IEOR223: Final Project Report
Generative AI for Time Series Generation

Group 4
Adrian Enders, Louis Battesti, Loic Jannin, Paul Faverjon,
Jasmine Zhang, Alexandre Soppelsa, Francois Gudel

University of California, Berkeley
May 2024

Contents

1 Introduction

Using neural networks to produce synthetic stock data has become an ever-growing field of interest in financial engineering. It helps traders back-test their models on simulated environments and perform risk modeling for abnormal events. Our project intends to produce new models that can beat or match current industry standard models. However, this can be hard due to our group's computational limitations. Thus, we employ methods such as Principal Component Analysis (PCA) to reduce computational demands.

2 Problem Definition

This project focuses on producing novel generative models for historical stock data. Such models include GAN, VAE, and Consistency models. The samples from these models are then assessed by comparing them to well-known time series generative models. This is done by comparing the models' discriminator scores, which is the sample's similarity to the original data. However, due to the computational limitations of the models produced, comparison to well-known models is somewhat subjective.

To support our analysis, we use normalized historical stock data from Google, sourced from Yahoo Finance. This dataset is widely recognized in financial research, providing a standard benchmark for evaluating our models. Using this familiar dataset helps ensure that our findings are both robust and comparable to existing models, despite the computational limitations of our approaches.

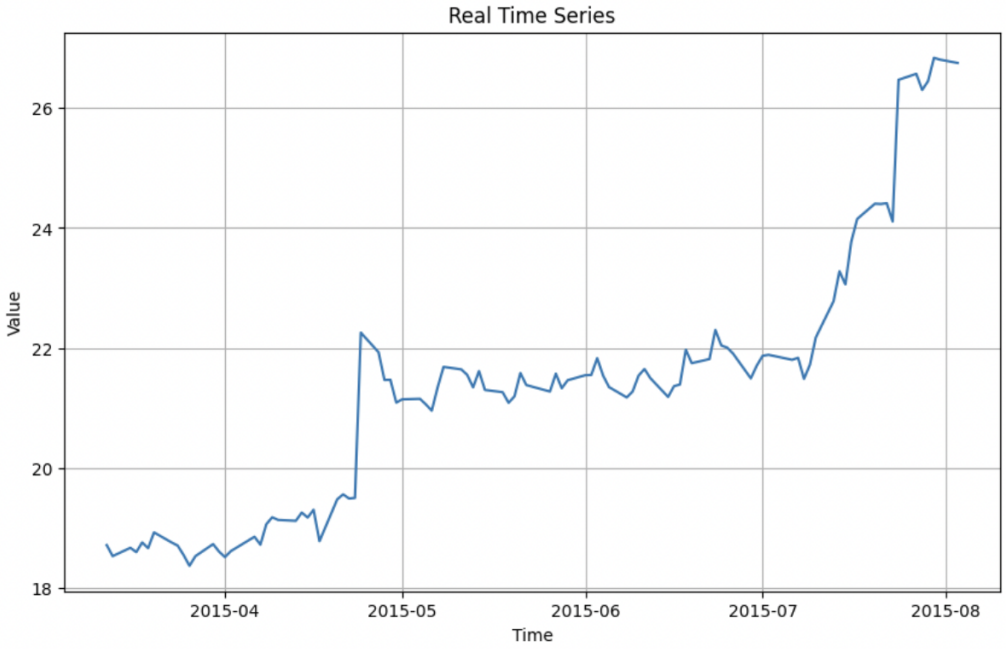


Figure 1: Google, time serie

3 Models Developed

3.1 TimeGAN

For our TimeGAN model, we used the ydata TimeSeriesSynthesizer implementation first introduced in Jinsung Yoon and Daniel Jarrett's 2019 paper. TimeGAN consists of four network components: an embedding function, a recovery function, a sequence generator, and a sequence discriminator. The key insight is that the auto-encoding components (first two) are trained jointly with the adversarial components (latter two), such that TimeGAN simultaneously learns to encode features, generate representations, and iterate across time.

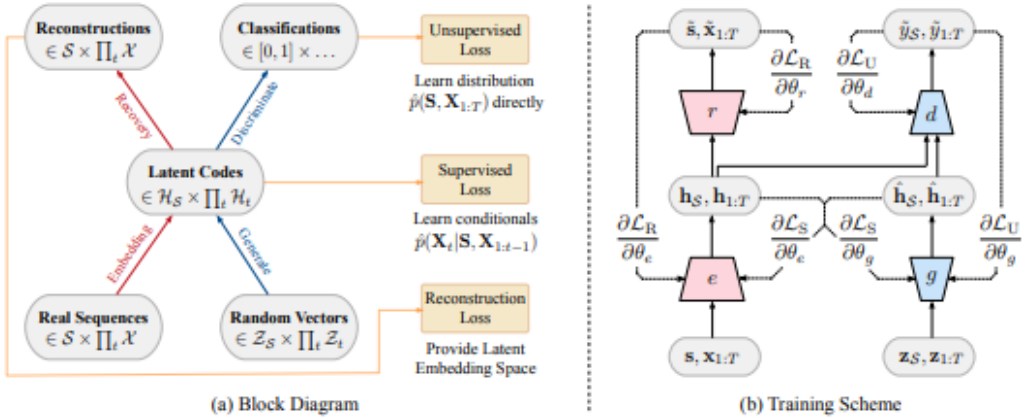


Figure 2: TimeGAN Architecture [1]

One difference between our model and the one mentioned in the paper is that our model is trained on 500 epochs compared to the 50,000 mentioned in the paper. This led to suboptimal samples; however, it effectively showed how the PCA affected the model's performance.

3.2 VAE

The Variational Auto-Encoder is a model that uses an encoder and a decoder. The encoder reduces the dimension of the input data by computing parameters, here the standard deviation and the mean of a normal distribution, in order to sample data in the latent space. The decoder then uses the sample to reconstruct the original data. We can then generate synthetic data by using the decoder on randomized samples.

This model have proven efficiency in data generation for images, by using architectures based on 2D-Convolutional networks. In our work we try to adapt the model for financial time series generation.

We tried two different alternatives. In both our models, we use a symmetric architecture for the encoder and the decoder. We first use successive linear layers to encode and decode our data. We then replace the linear layers with 1D-Convolutional layers [2] in order to exploit the importance of the proximity in successive values for a time series.

3.3 PCA GAN

The generation of time series data presents a complex puzzle within the realm of machine learning. Traditional methods often falter in capturing the nuanced patterns and dependencies inherent in sequential data. However, recent advancements, notably diffusion models [3] and Generative Adversarial Networks (GANs) like TimeGAN [4] mentioned before, show promise in addressing this challenge.

These cutting-edge approaches employ sophisticated techniques to discern the underlying distribution of time series data and produce realistic samples. Nonetheless, their implementation comes at a cost. The computational demands are substantial, requiring ample processing power and time. Furthermore, grasping the intricate distribution of time series data proves to be a formidable task, necessitating extensive model training and parameter adjustment.

This section delves into a new methodology to generate Time Series at a low computation cost, using principal component analysis.

3.3.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in data analysis and machine learning. Its goal is to reduce the dimensionality of a dataset while preserving as much of its variance as possible. PCA achieves this by identifying the directions (principal components) along which the data varies the most.

How PCA Works

1. **Data Standardization**: PCA typically begins with the standardization of the data to ensure that all variables are on the same scale. This step involves subtracting the mean and dividing by the standard deviation for each feature.

2. **Covariance Matrix Calculation**: Next, PCA computes the covariance matrix of the standardized data. The covariance matrix summarizes the relationships between the different features in the dataset.

3. **Eigenvalue Decomposition**: PCA then performs eigenvalue decomposition on the covariance matrix to obtain its eigenvectors and eigenvalues. The eigenvectors represent the principal components, and the corresponding eigenvalues indicate the amount of variance explained by each principal component.

4. **Principal Component Selection**: PCA selects a subset of the eigenvectors (principal components) based on their corresponding eigenvalues. Typically, the principal components are ranked in descending order of eigenvalues, and only the top components that capture the majority of the variance are retained.

5. **Projection**: Finally, PCA projects the original data onto the selected principal components to obtain a lower-dimensional representation of the dataset.

PCA Projection of Time Series Slices

To project the PCA of all the slices of a time series into 2D, you would follow these steps:

1. **Slice Time Series**: Divide the time series into slices, such as consecutive windows of fixed length.

2. **Apply PCA**: Apply PCA separately to each slice of the time series data. This results in a set of principal components for each slice.

3. **Aggregate Principal Components**: Aggregate the principal components obtained from all slices into a single dataset.

4. **Project to 2D**: Finally, apply PCA again to the aggregated principal components, but this time, project them onto the two principal components that capture the most variance. This yields a 2D representation of the PCA of all the slices of the time series.

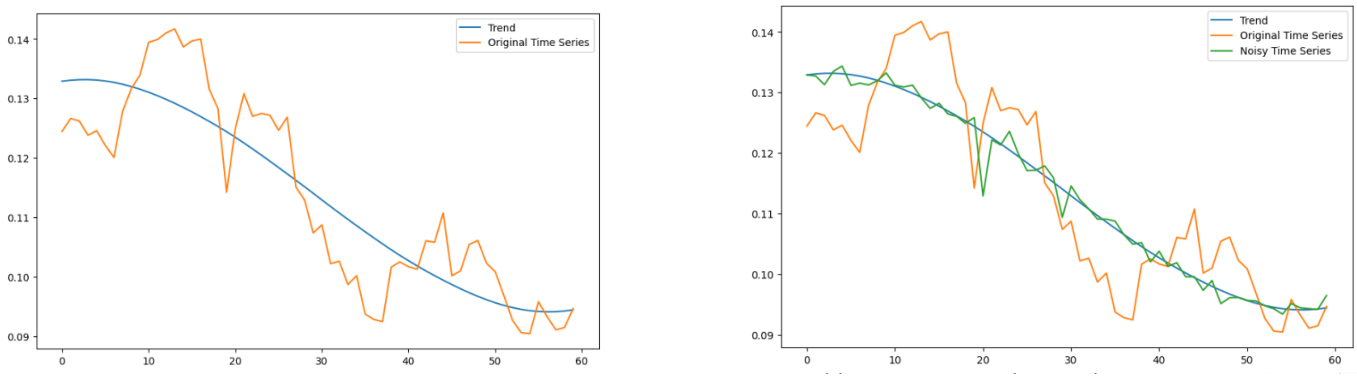
By projecting the PCA of all the slices of the time series into 2D, we can visualize the overall structure and patterns present in the time series data in a lower-dimensional space, facilitating easier interpretation and analysis. This method is widely used to asses the quality of the generated Time Series, by projecting in 2D the PCA of the original data and synthetic data and making sure these data points overlap, we have a visual qualitative metric of success.

3.3.2 Time Series recovery from PCA

When dealing with time series generation, ideally, the PCA points from the generated data should overlap with the PCA points of the original data. Having good principal components is necessary for successful time series generation, but is it sufficient? In this section, we attempt to generate time series from PCA points to investigate whether assuming we can generate some PCA points that perfectly match the original distribution enables us to recreate realistic time series.

PCA inverse. Our method is fairly simple, from a PCA point we apply the inverse transformation of the PCA projection. This inverse transformation returns us a smooth TS, we call it the trend TS. On figure 3a we can see a random TS from our dataset and the inverse transformation of the pca point associated with this time series.

Add noise. Once that we have a trend, we need to add some noise to recreate the properties of original data. A first idea is to sequentially add noise to the trend following the lag1 return distribution of the original data (4). In our pursuit of generating optimal noise, we experimented with using the first three lags of the returns instead of lag 1 only: this allows the TS to deviate more from the trend and capture some longer lag correlation properties. Additionally, we explored incorporating a mean-reverting stochastic process to model the noise added to the Trend. While this



(a) Original TS vs Trend

(b) Noisy TS vs Trend and Original TS

Figure 3: Overall Caption for Both Images

approach improved the visual fidelity of our generated sequences, it unfortunately led to a decrease in performance as measured by the discriminative score and t-SNE.

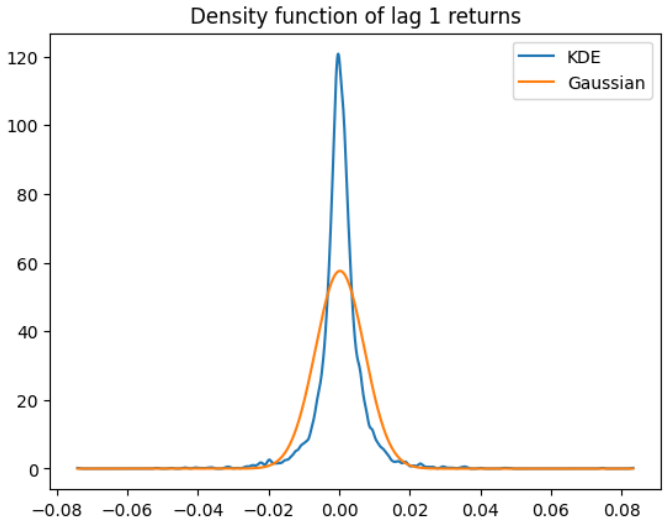
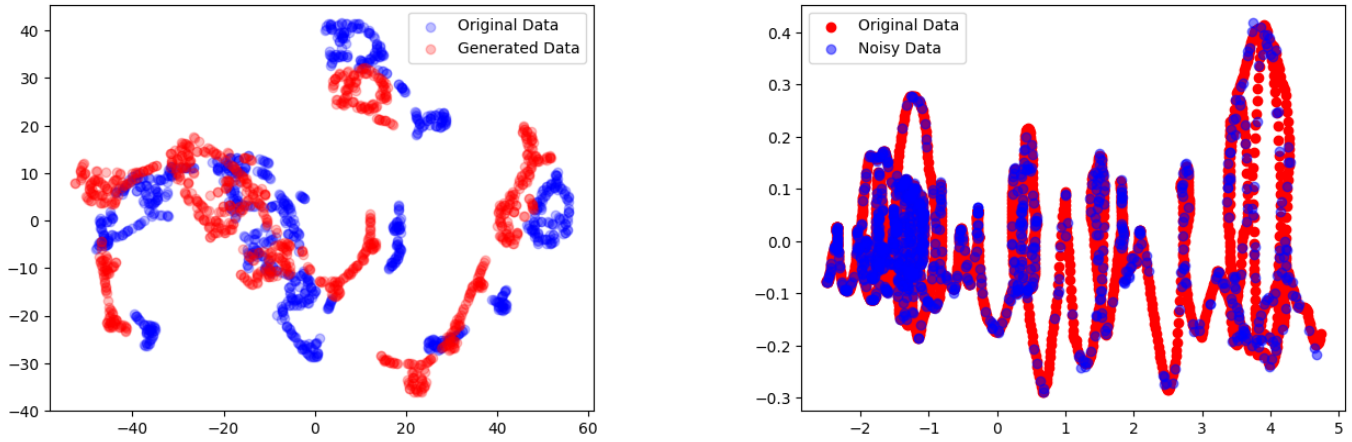


Figure 4: Empirical vs. Gaussian, same mean and standard deviation

Following our methodology of incorporating ideal noise, we have achieved promising outcomes. Our approach demonstrates a discriminative score of 0.011, significantly outperforming the DiffTime model, which yielded a score of 0.05. Furthermore, the t-SNE plot (5a) reinforces our approach, revealing a notable overlap between the clusters of original and generated data same goes with the PCA plot (5b). Here, data used are 60 data points slicing windows, normalised, from google daily close prices.

It is important to keep in mind that here, the recovered noisy data are created based on ORIGINAL DATA PCA points. The results here are outstanding for TS generation, but there is here no machine learning model involved in the process. Given these successful outcomes, our next step involves leveraging GANs to generate the 2D PCA distribution. Ideally, a perfect model should yields results similar to the ones we presented here, and thus be a state of the art TS generation method. This advancement will be detailed in the subsequent section.



(a) Noisy data vs original data model t-SNE

(b) original and noisy data PCA

Figure 5: PCA and t-SNE for recovered noisy TS

3.3.3 PCA-GAN

We first attempted to generate PCA components using a GAN. After training the model, we generated a new batch of 1000 synthetic samples to evaluate the effectiveness of our model. As the preliminary results were not beating our benchmarks, in our second attempt, we adopted a segmented approach by dividing the generation process into four distinct parts, each handled by a separate GAN. This strategy was designed to address different ranges of PCA data distributions effectively. We started by categorizing the PCA data into four groups based on their value ranges and created separate datasets for each category. This segmentation allowed for more tailored training for each GAN, ensuring that each model could specialize in a specific segment of the data distribution. Each GAN was then trained on its respective dataset, optimizing it to generate data that closely resembles the characteristics of that segment. By specializing the GANs, we aimed to enhance the quality and accuracy of the generated data, as each model could focus on learning the nuances of a narrower range of data.

3.3.4 Discriminative Score

When evaluating our models, we used an implementation of the discriminative score similarily "Time-series Generative Adversarial Networks" by Yoon et al. [4], by using a Discriminator neural network tasked with classifying sequences as real or synthetic. This neural network is structured with two layers: a hidden layer employing ReLU activation to introduce non-linearity and an output layer that uses a sigmoid activation to provide a probability indicating the likelihood of the input being real.

The training process involves dividing the real and synthetic datasets into training and testing subsets. During each training iteration, we select random batches from these subsets and compute the Binary Cross-Entropy (BCE) Loss, which quantifies the classifier's performance on distinguishing between real and synthetic samples. Adjustments to the Discriminator's weights are made via backpropagation using the Adam optimizer to minimize the BCE loss.

The effectiveness of the Discriminator is then evaluated on a separate testing set. We measure its accuracy in classifying the data and calculate the discriminative score as the absolute deviation of this accuracy from 0.5. A lower discriminative score, approaching 0, indicates that the Discriminator finds it challenging to differentiate between the real and generated sequences, suggesting that the synthetic data closely mimics the real data in capturing the essential statistical properties and temporal dynamics of the financial time series. This method provides a robust measure of the generative model's performance in producing realistic and statistically consistent sequences.

3.4 Consistency Model

In the past years, Generative Models have quickly evolved and Diffusion Models have emerged as one of the best options when it comes to image generation while also being greatly studied for time series generation and forecasting ([5],[6], [7]). However, Diffusion models are known to lack controllability and have computationally expensive training and inference requirements.

Consistency Models, a novel class of generative models developed in 2023 by Song et al. [8], are addressing some of these issues. A key feature of diffusion models is the iterative sampling process removing noise from random initial vectors. This usually leads to greater quality in generated output but a considerably larger complexity (10–2,000 times [8]). This trade-off leads to diffusion models being only reserved for applications with extensive computing resources. Consistency models are addressing this trade-off.

Consistency models satisfy a notable property, namely, self-consistency: *points on the same trajectory map to the same initial point.* Given a Probability Flow (PF) ODE that smoothly converts data to noise, we learn to map any point (e.g., x_t, x_{t_1} , and x_T) on the ODE trajectory to its origin (e.g., x_0) for generative modeling (Figure 6), i.e., their outputs are trained to be consistent for points on the same trajectory. [8]

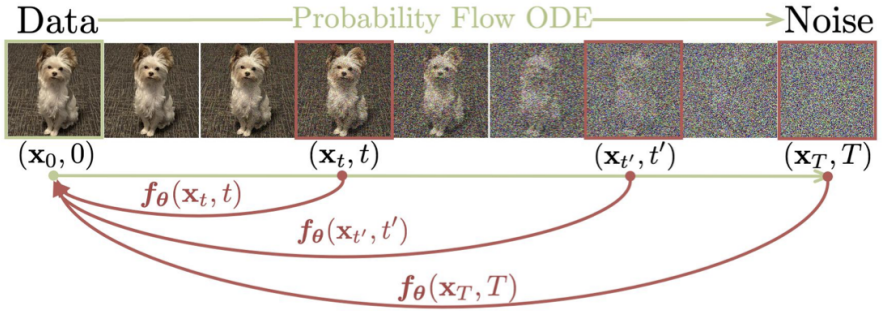


Figure 6: Consistency Model mapping process [8]

Consistency models could thus be highly effective at time series generation due to their ability to capture sequential dependencies and long-range patterns allowing conditional generation of future steps based on past observations. A major advantage can also be the coherent sequential data generation.

To implement our model, we will utilise the architecture described in Ho et al.[9] such that the neural network architecture follows the backbone of PixelCNN++ [10], which is a U-Net [11] based on a Wide ResNet [12]. Now, a big challenge is to create an efficient architecture that takes as input time series. As our base implementation, we will follow the Autoregressive Denoising Diffusion Model developed by Rasul et al. [6] for which will adapt the mapping process developed by Song et al. to map data to a noise distribution (the prior) with an Itô SDE, and reverse this SDE for generative modeling (Figure 7). We can also reverse the associated probability flow ODE, which yields a deterministic process that samples from the same distribution as the Itô SDE. [13].

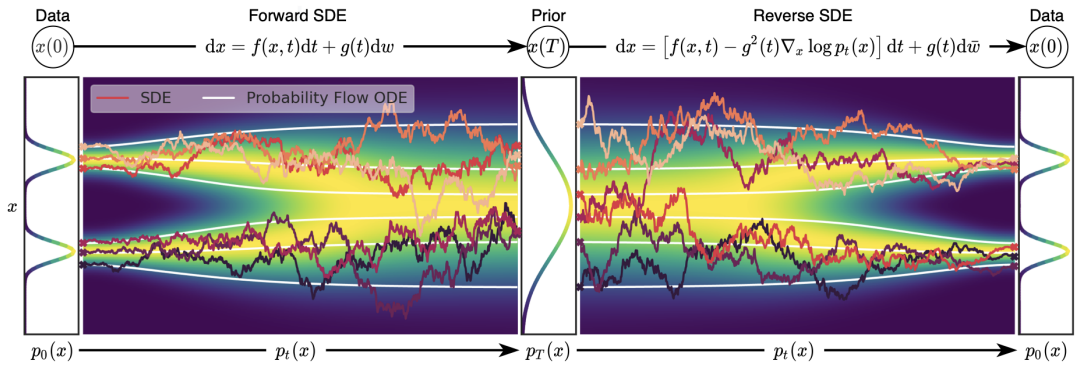


Figure 7: Mapping of noise distribution developed by Song et al. [13]

4 Results

4.1 TimeGAN

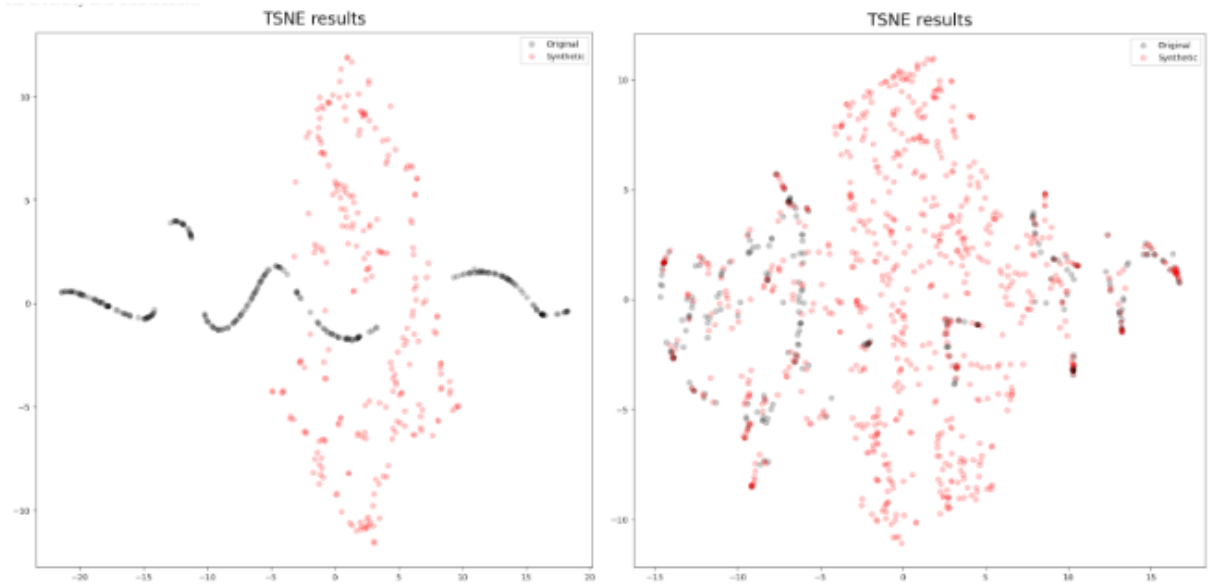
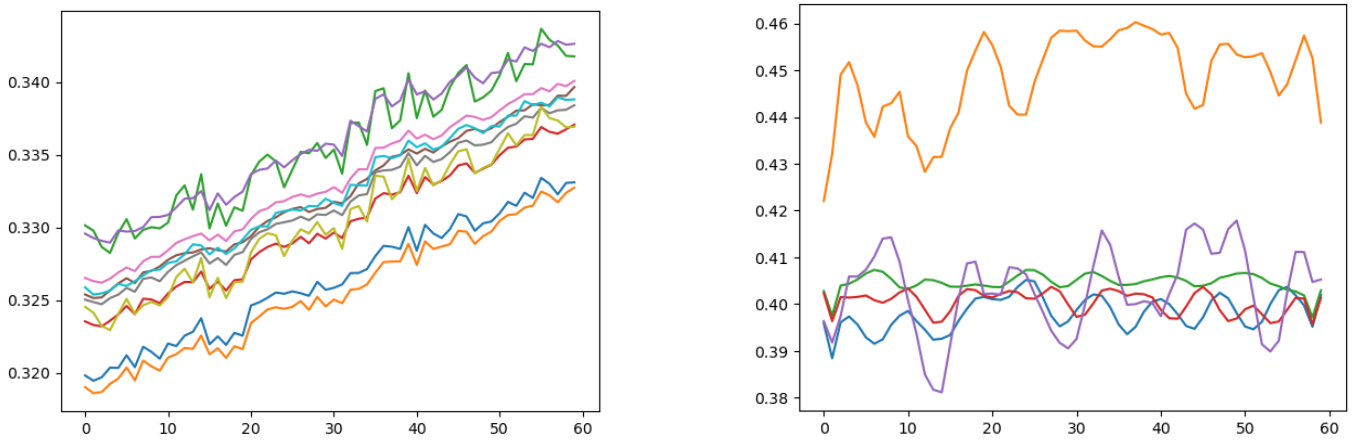


Figure 8: TimeGAN Sample Variance Analysis and Comparison [1]

On the right, we have the t-SNE plot of the samples and original data without PCA; on the right, we have the t-SNE using PCA. PCA allows for samples that capture the original data variance better, as there is more overlap in the points. It also reduced the model's training time, decreasing from 2.5 hours to roughly 1.5 hours.

4.2 VAE



(a) Multi-linear VAE

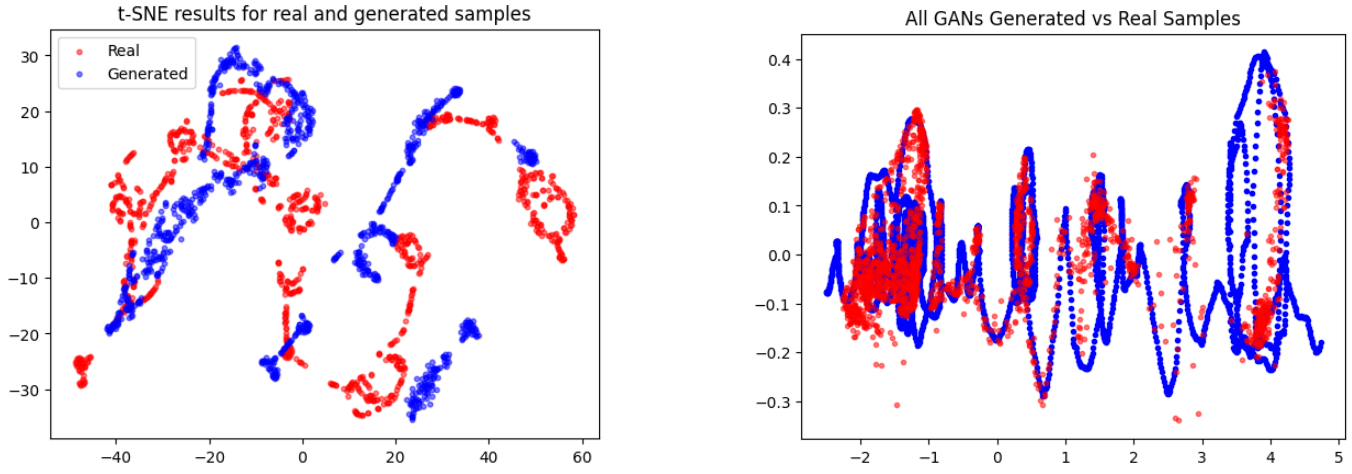
(b) 1D-Convolutional VAE

Figure 9: Generated time series using VAE models

We can see that both our VAE models over-fit. We manage to obtain realistic time series but our dataset has series with various trends and values that ranges from 0 to 1. Our generated series are following a unique trend and have a way lower range. Thus the VAE architectures we use don't manage to capture the time series features needed for their replication. It over-fit towards a time series that lower the reconstruction error for the whole dataset.

4.3 PCA GAN

Figure 10 depicts the segmented GAN results, with collective results from all four GANs, plotted against the real PCA data. The plot demonstrates a substantial overlap between the generated and real samples, indicating that our GANs have successfully learned to mimic the underlying distribution of the PCA data across different segments. With our segmented GAN approach, we achieved a Mean discriminative score of 0.0967, with a standard deviation of 0.0228.



(a) Generated data vs original data model t-SNE

(b) original and generated data PCA

Figure 10: PCA Generation with segmented GANs

Model	Discriminative Score
DiffTime	0.050 ± 0.017
PCA-GAN (Ours)	0.0967 ± 0.0228
TimeGan	0.102 ± 0.021
WaveGAN	0.217 ± 0.022
Diff2 (Capstone)	0.38 ± 0.013
C-RNN-GAN	0.399 ± 0.028

Table 1: Discriminative Scores for Various Generative Models

As shown in Table 1, we are currently beating all benchmarks except DiffTime. It is important to note that as we saw in section 3.3.2, with an ideal PCA component generation model, we could theoretically also beat the DiffTime benchmark, as we could achieve a discriminative score of 0.011 with our recovery method. On another note, this 2D method is also less computationally intensive that other models from the literature, which is a considerable advantage.

4.4 Consistency Model

So far we were not able to produce an effective consistency model for time series generation. However, we were able to produce satisfying results for image generation on the MNIST dataset (Figure 11) with limited computing power.

Figure 11: Consistency model trained on MNIST [14]

Following the training on MNIST, we also tried to train and fine-tune our model for CIFAR10 [15]. With limited computing power and about 6hrs of training on a A100 GPU with hihg-ram, we were able to train on 100 epochs with learning rate of 0.001 and got the following results (Figure 12):

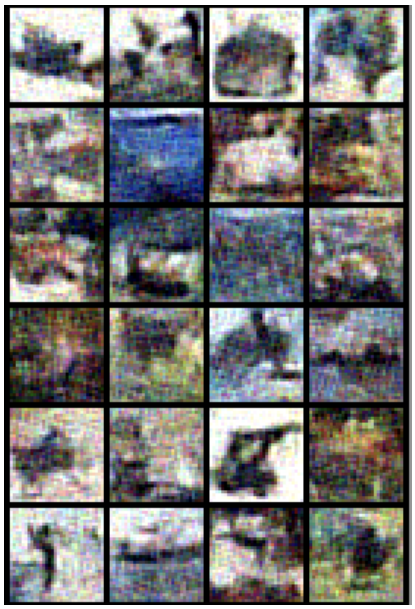


Figure 12: Consistency model trained on CIFAR10 [15]

As seen in Figure 12, we seen that the denoising process is not complete but the image generation process seems to be promising. One of the main issues with diffusion models is the computing power required for training. Here seem to be faced with a similar issue and conjecture that increasing the number of epochs to 300-1,000 would significantly increase our accuracy while training on CIFAR10.

5 Discussions

Our results show that our PCA-GAN was one of our more promising models due to the similarity between synthetic and real data. While, TimeGAN should've have been better, it was limited due to his high computational requirements. If we had more time, it would have been interesting to see how the consistency model would have fared since it can produce coherent images. However, converting this model for tabular data would have taken a long time to do.

6 Conclusion and Future Work

During this project, we explored the application of different generative models for time series generation, including the TimeGAN, VAE, PCA GAN, and Consistency model. With the results above, we showed that the recently developed neural networks architecture can be effectively used in an adversarial modeling framework to generate predicted financial time series in discrete-time. Although we have tried our best to train and tune these models, our results failed on outperforming from the well-known models in generating realistic and diverse time series data. So, there is a large space to work on and there are several directions for future work to further improve the quality and applicability of our models.

One promising direction for further work is to integrate Diffusion models into our framework. Diffusion models have indicated remarkable capabilities in generating high quality financial tabular data [3] as well as time series [16]. The high computational cost of the iterative denoising process would be the limitation while enhancing the diversity and fidelity of the generated data. Another direction would be to seek for advancement in Generative Adversarial Networks (GANs). Start from an early model generating financial time series, Quant GANs [17], to a recent application of GANs on generating real-looking synthetic Limit Order Book (LOB) data for simulating stock market dynamics [18], suggested that GANs have the potential to capture the underlying structure of data effectively. By addressing these challenges and opportunities, the integration of generative AI models can be advanced for generating realistic

financial time series data.

References

[1] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper$_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490 - Paper.pdf$

[2] D. Kavran, B. Žalik, and N. Luka, "Time series augmentation based on beta-vae to improve classification performance," in *International Conference on Agents and Artificial Intelligence*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246920404

[3] T. Sattarov, M. Schreyer, and D. Borth, "Findiff: Diffusion models for financial tabular data generation," *Proceedings of the Fourth ACM International Conference on AI in Finance*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:261531030

[4] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," 2019.

[5] Y. Li, X. Lu, Y. Wang, and D. Dou, "Generative time series forecasting with diffusion, denoise, and disentanglement," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 23 009–23 022. [Online]. Available: https://proceedings.neurips.cc/paper$_files/paper/2022/file/91a85f3fb8f570e6be52b333b5ab017a - Paper - Conference.pdf$

[6] K. Rasul, C. Seward, I. Schuster, and R. Vollgraf, "Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8857–8868. [Online]. Available: https://proceedings.mlr.press/v139/rasul21a.html

[7] D. Daiya, M. Yadav, and H. S. Rao, "Diffstock: Probabilistic relational stock market predictions using diffusion models," in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 7335–7339.

[8] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, "Consistency models," 2023.

[9] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[10] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," 2017.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[12] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2017.

[13] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," 2021.

[14] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[16] M. Kollovieh, A. F. Ansari, M. Bohlke-Schneider, J. Zschiegner, H. Wang, and Y. Wang, "Predict, refine, synthesize: Self-guiding diffusion models for probabilistic time series forecasting," *ArXiv*, vol. abs/2307.11494, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260091522

[17] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, "Quant gans: deep generation of financial time series," *Quantitative Finance*, vol. 20, pp. 1419 – 1440, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:196831870

[18] B. Labiad, A. Berrado, and L. Benabbou, "Generative adversarial neural networks for realistic stock market simulations," *International Journal of Advanced Computer Science and Applications*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:268820667