecture consists of far fewer layers than comparable models in recent literature. We have also shown that this GAN design can be extended to a conditional framework, using the volatility parameter in the Black-Scholes model as an example.

7.1 Further Work

GAN research is a rapidly developing and expanding field, with new implementations being devised constantly. Exploring new architectural designs, cost functions, and methodologies for generating time series data presents a vast landscape of potential research opportunities, expanding upon the content addressed in this dissertation. One such approach for conditional time series generation in recent works includes the concept of *path signatures* [26] [105] [106]. Exploring some of these new ideas is an exciting proposition.

Given more time, further tuning could also be done with the presented models, training data sets, and their parameters. Further experimentation with various conditions using the cGAN framework could also be conducted. Extending the real-data model to a conditional framework by engineering market features (e.g., periods of recession) may also provide useful results.

There stands another open question regarding the evaluation of generated time series. Here, we have advocated for qualitative examination of their properties; however, echoing the sentiment of the QuantGAN [28] authors, more robust, numerical evaluation criteria combining distributional properties with levels of adherence to the stylized facts are required.

References

[1] Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quant. Finance*, 1(2):223, March 2001.

[2] Anirban Chakraborti, Ioane Muni Toke, Marco Patriarca, and Frédéric Abergel. Econophysics review: I. empirical facts. *Quant. Finance*, 11(7):991–1012, 2011.

[3] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[4] Mark Rubinstein. Nonparametric tests of alternative option pricing models using all reported trades and quotes on the 30 most active CBOE option classes from august 23, 1976 through august 31, 1978. *J. Finance*, 40(2):455–480, June 1985.

[5] Robert Almgren. Financial derivatives and partial differential equations. *Am. Math. Mon.*, 109(1):1–12, January 2002.

[6] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

[7] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *J. Econom.*, 31(3):307–327, April 1986.

[8] Robert Engle. GARCH 101: The use of ARCH/GARCH models in applied econometrics. *J. Econ. Perspect.*, 15(4):157–168, December 2001.

[9] Kris Boudt, Alexios Galanos, Scott Payseur, and Eric Zivot. Chapter 7 - multivariate GARCH models for large-scale applications: A survey. In Hrishikesh D Vinod and C R Rao, editors, *Handbook of Statistics*, volume 41, pages 193–242. Elsevier, January 2019.

[10] Hans Malmsten and Timo Teräsvirta. Stylized facts of financial time series and three popular models of volatility. *Eur. J. Pure Appl. Math.*, 3(3):443–477, May 2010.

[11] Eric Zivot. Practical issues in the analysis of univariate GARCH models. In Thomas Mikosch, Jens-Peter Kreiß, Richard A Davis, and Torben Gustav Andersen, editors, *Handbook of Financial Time Series*, pages 113–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[12] R G Palmer, W Brian Arthur, John H Holland, Blake LeBaron, and Paul Tayler. Artificial economic life: a simple model of a stockmarket. *Physica D*, 75(1):264–274, August 1994.

[13] Thomas Lux and Michele Marchesi. Scaling and criticality in a stochastic multi-agent model of a financial market. *Nature*, 397(6719):498–500, February 1999.

[14] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl_3):7280–7287, 2002.

[15] Robert C Merton. Option pricing when underlying stock returns are discontinuous. *Journal of financial economics*, 3(1-2):125–144, 1976.

[16] Steven L Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2):327–343, 1993.

[17] J C Cox, J E Ingersoll, Jr, and S A Ross. A theory of the term structure of interest rates. *Theory of valuation*, 2005.

[18] Patrick S Hagan, Deep Kumar, Andrew S Lesniewski, and Diana E Woodward. Managing smile risk. *The Best of Wilmott*, 1:249–296, 2002.

[19] Mark Rubinstein. Displaced diffusion option pricing. *J. Finance*, 38(1):213–217, 1983.

[20] Damiano Brigo and Fabio Mercurio. Lognormal-mixture dynamics and calibration to market volatility smiles. *International Journal of Theoretical and Applied Finance*, 5(04):427–446, 2002.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[22] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[23] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[24] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. *Advances in neural information processing systems*, 29, 2016.

[25] J Yoon, D Jarrett, and others. Time-series generative adversarial networks. *Adv. Neural Inf. Process. Syst.*, 2019.

[26] Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional Sig-Wasserstein GANs for time series generation. *arXiv preprint arXiv:2006.05421*, 2020.

[27] Xiaomin Li, Vangelis Metsis, Huangyingrui Wang, and Anne Hee Hiong Ngu. Tts-gan: A transformer-based time-series generative adversarial network. In *International Conference on Artificial Intelligence in Medicine*, pages 133–143. Springer, 2022.

[28] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: deep generation of financial time series. *Quantitative Finance*, 20(9):1419–1440, 2020.

[29] Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261, August 2019.

[30] Eryk Lewinson. *Python for Finance Cookbook: Over 50 recipes for applying modern Python libraries to financial data analysis*. Packt Publishing Ltd, January 2020.

[31] Benoit B Mandelbrot. The variation of certain speculative prices. In Benoit B Mandelbrot, editor, *Fractals and Scaling in Finance: Discontinuity, Concentration, Risk. Selecta Volume E*, pages 371–418. Springer New York, New York, NY, 1997.

[32] Fischer Black. Studies of stock market volatility changes. *Proceedings of the American Statistical Association, Business & Economic Statistics Section, 1976*, 1976.

[33] Kenneth R French, G William Schwert, and Robert F Stambaugh. Expected stock returns and volatility. *J. financ. econ.*, 19(1):3–29, September 1987.

[34] G William Schwert. Why does stock market volatility change over time? *J. Finance*, 44(5):1115–1153, December 1989.

[35] J P Bouchaud, A Matacz, and M Potters. Leverage effect in financial markets: the retarded volatility model. *Phys. Rev. Lett.*, 87(22):228701, November 2001.

[36] Rama Cont. Volatility clustering in financial markets: Empirical facts and Agent-Based models. In Gilles Teyssière and Alan P Kirman, editors, *Long Memory in Economics*, pages 289–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[37] R F Engle and A J Patton. What good is a volatility model? *Quant. Finance*, 1(2):237–245, February 2001.

[38] Desmond J Higham. *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*. Cambridge University Press, April 2004.

[39] Paul Wilmott. *Derivatives: the theory and practice of financial engineering*. John Wiley Sons Ltd., 1998.

[40] Jean-Pierre Fouque, George Papanicolaou, and K Ronnie Sircar. Mean reverting stochastic volatility. *Int. J. Theor. Appl. Finance*, 03(01):101–142, January 2000.

[41] Chris Stokel-Walker and Richard Van Noorden. What ChatGPT and generative AI mean for science. *Nature*, 614(7947):214–216, 2023.

[42] Dan Milmo. Britain must become a leader in AI regulation, say MPs. *The Guardian*, 2023-08-31 00:01 BST.

[43] Cecilia Kang. Tech Chiefs to Gather in Washington Next Month on A.I. Regulations. *The New York Times*, 2023-08-28.

[44] Марков, А. А. Распространение закона больших чисел на величины, зависящие друг от друга. *Известия Физико-математического общества при Казанском университете 2-ясерия, том15, ст. 135–156*, 1906.

[45] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. Lond. Math. Soc.*, s2-42(1):230–265, 1937.

[46] Alan M Turing. Computing machinery and intelligence. *MIND*, 236:433–460, 1950.

[47] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5(4):115–133, December 1943.

[48] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[50] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, December 2013.

[51] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[52] Alex Zhavoronkov, Yan A Ivanenkov, Alex Aliper, Mark S Veselov, Vladimir A Aladinskiy, Anastasiya V Aladinskaya, Victor A Terentiev, Daniil A Polykovskiy, Maksim D Kuznetsov, Arip Asadulaev, Yury Volkov, Artem Zholus, Rim R Shayakhmetov, Alexander Zhebrak, Lidiya I Minaeva, Bogdan A Zagribelnyy, Lennart H Lee, Richard Soll, David Madge, Li Xing, Tao Guo, and Alán Aspuru-Guzik. Deep learning enables rapid identification of potent DDR1 kinase inhibitors. *Nat. Biotechnol.*, 37(9):1038–1040, September 2019.

[53] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. Accelerating science with generative adversarial networks: An application to 3D particle showers in multilayer calorimeters. *Phys. Rev. Lett.*, 120(4):042003, January 2018.

[54] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(12):4217–4228, December 2021.

[55] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, November 2014.

[56] L Theis, A Oord, and M Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[57] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 2017.

[58] B Fuglede and F Topsoe. Jensen-Shannon divergence and Hilbert space embedding. In *International Symposium on Information Theory, 2004. ISIT 2004. Proceedings.*, pages 31–, June 2004.

[59] Pedro W Lamberti and Ana P Majtey. Non-logarithmic Jensen-Shannon divergence. *Physica A: Statistical Mechanics and its Applications*, 329(1-2):81–90, 2003.

[60] Yann LeCun, Yoshua Bengio, and Others. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[61] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, February 2017.

[62] Ioannis E Livieris, Emmanuel Pintelas, and Panagiotis Pintelas. A CNN–LSTM model for gold price time-series forecasting. *Neural Comput. Appl.*, 32(23):17351–17360, December 2020.

[63] Chien-Liang Liu, Wen-Hoar Hsaio, and Yao-Chung Tu. Time series classification with multivariate convolutional neural network. *IEEE Trans. Ind. Electron.*, 66(6):4788–4797, June 2019.

[64] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[65] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[66] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[67] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. *Advances in neural information processing systems*, 30, 2017.

[68] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of GANs. *arXiv preprint arXiv:1705.07215*, May 2017.

[69] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, January 2017.

[70] David Foster. *Generative deep learning*. O'Reilly Media, Inc., 2022.

[71] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *Adv. Neural Inf. Process. Syst.*, 29, 2016.

[72] Ian J Goodfellow. On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515*, December 2014.

[73] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? pages 3481–3490 Proceedings of Machine Learning Research, January 2018.

[74] Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in GANs. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, July 2020.

[75] Eoin Brophy, Zhengwei Wang, Qi She, and Tomás Ward. Generative adversarial networks in time series: A systematic literature review. *ACM Comput. Surv.*, 55(10):1–31, February 2023.

[76] Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[77] Adrian A Dragulescu and Victor M Yakovenko. Probability distribution of returns in the Heston model with stochastic volatility. *Quant. Finance*, 2(6):443, December 2002.

[78] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, February 2018.

[79] Georg M Goerg. The Lambert way to gaussianize heavy-tailed data with the inverse of Tukey's h transformation as a special case. *Scientific World Journal*, 2015:909231, August 2015.

[80] Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, number Article 44 in ICAIF '20, pages 1–8, New York, NY, USA, October 2021. Association for Computing Machinery.

[81] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N Cohen, and Adrian Weller. Synthetic data – what, why and how? *arXiv preprint arXiv:2205.03257*, May 2022.

[82] Sergey I Nikolenko. *Synthetic Data for Deep Learning*. Springer International Publishing, 2021.

[83] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer New York, 2004.

[84] Yves Hilpisch. *Python for Finance: Analyze big financial data*. O'Reilly Media, Inc., 2014.

[85] Vance W Berger and YanYan Zhou. Kolmogorov-Smirnov test: Overview. *Wiley statsref: Statistics reference online*, 2014.

[86] The pandas development team. pandas-dev/pandas: Pandas, June 2023.

[87] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández Del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[88] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. pages 2825–2830, January 2012.

[89] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[90] François Chollet et al. Keras. `https://keras.io`, 2015.

[91] Burton G Malkiel. Efficient market hypothesis. In John Eatwell, Murray Milgate, and Peter Newman, editors, *Finance*, pages 127–134. Palgrave Macmillan UK, London, 1989.

[92] Kiyosi Ito. 109. stochastic integral. *Proceedings of the Imperial Academy*, 20(8):519–524, 1944.

[93] Damiano Brigo and Fabio Mercurio. *Interest Rate Models - Theory and Practice*. Springer Berlin Heidelberg, 2006.

[94] Tomas Björk. *Arbitrage theory in continuous time*. Oxford university press, 2009.

[95] Aurélien Alfonsi. On the discretization schemes for the CIR (and bessel squared) processes. *Monte Carlo Methods Appl.*, 11(4), January 2005.

[96] Roger Lord, Remmert Koekkoek, and Dick Van Dijk. A comparison of biased simulation schemes for stochastic volatility models. *Quant. Finance*, 10(2):177–194, February 2010.

[97] Steffen Dereich, Andreas Neuenkirch, and Lukasz Szpruch. An Euler-type method for the strong approximation of the Cox-Ingersoll-Ross process. *Proceedings of the royal society A: mathematical, physical and engineering sciences*, 468(2140):1105–1115, 2012.

[98] Eugene N White. The Stock Market Boom and Crash of 1929 Revisited. *J. Econ. Perspect.*, 4(2):67–83, June 1990.

[99] David S Bates. The crash of 87: Was it expected? the evidence from options markets. *J. Finance*, 46(3):1009–1044, July 1991.

[100] Mieszko Mazur, Man Dang, and Miguel Vega. COVID-19 and the march 2020 stock market crash. evidence from S&P1500. *Finance Research Letters*, 38:101690, January 2021.

[101] Jim Gatheral, Thibault Jaisson, and Mathieu Rosenbaum. Volatility is rough. *Quant. Finance*, 18(6):933–949, June 2018.

[102] Jean-Philippe Bouchaud. Power laws in economics and finance: some ideas from physics. *Quant. Finance*, 1(1):105, January 2001.

[103] P Gopikrishnan, M Meyer, L A N Amaral, and H E Stanley. Inverse cubic law for the distribution of stock price variations. *The European Physical Journal B - Condensed Matter and Complex Systems*, 3(2):139–140, May 1998.

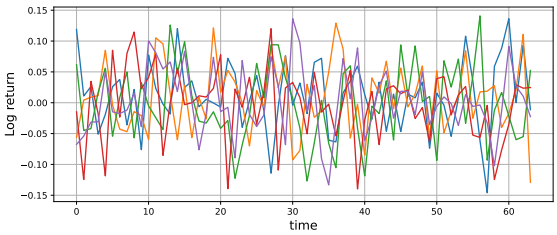
[104] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, December 2014.

[105] Pere Díaz Lozano, Toni Lozano Bagén, and Josep Vives. Neural stochastic differential equations for conditional time series generation using the Signature-Wasserstein-1 metric. *Journal of Computational Finance*, 27(1), 2023.

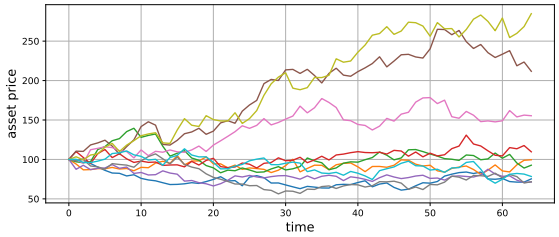
[106] Magnus Wiese, Phillip Murray, and Ralf Korn. Sig-splines: universal approximation and convex calibration of time series generative models. *arXiv preprint arXiv:2307.09767*, 2023.

8 Appendix

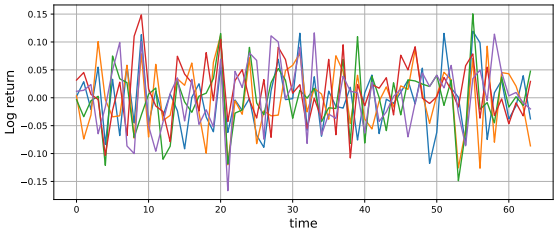
8.1 Exemplary paths for SDE models



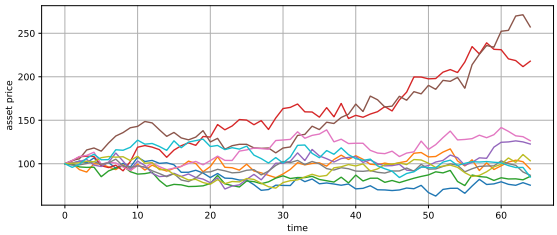
(a) Log returns from Black-Scholes SDE



(b) 10 price paths from Black-Scholes SDE

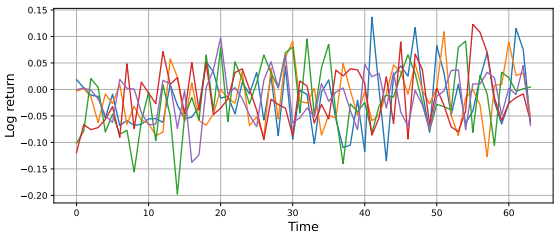


(c) Log returns from Black-Scholes GAN

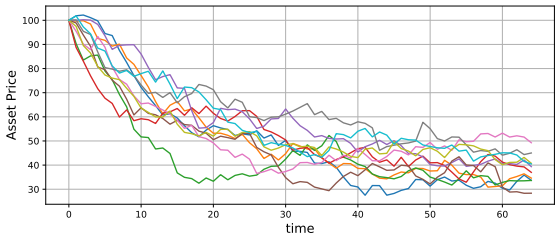


(d) 10 asset price paths from Black-Scholes GAN

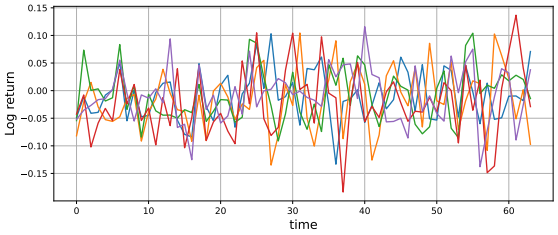
Figure 24: Examples of Black Scholes data from the training set and from GAN outputs



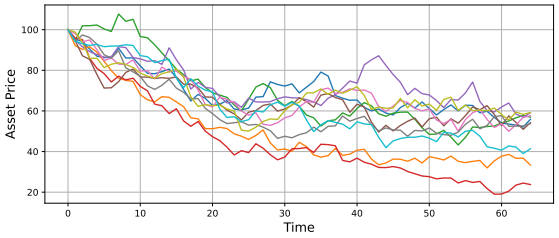
(a) Log returns from CIR SDE.



(b) 10 asset price paths from CIR SDE.

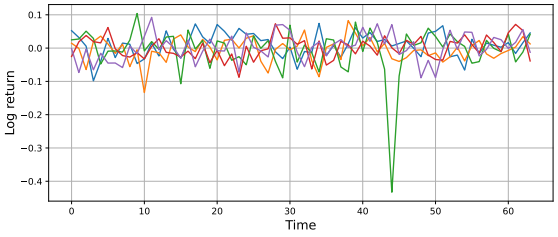


(c) Log returns from CIR GAN.

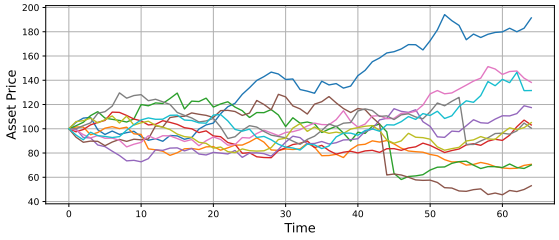


(d) 10 asset price paths from CIR GAN.

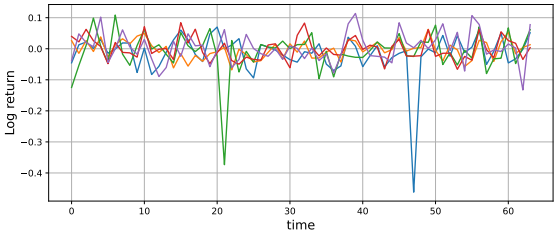
Figure 25: Examples of CIR data from the training set and from GAN outputs



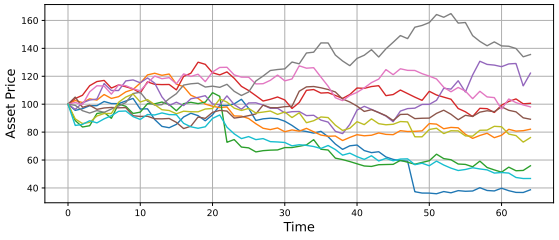
(a) Log returns from Jump Diffusion SDE.



(b) 10 price paths from Jump Diffusion SDE.



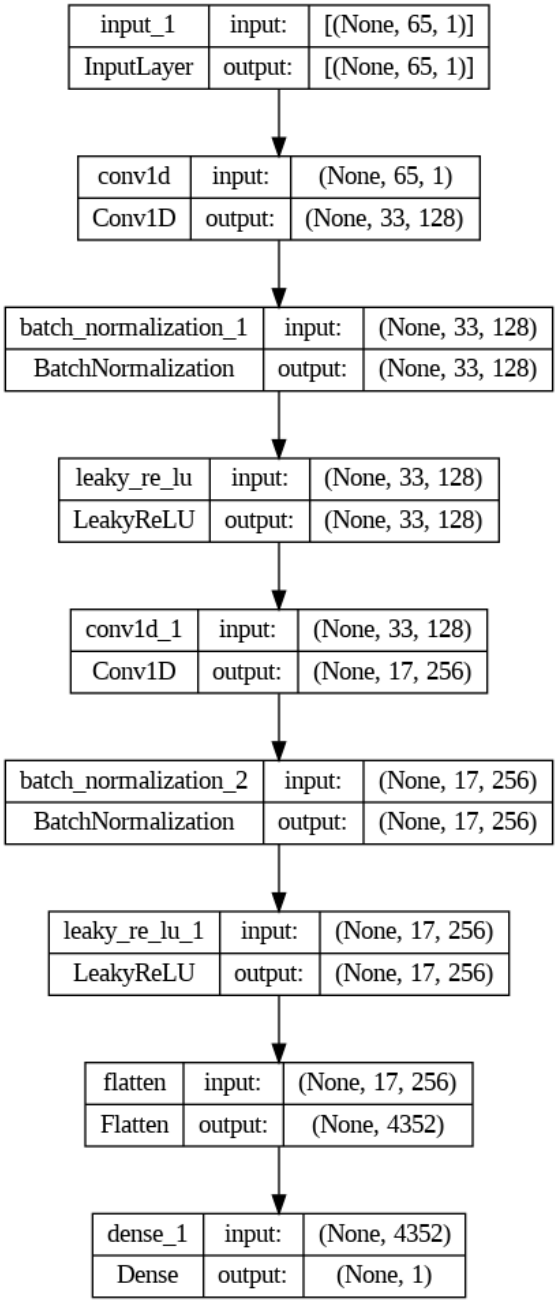
(c) Log returns from Jump Diffusion GAN.



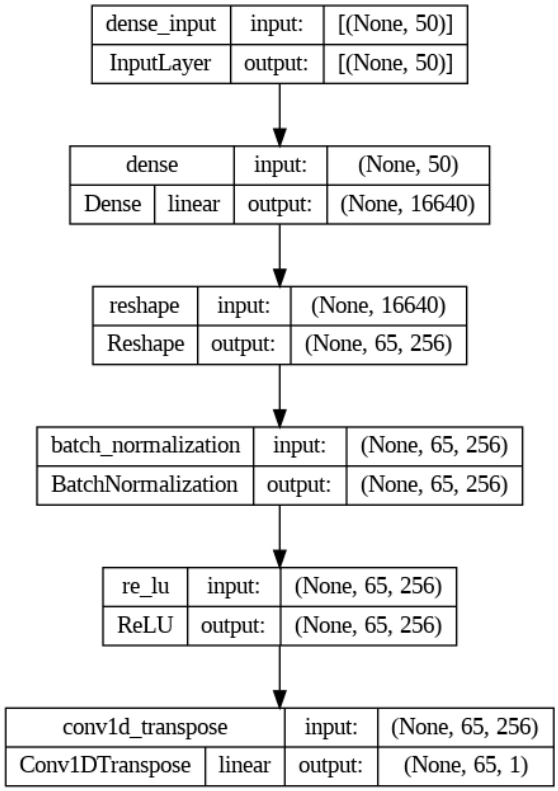
(d) 10 price paths from Jump Diffusion GAN.

Figure 26: Examples of Jump Diffusion data from the training set and from GAN outputs

8.2 Architecture for G and D - Section 4



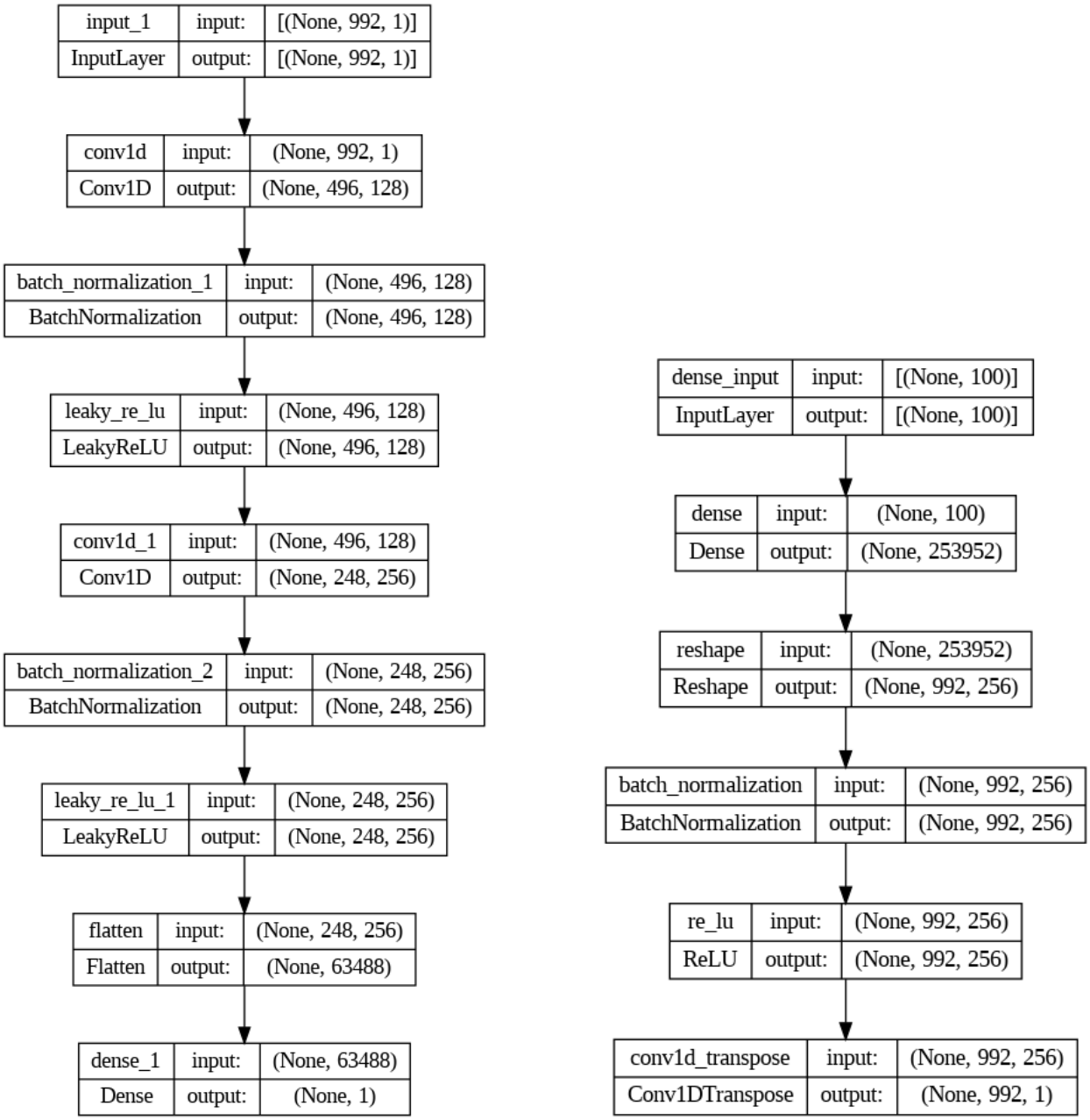
(a) Discriminator Architecture



(b) Generator Architecture

Figure 27: Unconditional GAN. G and D neural network architectures. The shallow network architecture limits the number of trainable parameters, allowing reasonable training times on standard CPU inference.

8.3 Architecture for G and D for Empirical Data Model - Section 6

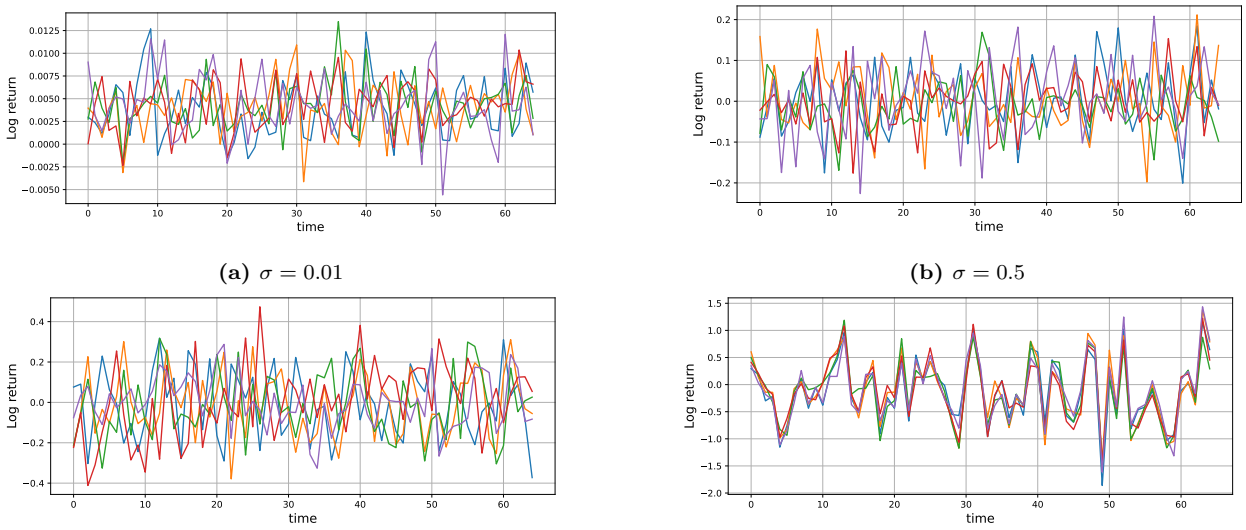


(a) Discriminator Architecture

(b) Generator Architecture

Figure 28: The G and D architectures for our GAN model applied to real asset return data.

8.4 Conditionally Generated Paths



(a) $\sigma = 0.01$

(b) $\sigma = 0.5$

(c) $\sigma = 1$

(d) $\sigma = 10$

Figure 29: Black-Scholes GAN outputs - Conditioned on volatility. Evidence of mode collapse can be seen in Figure 29d

8.5 Preprocessing functions for real stock data

```python
def extract_stock_data(tickers):
    stock_data = pd.DataFrame()

    for ticker in tickers:
        stock = yf.Ticker(ticker)
        stock_history = stock.history(period="3000d")

        if len(stock_history) >= 3000:
            stock_returns = pd.DataFrame()
            stock_returns['Close_Price'] = stock_history['Close']
            stock_data[ticker] = stock_returns['Close_Price']

    return stock_data

stock_data = extract_stock_data(tickers)

def calculate_log_returns(df):
    return np.log(df / df.shift(1))

log_returns_data = calculate_log_returns(stock_data)
log_returns_data = log_returns_data.dropna()

def rolling_window(data, window_size=992, stride=20):
```

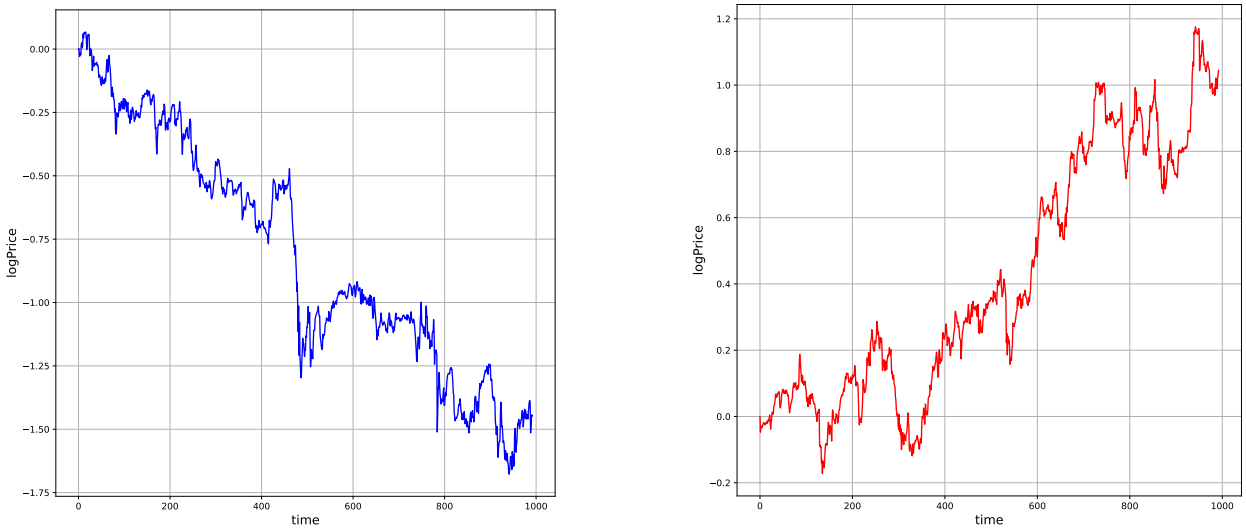
```
24
25      windows = []
26      current_index = 0
27      total_windows = (len(data) - window_size) // stride + 1
28
29      for _ in range(total_windows):
30          window = data[current_index:current_index + window_size]
31          windows.append(window)
32          current_index += stride
33
34      return windows
35
36  output = np.stack([rolling_window(log_returns_data[column]) for column in
        log_returns_data.columns], axis=1)
37  reshaped_output = output.reshape(-1, 992)
38  np.random.shuffle(reshaped_output)
```

Listing 5: Extracting data from Yahoo! finance

```
1       tickers = ['A', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABC', 'ABMD', 'ABT', '
        ACN', 'ADI', 'ADM', 'ADP', 'ADSK', 'AEE', 'AEP', 'AIZ', 'AJG', 'AKAM'
        , 'ALB', 'ALGN', 'ALK', 'ALLE', 'ALTR', 'AMAT', 'AMD', 'AME', 'AMGN',
          'AMP', 'AMT', 'AMZN', 'ANET', 'AON', 'AOS', 'APA', 'APD', 'APH', '
        ARE', 'ATVI', 'AVB', 'AVY', 'AWK', 'AXP', 'AZO', 'BA', 'BAC', 'BAX',
         'BBY', 'BDX', 'BEN', 'BF-A', 'BIIB', 'BIO', 'BK', 'BLK', 'BMRA', '
        BMY', 'BR', 'BRK-A', 'BSHI', 'BSX', 'BWA', 'BXP', 'C', 'CAG', 'CAH',
         'CAT', 'CB', 'CCI', 'CDE', 'CDNS', 'CF', 'CFG', 'CHD', 'CHRW', 'CHTR
        ', 'CINF', 'CL', 'CLX', 'CME', 'CMG', 'CMI', 'CNC', 'CNP', 'CNWT', '
        COO', 'COP', 'COST', 'COTY', 'CPB', 'CPRT', 'CRM', 'CSCO', 'CTAS', '
        CTSH', 'CUK', 'D', 'DAL', 'DE', 'DFS', 'DG', 'DGX', 'DHI', 'DIS', '
        DLTR', 'DOV', 'DPZ', 'DRI', 'DTE', 'DVA', 'DXCM', 'EA', 'EBAY', 'ECL'
        , 'ED', 'EFX', 'EIX', 'EL', 'EMN', 'EMR', 'ENS', 'EOG', 'EQIX', 'EQR
        ', 'ES', 'ESS', 'EW', 'EXR', 'F', 'FANG', 'FAST', 'FCX', 'FDX', 'FE'
        , 'FFIV', 'FIS', 'FITB', 'FLS', 'FLT', 'FMBM', 'FMC', 'FN', 'FPLPF',
         'FRMC', 'FRT', 'FTI', 'GD', 'GE', 'GGG', 'GILD', 'GIS', 'GM', 'GOOG'
        , 'GPC', 'GPN', 'GRMN', 'GS-PJ', 'GWW', 'HAL', 'HAS', 'HBAN', 'HBI',
         'HCA', 'HD', 'HES', 'HII', 'HLT', 'HOLX', 'HON', 'HPE', 'HPQ', 'HRB'
        , 'HRL', 'HSIC', 'HST', 'HSY', 'HTLF', 'HUM', 'IBM', 'ICE', 'IDXX',
        'IEX', 'IFF', 'ILMN', 'INTH', 'INTU', 'IP', 'IPGP', 'IR', 'IRM', '
        ISRG', 'IT', 'ITW', 'IVZ', 'JBHT', 'JCI', 'JKHY', 'JNJ', 'JNPR', 'JPM
        ', 'K', 'KACPF', 'KEY', 'KEYS', 'KHC', 'KIM', 'KMB', 'KMX', 'KO', '
        KR', 'KSS', 'LBTYA', 'LDOS', 'LEG', 'LH', 'LKQ', 'LMT', 'LNC', 'LNT',
          'LOW', 'LRCX', 'LUV', 'LVS', 'LYB', 'LYV', 'MAA', 'MAR', 'MCD', '
        MCHP', 'MCK', 'MCO', 'MDLZ', 'MDT', 'MET', 'MGM', 'MHK', 'MKTX', 'MLM
        ', 'MMC', 'MMM', 'MNST', 'MO', 'MOS', 'MPC', 'MRCR', 'MRK', 'MRO', '
        MS-PF', 'MSCI', 'MSFT', 'MSI', 'MU', 'NCLH', 'NCTKF', 'NDAQ', 'NEE',
        'NEOG', 'NFLX', 'NI', 'NMHLY', 'NOC', 'NOK', 'NOV', 'NOW', 'NOXL', '
        NRG', 'NSC', 'NTAP', 'NTRA', 'NTRR', 'NTRS', 'NVR', 'NVRO', 'NWL', 'O
        ', 'ODFL', 'OKE', 'OMC', 'ORLY', 'OXY', 'PAYX', 'PCAR', 'PEG', 'PEP',
          'PFE', 'PG', 'PH', 'PHM', 'PKG', 'PLD', 'PM', 'PNR', 'PNW', 'PPG',
         'PRU', 'PSX', 'PVH', 'PWR', 'PXD', 'QRVO', 'RCL', 'RE', 'REG', 'REGN
```

```
', 'RF', 'RHI', 'RIBT', 'RJF', 'RL', 'RLI', 'RMD', 'ROK', 'ROL', '
ROP', 'ROST', 'RSG', 'RSNHF', 'RXMD', 'SBUX', 'SCHW', 'SEE', 'SEGXF',
 'SHW', 'SLB', 'SLG', 'SNPS', 'SO', 'SPG', 'SRE', 'SRG', 'STT', 'STX'
, 'SWK', 'SWKS', 'SYF', 'SYK', 'T', 'TAP', 'TCYSF', 'TEL', 'TIME', '
TJX', 'TMO', 'TMUS', 'TRAUF', 'TROW', 'TRV', 'TSCO', 'TSN', 'TTWO', '
TW', 'TXN', 'TXT', 'TYL', 'UA', 'UAL', 'UDR', 'UEEC', 'UHS', 'ULTA',
 'UNM', 'UNP', 'UPS', 'URI', 'USB', 'V', 'VFC', 'VMC', 'VRSK', 'VRSN'
, 'VTR', 'VZ', 'WAT', 'WBA', 'WDC', 'WEC', 'WHR', 'WM', 'WMB', 'WRB'
, 'WRK', 'WSPOF', 'WST', 'WU', 'WY', 'WYNN', 'XEL', 'XLEFF', 'XOM', '
XYL', 'YUM', 'ZBH', 'ZION', 'ZTS']
```

Listing 6: Stock tickers used in training set. All tickers listed belong to S&P500 companies.



(a) $G(z^{(1)}) = x^{(1)}$

(b) $G(z^{(2)}) = x^{(2)}$

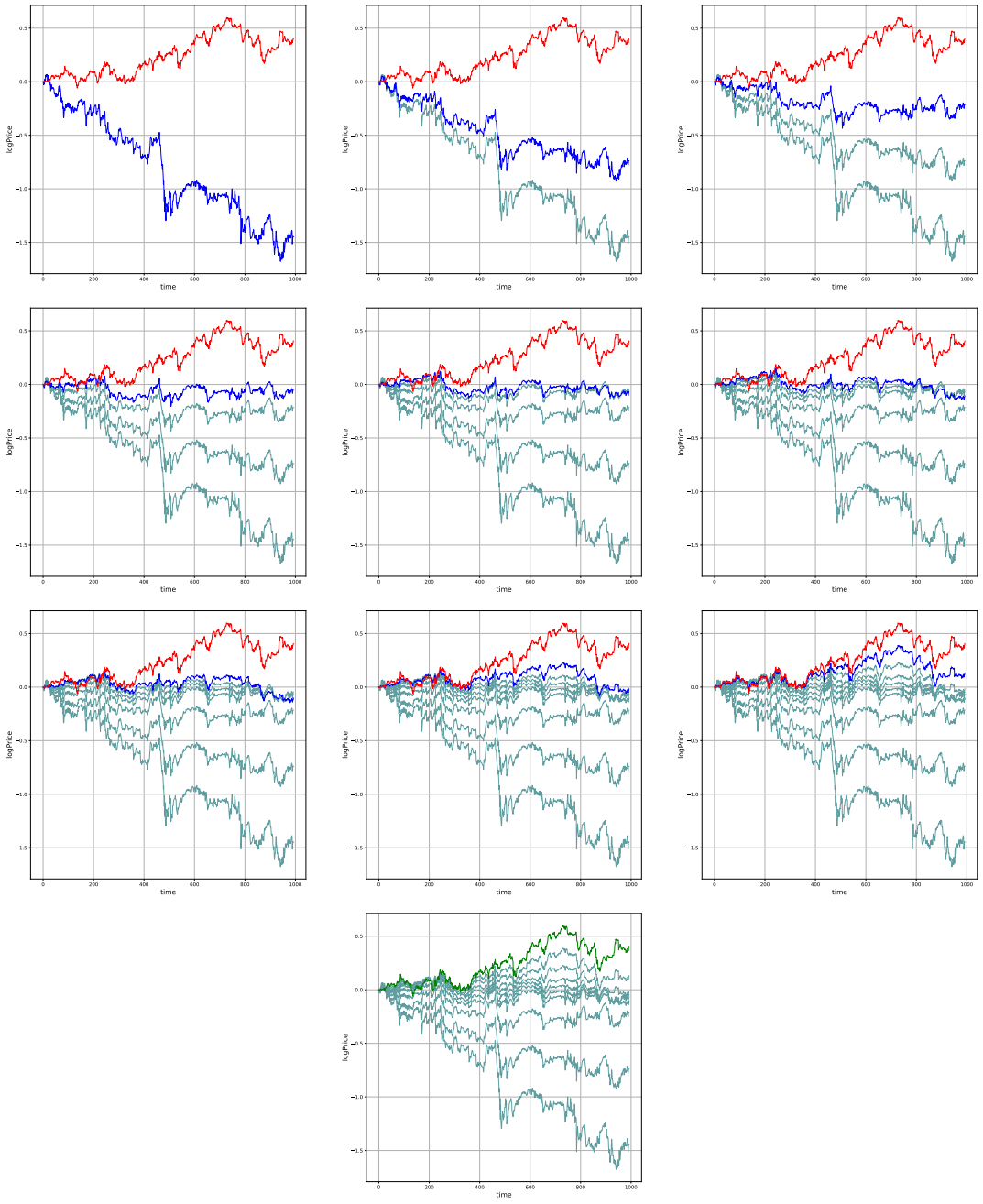
Figure 30: 2 log price paths produced by our GAN trained on real financial data.

8.6 Interpolation Through Latent Space

This subsection describes an interesting consequence of the continuity of our function G. This does not necessarily serve a practical purpose. Still, it graphically conveys an important aspect of the generative mechanism underlying GANs. From Section 3, we recall that G maps from the latent space to the defined data space:

$$G : Z \to X \tag{23}$$

Thus, we can take any two elements of our vector space Z, say $z^{(1)}$ and $z^{(2)}$, with $G(z^{(1)}) = x^{(1)}$ and $G(z^{(2)}) = x^{(2)}$; $x^{(1)}, x^{(2)} \in X$. By continuously interpolating between $z^{(1)}$ and $z^{(2)}$ in Z, we can produce a set of outputs displaying a "continuous" transformation between $x^{(1)}$ and $x^{(2)}$ in X. Take the examples shown in Figure 30. We can interpolate through the latent space Z to produce a continuous transformation from the log price path $x^{(1)}$ to the log price path $x^{(2)}$, as shown on the following page.


logPrice
time

8.7 Replication of QuantGAN Results

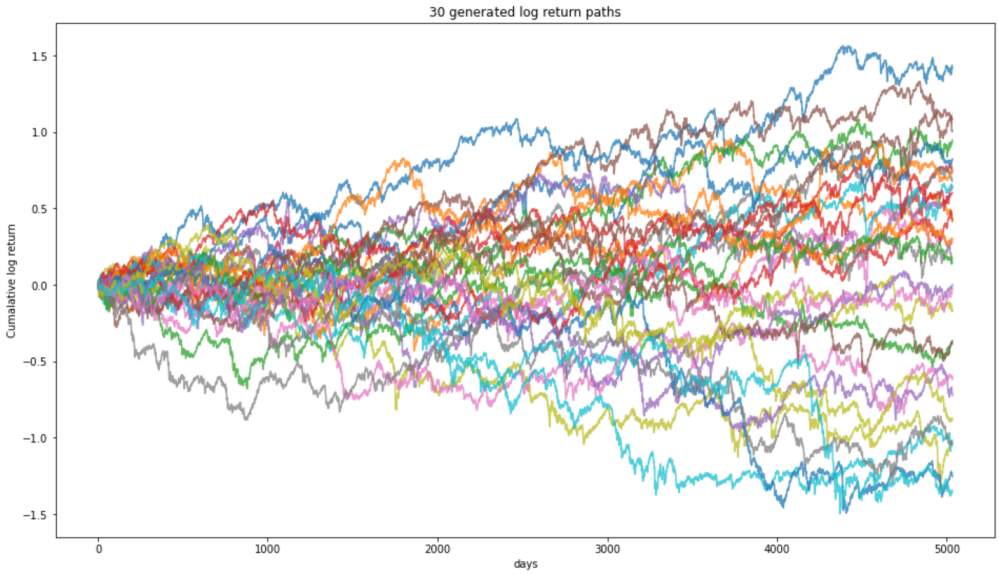


Figure 31: 30 simulated log paths using reproduced QuantGAN [28] architecture over 5032 time steps (20 years in trading days). The generator architecture used to produce these log paths can be seen in Figure 32

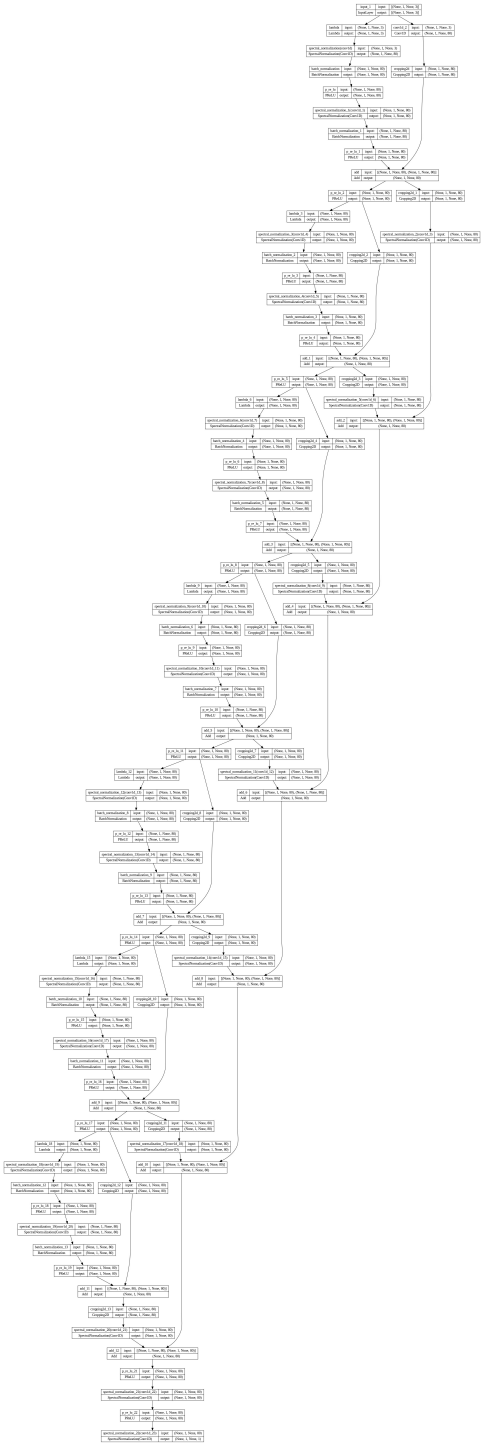


Figure 32: QuantGAN [28] TCN. Generator Architecture used to produce log paths in Figure 31. Lambda indicates identity mapping. The depth of this model is far greater than that of the shallow architectures used in our implementation (compare with Figure 28b).