Leveraging Quant GANs for Predictive Time Series Generation of High-Volatility Stocks

Author: Nikhilesh Balaji
Supervisor: Julian Sester

A thesis presented for MA2288 (UROPS)

13 April 2025

Contents

1 Introduction

AI has been used for decades to attempt to generate predictive time series. Historically, models such as the ARCH (Engle, 1982), GARCH (Bollerslev, 1986), and more were introduced for this very purpose. However, in 2014 a new model was proposed; namely, Generative Adversarial Networks (GANs henceforth). A GAN is set up by having two neural networks (called the discriminator and generator) compete against each other with each having opposing objectives (Goodfellow et al., 2014).

Wiese et al. (2020) introduces an application of GANs, they refer to as Quant GANS, as another potentially improved model to generate predictive discrete time series. Their work introduces the idea of the "Pure TCN model" which leverages Temporal Convolutional Networks (TCNs) as the key architecture for both the discriminator and generator. The paper also provides rigorous mathematical definitions of key neural network topologies, such as Multilayer Perceptron layers (MLPs) and TCNs, and frames these within the context of stochastic processes (which is what fundamentally underpins the functionality of Quant GANs in time-series generation) (Wiese et al., 2020).

The objective and contribution of this paper is to train the "Pure TCN" model using high volatility data and to analyse its predictive accuracy. Two methods are proposed to filter high volatility data: (1) a rolling standard deviation, and (2) a GARCH(1,1) volatility fit. Although the model trained on the data filtered using the GARCH(1,1) volatility fit performs comparatively better than the one trained using the rolling standard deviation filtered data, this paper is unable to fully replicate the results obtained by Wiese et al. (2020). This issue is discussed in more detail in Section 5.6.

The remainder of the paper is organized as follows. Sections 2, 3, and 4 present the key mathematical definitions introduced by Wiese et al. (2020) along with annotations that clarify how these definitions relate to neural network topologies. Section 5 details the experimental data, filtering methods, implementation of the Quant GAN, and an evaluation of the Pure TCN model's performance.

2 Neural network Topologies

We start with the definition of an MLP.

2.1 Multilayer Perceptrons

Multilayer Perceptrons are a basic feed-forward architecture/model where any single node in the model is connected to every other node in the subsequent layer. Wiese et al. (2020) first defines an activation function, a key element in the process of assigning weights to each node (Popescu et al., 2009).

Definition 2.1.1 (Activation Function) An activation function $\phi : \mathbb{R} \to \mathbb{R}$ is a

function that is Lipschitz continuous and monotonic.

The Lipschitz continuity in this context is defined as:

$$\exists K \in \mathbb{R}_{>0} \text{ such that } \forall r_1, r_2 \in \mathbb{R} \text{ it holds that}$$
$$\frac{|\phi(r_2) - \phi(r_1)|}{|r_2 - r_1|} < K \quad \text{(Tao, 2016)}$$

The monotonic condition in conjunction with the Lipschitz continuity ensures that the activation function is strictly increasing on a given subset of the domain and that "distance" between two outputs, $\phi(r_1)$ and $\phi(r_2)$ is proportional to "distance" between the respective inputs r_1 and r_2. This improves stability/convergence of the model during the training process, by not allowing large jumps in the output values during back-propagation (Xu & Zhang, 2023)

Definition 2.1.2 (Multilayer Perceptron). Let L, $N_0, ..., N_{L+1} \in \mathbb{N}$, ϕ an activation function, Θ a Euclidean vector space (parameter space). For any $l \in \{1, ..., L+1\}$ define an affine map $a_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$. A function $f : \mathbb{R}^{N_0} \times \Theta \to \mathbb{R}^{N_{L+1}}$ defined by $f(x, \theta) = a_{L+1} \circ f_L \circ \cdots \cdot f_1(x)$ where $f_l \coloneqq \phi \circ a_l$ is a multilayer perceptron with L hidden layers

$L \in \mathbb{N}$ represents the number of layers and $N_i \in \mathbb{N}$ for $i \in \{1, ..., L\}$ represents the dimension of the i^{th} hidden layer in this model. Naturally, then, N_0 and N_{L+1} represent the dimension of the input layer and output layer, respectively, and a_{L+1} represents the output layer. The behaviour of a_l is defined as for $l \in \{1, ..., L\}$ $a_l : x \mapsto W^{(l)}(x) + b^{(l)}$, where $W^{(l)} \in \mathbb{R}^{\mathbb{N}_l} \times \mathbb{R}^{N_{l-1}}$ is the weight matrix of layer l and $b^{(l)} \in \mathbb{R}^l$ the bias vector of layer l. An MLP's parameters may then be defined as:

$$\Theta \coloneqq (W^{(1)}, ...W^{(L+1)}, b^{(1)}, ..., b^{(L+1)})$$

In essence, each layer a_{l-1} has N_{l-1} nodes called $x \in \mathbb{R}^{\mathbb{N}_{l-1}}$. The nodes x are then mapped by a_{l-1} to a_l by pre-multiplying their values with a weight matrix and then adding a bias vector. The function f_{l-1} then composes the activation function with a_{l-1} and moves to the next layer of the model. This is the mathematical definition of an MLP's feedforward process (Wiese et al., 2020)

2.2 Temporal Convolutional Networks

TCNs are a type of convolutional network that are more suited for handling data with some sort of time (temporal) element, e.g. time-series data. Wiese et al. (2020) asserts that it is shown empirically that TCNs are better at handling long-range dependencies as compared to Recurrent Neural Networks (RNNs), another neural network topology.

The key element of a TCN that makes it unique is the use of *dilated causal convolutions*. A convolution is just a type of operator that has a window/filter

(called the kernel) that slides across the data manipulating it element by element. Causality implies that a model is only able to access data from the "past" time steps with no ability to access future data. The dilation aspect is a scale factor $D \in \mathbb{N}$ that creates "holes" (Wiese et al., 2020) in the layers of a model that grow exponentially. This is the element of a TCN that makes it most effective in capturing long-range dependencies. The paper first defines the central operator required for a *dilated causal convolution* called a *dilated causal convolution operator*

We adopt the notation from Wiese et al. (2020) for the rest of the discussion on TCNs in **Section 2.2**, let $N_I, N_O, K, D, T \in \mathbb{N}$. Here, N_I is the dimension of the input layer, N_O the output layer's dimension, K the kernel size, and D the dilation factor.

Definition 2.2.1 ($*_D$ operator) Let $X \in \mathbb{R}^{N_l \times T}$ be an input sequence of length T with N_l features and $W \in \mathbb{R}^{K \times N_I \times N_O}$ be the Kernel tensor containing the data on the weights for each element in the the input and output layers. Then for $t \in \{D(K-1)+1, \cdots, T\}$ and $m \in \{1, \cdots, N_O\}$, the operator $*_D$ is defined by

$$(W *_D X)_{m,t} \coloneqq \sum_{i=1}^{K} \sum_{j=1}^{N_I} W_{i,j,m} \cdot X_{j,t-D(K-i)}$$

and is called the dilated convolutional operator with dilation D and kernel size K.

The kernel size, creates a sort of temporal "skip" of size K in how the input data is processed, while the dilation scales this "skip" by a factor of D, hence the time index takes the form $t - D(K-i)$ in the above sum. The kernel tensor element $W_{i,j,m}$ represents the weight in the i^{th} position of the kernel for the j^{th} feature of the input sequence and m^{th} feature of the output sequence. One can see that this operator is analogous to the feed-forward process of applying weights in the previously discussed MLP, the key difference being the "skips" in processing the input data as a result of a kernel size and dilation factor.

Using the *dilated convolutional operator*, the paper proceeds to define a causal convolutional layer

Definition 2.2.2 (Causal Convolutional Layer) Let W be as defined in Definition 2.2.1 and $b \in \mathbb{R}^{\mathbb{N}_O}$ a bias vector. A function

$$w : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times (T - D(K-1))}$$

for $t \in \{D(K-1)+1, \cdots, T\}$ and $m \in \{1, \cdots, N_O\}$ defined by

$$w(X_{m,t}) \coloneqq (W *_D X)_{m,t} + b_m$$

is called a causal convolutional layer with dilation D

The key aspect of the convolutional layer is the effect that the dilation factor has on length of the output series to the next layer. the function w gives weight and adds bias to the m^{th} input layer at some time $t \in \{D(K-1)+1, \cdots, T\}$. This is once again just a feed-forward process.

Definition 2.2.3 (1×1 Convolutional Layer). Let $X \in \mathbb{R}^{N_I \times T}$ be an input sequence with N_I features of length T and $w : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times T}$ a causal convolutional layer with parameters $(N_I, N_O, 1, 1)$ where the last parameters are the kernel size and dilation factor, respectively. Such a layer is called a 1×1 convolutional layer.

The paper then goes on to define a *block module*, a concept that helps to generalise the building blocks of a TCN.

Definition 2.2.4 (Block Module) Let $S \in \mathbb{N}$. A function $\psi : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times (T-S)}$ that is Lipschitz continuous is called a *block module* that takes arguments (N_I, N_O, S)

The $S \in \mathbb{N}$ in the definiton represents the fact that due to the dilation factor and kernel size, the length of the output series data will be shorter moving from one layer to the next. One can choose to represent S in the l^{th} layer as

$$S \coloneqq D^l(K-1)$$

where, $l \in \{1, \cdots, L\}$ represents the layer being traversed. This notion helps with the understanding of Definition 2.2.6. With the block module defined, we are now in a good position to define a TCN, which is a collection of some arbitrary number of causal convolutional layers.

Definition 2.2.5 (Temporary Convolutional Network) Let $T_0, N_0, \cdots, N_{L+1} \in \mathbb{N}$. Further, for $l \in \{1, \cdots, L\}$, we define $S_l \in \mathbb{N}$ such that $\sum_{l=1}^{L} S_l \leq T_0 - 1$. Then, for $T_l \coloneqq T_{l-1} - S_l$ it hold that

$$T_L = T_0 - \textstyle\sum_{l=1}^{L} S_l \geq 1$$

Let, $\psi_l : \mathbb{R}^{N_{l-1} \times T_{l-1}} \to \mathbb{R}^{N_l \times T_l}$ for $l \in \{1, \cdots, L\}$ represent block modules and $w : \mathbb{R}^{N_L \times T_L} \to \mathbb{R}^{N_{L+1} \times T_L}$ represent a 1×1 convolutional layer. A function $f : \mathbb{R}^{N_0 \times T_0} \times \Theta \to \mathbb{R}^{N_{L+1} \times T_L}$ defined by

$$f(X, \theta) = w \circ \psi_L \circ \cdots \circ \psi_1(X)$$

is a temporal convolutional network with L hidden layers. We represent a TCN with L hidden layers and input layer dimension N_0 as well as output layer dimension N_{L+1} with the notation $TCN_{N_0, N_{L+1}, L}$

We can now define a special case of TCN, namely a *Vanilla TCN*.

Definition 2.2.6 (Vanilla TCN). Let $f : \mathbb{R}^{N_O \times T_O} \times \Theta \to \mathbb{R}^{N_{L+1} \times T_L}$ be a TCN with parameters (N_O, N_{L+1}, L). For all $l \in \{1, \cdots, L\}$, each block module ψ_l is defined as $\phi \circ w_l$, where w_l is the causal convolutional layer with arguments (N_{l-1}, N_l, K_l, D_l) and ϕ is an activation function. We call $f \in TCN_{N_O, N_{L+1}, L}$, a Vanilla TCN. If for all $l \in \{1, \cdots, L\}$ one has $D_l = D^{l-1}$, then f is a vanilla TCN with a dilation factor of D. Similarly, if for all $l \in \{1, \cdots, L\}$ one has $K_l = K$, then f has kernel size K.

In the paper, it has been asserted that TCNs are empirically better at capturing long-range dependencies. This is as a result of a metric known as the Receptive

Field Size (RFS), that is, loosely speaking, the number of elements in a given series of data that the model (TCN in our case) is able to capture and process. A more rigorous formulation given by the paper is the following:

Definition 2.2.7 (Receptive Field Size) let $f \in TCN_{N_Q,N_{L+1},L}$ and let $S_1, \cdots, S_L$ be defined as in Definition 2.2.4. i.e $S_l \coloneqq D^l(K-1)$. The number $T^{(f)} \in \mathbb{N}$ defined as

$$T^{(f)} \coloneqq 1 + \sum_{l=1}^{L} S_l$$

is called the receptive field size (RFS) of f.

3 Generative adversarial networks

Generative adversarial networks are a model with two parts, a generator and a discriminator. In the context of time series generation, the generator's role is to try to generate some sequence of data that the discriminator will take as an input to verify whether the data is synthetic or real. This creates an adversarial setup where the generator and discriminator engage in a two player game. The generator aims to create synthetic data that is identical in distribution to that of the target data. A more rigorous framing of this idea is given below:

Let $(\mathbb{R}^{N_Z}, \mathcal{B}(\mathbb{R}^{N_Z})$ be the *latent measure space* and $(\mathbb{R}^{N_X}, \mathcal{B}(\mathbb{R}^{N_X})$ be the *target/data measure space*. The generator will sample some random variable $Z \in (\mathbb{R}^{N_Z}, \mathcal{B}(\mathbb{R}^{N_Z})$. The GAN's objective is to train the network $g : \mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ to be able to match the distribution of $g_\theta(Z) \coloneqq g_\theta \circ Z$, for parameters $\theta \in \Theta^{(g)}$, to the distribution of the targeted random variable $X \in (\mathbb{R}^{N_X}, \mathcal{B}(\mathbb{R}^{N_X})$ such that the discriminator will not be able to identify $g_\theta(Z)$ as synthetic/generated.

Naturally, we should introduce a mathematical definition and notation for the generator and discriminator as proposed by Wiese et al. (2020).

Definition 3.1 (Generator) The network $g : \mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ with some parameter space $\Theta^{(g)}$ is called a *generator*. Additionally the random variable $\tilde{X}$ defined by:

$$\begin{aligned} \tilde{X} : \Omega \times \Theta^{(g)} &\to \mathbb{R}^{N_X} \\ (\omega, \theta) &\mapsto g_\theta(Z(\omega) \end{aligned}$$

is called a generated random variable.

Definition 3.2 (Discriminator). Let $\tilde{d} : \mathbb{R}^{N_X} \times \Theta^{(d)} \to \mathbb{R}$ be a network with parameters $\eta \in \Theta^{(d)}$ and an activation function $\sigma : \mathbb{R} \to [0,1]$ such that $x \mapsto \frac{1}{1+e^{-x}}$ called the sigmoid function. A function $d : \mathbb{R}^{N_X} \times \Theta^{(d)} \to [0,1]$ defined by $(x, \eta) \mapsto \sigma \circ \tilde{d}_\eta(x)$ is called a discriminator.

Given, these key components of a GAN, it is worth mathematically defining how the discriminator and generator are optimised during the training process. The loss function of a GAN is given by:

$$\begin{aligned}\mathcal{L}(\theta,\eta) &\coloneqq \mathbb{E}[\log(d_\eta(X))] + \mathbb{E}[1-\log(d_\eta(g_\theta(Z)))] \\ &= \mathbb{E}[\log(d_\eta(X))] + \mathbb{E}[1-\log(d_\eta(\tilde{X_\theta}))]\end{aligned}$$

The first term given by $\mathbb{E}[\log(d_\eta(X))]$, is the log expected probability that the discriminator is able to classify real data $X \in \mathbb{R}^{N_X}$ as real. The second term given by $\mathbb{E}[1-\log(d_\eta(\tilde{X_\theta}))]$, is the log expected probability that the discriminator classifies some synthetic/generated data $\tilde{X_\theta}$ as fake. The training process has 2 parts:

1. The discriminator is trained by tweaking its parameters $\eta \in \Theta^{(d)}$ to maximize $\mathcal{L}(\theta,\eta)$, essentially making it able to distinguish between synthetic and real data
2. The generator is trained to by tweaking its paraemters $\theta \in \Theta^{(g)}$ to minimize $\mathcal{L}(\theta,\eta)$ by attempting to fool the discriminator into thinking some generated data from the latent measure space is real data.

This gives rise to the min-max game:

$$\min_{\theta\in\Theta^{(g)}} \max_{\eta\in\Theta^{(d)}} \mathcal{L}(\theta,\eta)$$

also called the GAN objective. The updating of parameter values based on this objective is done during the gradient descent process.

4 Adapting GANs for Stochastic Processes

First we adopt the the paper's notation:
Let $(X_t)_{t\in\mathbb{Z}}$ be a stochastic process with parameters $\theta \in \Theta$. For $s,t \in \mathbb{Z}$ such that $s \leq t$ we write

$$X_{s:t,\theta} \coloneqq (X_{s,\theta}, \cdots, X_{t,\theta})$$

and for some ω-realisation:

$$X_{s:t,\theta}(\omega) \coloneqq (X_{s,\theta}(\omega), \cdots, X_{t,\theta}(\omega)) \in R^{N_X \times (t-s+1)}$$

We can now define a *neural process*

Definition 4.1(Neural Process). Let $(Z_t)_{t\in\mathbb{Z}}$ be an independent and identically noise process with values in $\mathbb{R}^{N_Z}$ and define a TCN with $RFS = T^{(g)}$ taking parameters $\theta \in \Theta^{(g)}$ by $g : \mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$. A stochastic process $\tilde{X}$ defined by

$$\begin{aligned}\tilde{X} : \Omega \times \mathbb{Z} \times \Theta^{(g)} &\to \mathbb{R}^{N_X} \\ (\omega, t, \theta) &\mapsto g_\theta(Z_{t-(T^{(g)}-1):t}(\omega))\end{aligned}$$

such that $\tilde{X}_{t,\theta} : \Omega \to \mathbb{R}^{N_X}$ is a $\mathcal{F} - \mathcal{B}(\mathbb{R}^{N_X})$ measurable map for all $t \in \mathbb{Z}$ and $\theta \in \Theta^{(g)}$ is called a *neural process* which is denoted by $\tilde{X_\theta} \coloneqq (\tilde{X}_{t,\theta})_{t\in\mathbb{Z}}$

The noise process $(Z_t)_{t\in\mathbb{Z}}$ represents the noise prior (in the context of GANs). The paper also adopts the convention that $t \in \mathbb{Z}$ (ie. discrete time) and that Z_t follows a standard multivariate distribution, ie. $Z_t \sim \mathcal{N}(0, I)$. So the neural process $\tilde{X}_\theta \coloneqq (\tilde{X_{\theta,t}})_{t\in\mathbb{Z}}$ is the sequence generated by feeding Z to the TCN, g, from definition 4.1.

Similarly, the discriminator $d : \mathbb{R}^{N_X \times T^{(d)}} \times \Theta^{(d)} \to [0, 1]$ is a TCN with RFS $T^{(d)}$. This now allows us to reframe the GAN objective in the context of stochastic processes. Namely the min-max game,

$$\min_{\theta\in\Theta^{(g)}} \max_{\eta\in\Theta^{(d)}} \mathcal{L}(\theta, \eta)$$

is maintained but we redefine the loss function by,

$$\mathcal{L}(\theta, \eta) \coloneqq \mathbb{E}[\log(d_\eta(X_{1:T^{(d)}}))] + \mathbb{E}[log(1 - d_\eta(\tilde{X}_{1:T^{(d)},\theta}))]$$

where, $X_{1:T^{(d)}}$ is the real process, while $\tilde{X}_{1:T^{(d)},\theta}$ is the generated process. The paper then describes the training process by first getting the realizations $\{\tilde{x}^{(i)}_{1:T^{(d)},\theta}\}_{i=1}^M$ and $\{\tilde{x}^{(i)}_{1:T^{(d)}}\}_{i=1}^M$ of samples of size M of the generated neural process and the target distribution, respectively. The realizations are then fed to the discriminator which outputs a real value in the interval $[0, 1]$ which are subsequently averaged to give the following "Monte Carlo estimate of the discriminator's loss function":

$$\mathcal{L}(\theta, \eta) \approx \frac{1}{M}\sum_{i=1}^M \log(d_\eta(\tilde{x}^{(i)}_{1:T^{(d)}})) + \frac{1}{M}\sum_{i=1}^M \log(1 - d_\eta(\tilde{x}^{(i)}_{1:T^{(d)},\theta}))$$

5 Quant GANS during Volatile Periods

Wiese et al. (2020) use data from the S&P 500 index to train their model. Being a market index, S&P 500 is less affected by the extreme fluctuations in price that individual stocks often experience. This offers much more stability during the training process.

In this section the aim is to analyse and evaluate how effectively the pure TCN model proposed by Wiese et al. (2020) is able to generate predictive time-series during periods of higher volatility. NVIDIA's historic close prices were used in this investigation. The data was taken from yahoo finance ranging from 22/1/1999 to 7/3/2025 consisting of 6573 data points.

5.1 Implementation and Method

JamesSullivan (2021) published an implementation of the Pure TCN QuantGAN model proposed by Wiese et al. (2020). This implementation was used while modifying the original code to ensure the historic closing data was processed appropriately.

For the training and testing of the NVIDIA log returns the Pure TCN model is used. Just as in Wiese et al. (2020), a TCN with a receptive field size $T^{(g)} = 127$

is used as the generator. In the context of the notation from Section 4, Given a three-dimensional noise prior $Z_t \overset{iid}{\sim} \mathcal{N}(0,1), (N_Z = 3)$ the generator's return process is given by:

$$R_{t,\theta} = g_\theta(Z_{t-(T^{(g)}-1):t})$$

where as in Definition 3.1, g_θ represents the network g along with parameters from a parameter space $\Theta^{(g)}$ known as the generator.

5.2 NVIDIA Data used

First, the high volatility time periods within the NVIDIA data set are flagged out and are then filtered out. Subsequently, from this high volatility data set, the most recent 100 data points are set aside to be used for testing (see Section 5.4). Doing so maintains the temporal structure of the time series. The remainder of the high volatility dataset is then used to train the model.

To compare the model's behaviour when trained with high volatility data, we also train it with the entire NVIDIA data set as another benchmark just as in Wiese et al. (2020). In this case we use 10% of the entire data set for testing purposes and use the remaining 90% for training.

The reason for choosing data in such a manner is to ensure that the models are at least trained on enough data while still allowing us to calculate metrics to benchmark their performance; a consequence of the shortage of data in the high volatility data sets.

The next subsection describes 2 methods that were used to filter out high volatility

5.3 Filtering of Data

In order to filter out higher volatility time periods in the NVIDIA time series, 2 methods were used.

1. (Rolling Standard Deviation Fit) A 30-day rolling standard deviation that was used to capture the volatility of stock prices at various times.
2. (Garch(1,1) Volatility Fit) Using the GARCH(1,1) models volatility fit to get a volatility approximation for the data.

Subsequently, the volatility values that fall in the upper quartile of both the filtered data sets are classified as "high volatility" and a portion of that data set is used to train the model. A summary of the data used is given by Table 1.

5.4 Distributional Metrics

The primary question motivating this paper is how closely the synthetic time series' fit the historical time series' distribution. The 2 distributional metrics

	Training Data		Testing Data	
Time Series	**Time Span**	**# of Data points**	**Time Span**	**# of Data points**
NVIDIA (Full)	Jan 1999 - Jul 2022	5916	Aug 2022 - Mar 2025	657
Rolling St. deviation fit	Mar 1999 - Dec 2022	1533	Jan 2023 - Mar 2025	104
GARCH(1,1) volatility fit	Jan 1999 - Dec 2022	1544	Jan 2023 - Mar 2025	100

Table 1: Summary of data sets used for training the model and testing the model

as proposed by Wiese et al. (2020) that will be used are the Wasserstein Distance/Earth movers Distance and the Drăgulescu & Yakovenko (DY) Metric.

We will be using these metrics to compare the historical and synthetic distribution of the t-differenced log returns which we now define below:

Definition 5.4.1 (t-differenced log returns) Given a time series of closing prices $\{S_n\}_{n=0}^{N}$ for $n \in \mathbb{N}$, the t-differenced log returns are defined as $r_{t,n} \coloneqq \log(\frac{S_n}{S_{n-t}})$ for $n \in \{t, t+1, ...N\}$ and $t \in \mathbb{N}$

We now introduce in detail the distributional metrics that will be used in the analysis of the Pure TCN model:

1. **Wasserstein Distance/Earth Movers Distance**
   Let $\mathbb{P}^h$ and $\mathbb{P}^g$ be the historical distribution and the generated distribution of the log returns respectively. Let $\Pi(\mathbb{P}^h, \mathbb{P}^g)$ represent the set of all joint distributions $\pi(x, y)$ with the marginal distributions $\mathbb{P}^h$ and $\mathbb{P}^g$. Also, let $F^{-1}_{\mathbb{P}^h}(x)$ and $F^{-1}_{\mathbb{P}^g}(x)$ represent the quantile function of $\mathbb{P}^h$ and $\mathbb{P}^g$ respectively. Below are equivalent ways of characterising the 1-D Wasserstein distance between $\mathbb{P}^h$ and $\mathbb{P}^g$ (Villani, 2008):
   $$EMD(\mathbb{P}^h, \mathbb{P}^g) = \inf_{\pi \in \Pi(\mathbb{P}^h, \mathbb{P}^g)} \int |x - y| d\pi(x, y)$$
   Or
   $$EMD(\mathbb{P}^h, \mathbb{P}^g) = \int_0^1 |F^{-1}_{\mathbb{P}^h}(x) - F^{-1}_{\mathbb{P}^g}(x)| dx$$
   The second equation offers a more intuitive explanation of EMD; namely the "area" between the historical and synthetic log returns' quantile distributions. This roughly describes how much effort or work it takes to transform the historical log returns' distribution into the synthetic returns' distribution (Wiese et al., 2020)

2. **Drăgulescu & Yakovenko (DY) Metric**
   This metric used in Wiese et al. (2020) was originally used by Drăgulescu & Yakovenko in Drăgulescu & Yakovenko (2002). Let $t \in \mathbb{N}$ be the time lag, $\mathbb{P}^h_t$ and $\mathbb{P}^g_t$ be the historical and generated log returns' t-differenced empirical probability distributions. We also define $A_{t,x}$ to be the xth "bin" of the data which partitions the distributions' log returns on the real number line. A more detailed explanation within the context of discussion is as below:

Definition 5.4.2 Given the t-differenced log returns $\{r_{t,n}\}_{n=0}^{N}$ of a time series $\{S_n\}_{n=0}^{N}$ as in Definition 5.3.1, we consider the collection of subsets of the t-differenced log returns $\{A_{t,x}\}_{x\in I}$ indexed over a finite family I with the following two properties:

$$\text{for any } x, y \in I \text{ with } x \neq y \text{ one has } A_{t,x} \cap A_{t,y} = \emptyset$$

And

$$\bigcup_{x\in I} A_{t,x} = \{r_{t,n}\}_{n=0}^{N}$$

Just as in Wiese et al. (2020), we construct the partitions such that each partition captures approximately an equal probability mass of $\frac{5}{T}$ for the total number of log returns $T \in \mathbb{N}$. The number of bins $N \in \mathbb{N}$ is then naturally approximated as $N := \frac{T}{5}$. The formula to compute the metric itself now follows as below:

$$DY(t) = \sum_{x} |\log \mathbb{P}_t^h(A_{t,x}) - \log \mathbb{P}_t^g(A_{t,x})|$$

It is then clear that a lower DY value implies that the empirical historical and synthetic distributions are similar to each other.

Both metrics will be evaluated on daily, weekly, monthly log returns just as in Wiese et al. (2020) (ie. using t-differenced log returns for $t \in \{1, 5, 20\}$). Since our testing data set for the high volatility data only has close to 100 data points, we choose to omit calculation of their metrics on the 100 day log returns (100-differenced log returns) as it would only yield very few data points. This could lead to an inaccurate or unreliable comparison of models.

5.5 Results

For all three data sets, the generated log return paths seem to follow similar distributions for day 0 to 20 after which they begin to diverge in terms of how similar the paths are as in Figure 1, Figure 3, and Figure 5. For the high volatility data sets, the reason for the large magnitude of cumulative log returns (greater than 1 or less than -1) is that high volatility data was used to train the model so there are occasionally large jumps in the values of the generated stock prices leading to large jumps in log returns.

The synthetic data was generated from 3-dimensional Gaussian noise which after processing has been standardised, which explains why the synthetic data histograms in Figure 2, Figure 4 and Figure 6 takes the rough shape of standardised Gaussian curves.

The GARCH(1,1) volatility fit allows for synthetic log returns that behave more similarly to the historic distribution, as compared to the rolling standard deviation fit. Yet, it still seems to be less accurate than what is proposed by Wiese et al. (2020). This is clear from looking at the EMD values of all the data sets in Table 2, which are higher by an order of roughly 1000. So, contrary to

	Rolling Std. Dev	GARCH(1,1)	Entire Data set
EMD(1)	1.1838	1.0566	1.0776
EMD(5)	1.1190	1.0579	1.0921
EMD(20)	1.0390	1.0225	1.0468
EMD(100)	-	-	1.1287
DY(1)	14.8544	16.2330	16.7044
DY(5)	13.6304	14.8876	17.5745
DY(20)	14.8645	15.5635	18.1864
DY(100)	-	-	16.9452

Table 2: EMD and DY values for rolling standard deviation volatility fit, GARCH(1,1) volatility fit and Entire Data set (from left to right) with respect to their training data sets. EMD(t) and DY(t) are calculated for the historical and synthetic distributions' t-differenced log return paths for $t \in \{1, 5, 20, 100\}$

expectation, even the model trained on the entire NVIDIA data set seems to yield much greater EMD values for all t-differenced log return paths.

However, the DY metric for both models is less than what is proposed by Wiese et al. (2020). Since the DY metric is a measurement of the empirical distributions, it seems to indicate that the pure TCN model that was used seems to be modeling the "shape" of the distribution accurately but is not necessarily able to center or scale the generated log return paths accurately. In fact, Figure 2, Figure 4, and Figure 6 illustrate exactly this.

5.6 Evaluation of Results

As mentioned at the beginning of Section 5, Wiese et.al (2020) usage of S&P 500 time series to train their model offers a few benefits. One such advantage that is important to highlight is the availability of S&P 500 data starting from 1987 (on Yahoo Finance). NVIDIA time series only begins in 1999 which naturally leaves us with fewer data points with which to train the model. This shortage of data is even further evident when we need to begin filtering the data set for high volatility data. This is a very natural limitation with no technical workaround.

Another key difference that was previously mentioned is the stability that S&P 500 offers, being a market index. NVIDIA, being an individual stock, is naturally more volatile than a market index. So, even for the distributional metric of the entire data set, the comparative volatility of NVIDIA and S&P 500 must be accounted for.

6 Conclusion

This paper has illustrated that applying the "Pure TCN" model directly to generate time series of high volatility stocks yields inaccurate log returns paths. Some aspects of the model may need to be changed for it to be more generally

applicable to any stock. One such idea could be to use a non-gaussian latent space from which the noise prior is fed to the generator. This would certainly lead to different shapes and results in the output, with some choices perhaps even offering distributional metrics that match that of Wiese et al. (2020).

References

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327. https://doi.org/https://doi.org/10.1016/0304-4076(86)90063-1

Drăgulescu, A. A., & Yakovenko, V. M. (2002). Probability distribution of returns in the heston model with stochastic volatility*. *Quantitative Finance*, *2*, 443–453. https://doi.org/10.1080/14697688.2002.0000011

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, *50*(4), 987–1007. Retrieved April 12, 2025, from http://www.jstor.org/stable/1912773

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. https://arxiv.org/abs/1406.2661

JamesSullivan. (2021). Temporalcn/torch$_t$*rain.ipynbatmainjamessullivan/temporalcn*. *GitHub*. Retrieved March 23, 2025, from https://github.com/JamesSullivan/temporalCN/blob/main/torch_train.ipynb

Popescu, M.-C., Balas, V., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*.

Tao, T. (2016). *Analysis i third edition*. Singapore Springer Singapore, Imprint: Springer.

Villani, C. (2008, October). *Optimal transport*. Springer Science Business Media.

Wiese, M., Knobloch, R., Korn, R., & Kretschmer, P. (2020). Quant gans: Deep generation of financial time series. *Quantitative Finance*, *20*, 1419–1440. https://doi.org/10.1080/14697688.2020.1730426

Xu, Y., & Zhang, H. (2023). Uniform convergence of deep neural networks with lipschitz continuous activation functions and variable widths. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2306.01692

A Numerical Results

A.1 Rolling Standard Deviation Fit

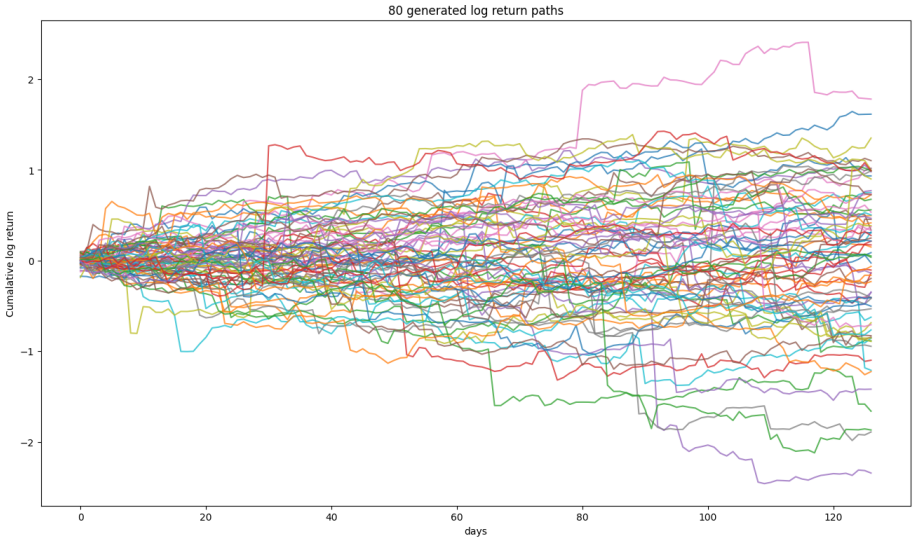

Figure 1: 80 log return paths generated by the model using the Rolling Standard Deviation Fit

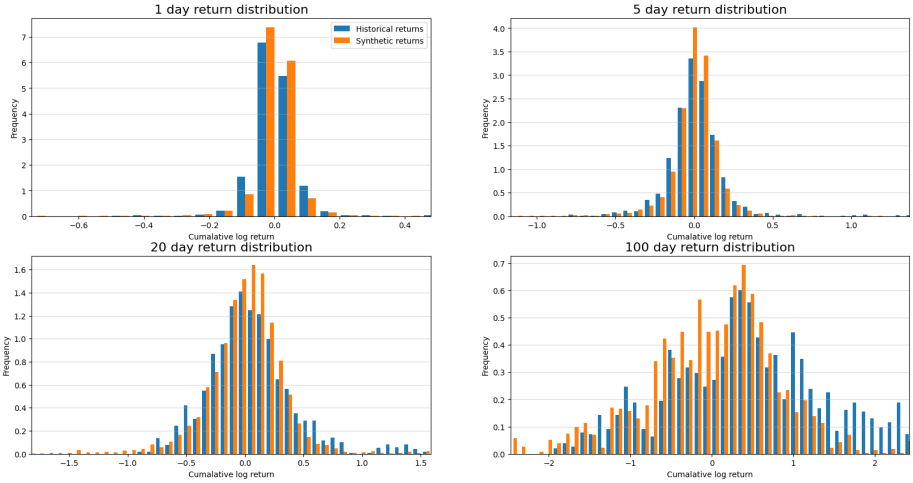

Figure 2: Histograms of the historical and synthetic distributions' daily, weekly, monthly and 100-day log return distributions using the rolling standard deviation to filter the data

A.2 GARCH(1,1) Volatility Fit

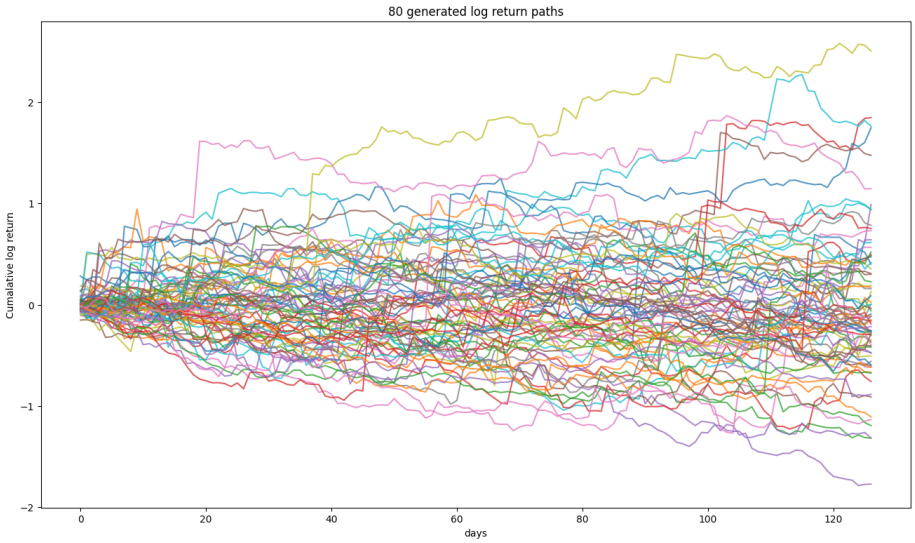

Figure 3: 80 log return paths generated by the model using the data filtered by the GARCH(1,1) Volatility Fit

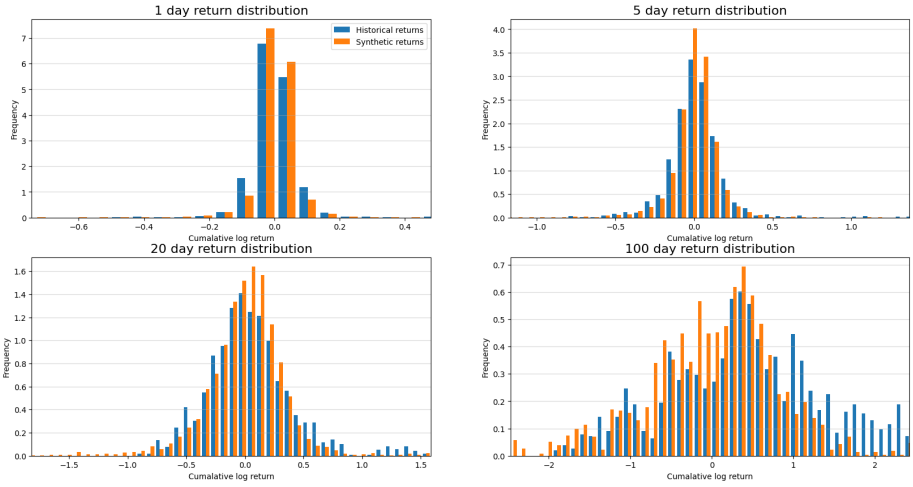

Figure 4: Histograms of the historical and synthetic distributions' daily, weekly, monthly and 100-day log return distributions using the GARCH(1,1) volatility fit to filter the data

A.3 Entire Data Set

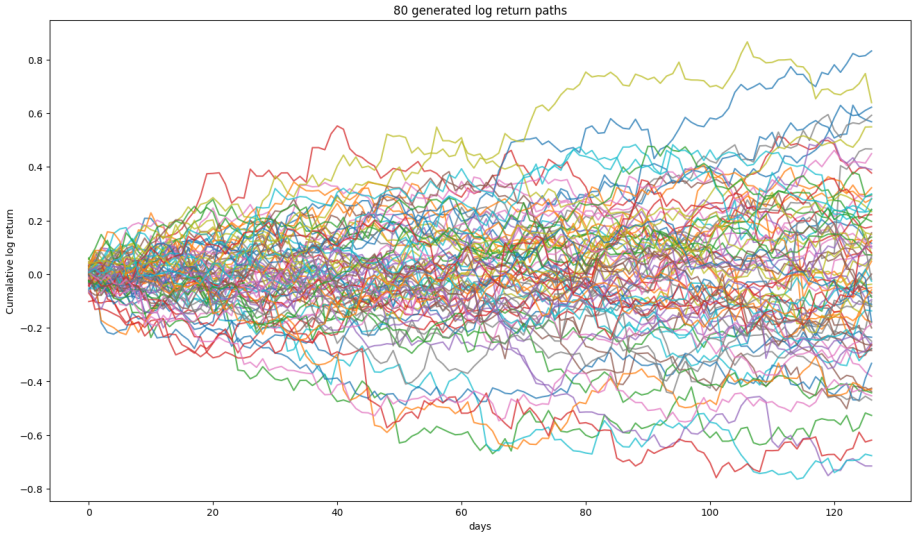

Figure 5: 80 log return paths generated by the model trained using the entire data set

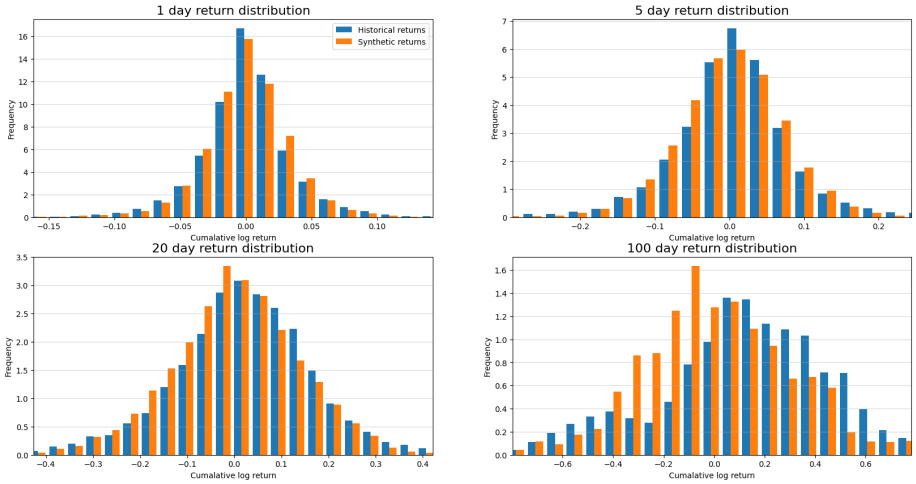

Figure 6: Histograms of the historical and synthetic distributions' daily, weekly, monthly and 100-day log return distributions using the entire data set

B Relevant Code

All code used for this paper were from

- https://github.com/JamesSullivan/temporalCN
- https://github.com/nikhilb66/QuantGANs-UROPS