EasyChair Preprint

№ 12798

Enhanced Financial Data Generation using Recurrent Neural Networks within Variational Autoencoders: a Sector-Based Analysis

Parth Garg, Pulkit Sharma and U M Prakash

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

March 28, 2024

Enhanced Financial Data Generation using Recurrent Neural Networks within Variational Autoencoders: A Sector-based Analysis

PARTH GARG
Department of Computing Technologies, School of Computing, SRM Institute of Science and Technology, Chennai, India
pg6945@srmist.edu.in

PULKIT SHARMA
Department of Computing Technologies, School of Computing, SRM Institute of Science and Technology, Chennai, India
ps3066@srmist.edu.in

Prakash U M
Department of Computing Technologies, School of Computing, SRM Institute of Science and Technology, Chennai, India
prakashm3@srmist.edu.in

***Abstract*—This study proposes a new generative model for quarterly financial data time series that is based on a variational autoencoder (VAE). The program provides financial data for a synthetic firm in a certain industry throughout a year, comprising four quarters of data. This technology employs a recurrent neural network-based VAE to successfully capture both multivariate distributions and temporal dependencies in the data. Compared to classic models such as the Multivariate Normal Monte Carlo Model and the Multivariate Gaussian State Space Model, our model's synthetic samples are more realistic, with higher visual fidelity and lower discriminative scores. In addition to the basic model, the derivative model has a conditional channel that generates samples with preset future performance. The user-friendly interface of our product makes it simple to utilize. Analyzing the patterns and features of synthetic samples can reveal significant information on alpha factor trading and risk management measures. Furthermore, applying the model to various datasets has the potential to improve these findings even more.**

Keywords—Generative model, Variational autoencoder ,Financial data simulation, Time series analysis ,Recurrent neural network (RNN)

I. Introduction

Analysing the time series of fundamental company data is essential for internal and external analytics, as well as traders and risk managers. In spite of the useful information it provides, such data are often limited, such as data scarcity and confidentiality concerns. The creation of synthetic financial fundamental data sequences with similar statistical properties is an effective way to address these gaps and weaknesses in actual data. These synthetic data can serve a number of purposes, such as alleviating the lack of training data, enabling the development of data-dependent products and services, and revealing hidden characteristics that offer valuable market insights.

Traditional methods for generating synthetic data usually rely on stochastic sampling based on known distributions, the Monte Carlo method is a well-known example. Although these methods are simple and interpretable, they often do not capture the intricacies of real-world data complexity. On the other hand, deep learning-based generative models possess strong unsupervised learning capabilities and demonstrate exceptional ability to generate complex entities, such as images, videos, text, and music. These models use deep neural networks to learn patterns and structures within training data. This knowledge can then be used to produce new and realistic data samples. Generative Adversarial Networks (GANs),Variational Autoencoders (VAEs), and autoregressive models such as Transformers are examples of deep-learning generative models with different frameworks and mechanisms, resulting in outputs with distinct characteristics.

Synthetic data production for time-series data is challenging owing to the temporal patterns in the data. A generating process must account for multivariate distributions and temporal interactions between features. Recurrent Neural Network (RNN), a specialized Neural Network architecture designed for sequential data processing, has emerged as a strong contender. RNNs, unlike typical neural networks, have an internal memory that maintains information from past steps, enabling them to successfully capture temporal correlations and patterns. As a result, RNNs are quite useful for time series analysis. Our new model uses Variational Auto-Encoders (VAEs) and a Recurrent Neural Network (RNN) architecture to simulate quarterly financial data sequences for a fictitious company. Quarterly financial data collected over a one-year period demonstrate higher fidelity when compared to baseline models using traditional methodologies, as evaluated qualitatively and statistically in our trials. We expand our model by using next-quarter return performance as a conditional channel, yielding intriguing results. We matched our model with historical financial data from real-world SP500 companies and created a user interface for our solution.

II. Related Work

The use of deep generative models with time series data is a relatively recent subject. Creating time series data has many practical applications. Time-series data, which consists of a succession of observations gathered at regular intervals throughout time, is essentially three-dimensional. Current approaches aim to identify the best combinations of advanced network architectures, including Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN), capable of processing this three-dimensional data, and deep learning-based generative models, primarily focused on Generative adversarial networks (GANs).

TABLE I. lists generative models and associated network types from the literature.

Model / Network	CNN	RNN
GAN	C-RNN-GAN (Mogren, 2016), WaveGAN(Donahue et al., 2019)	C-RNN-GAN (Mogren, 2016), RCGAN (Esteban et al., 2017), Time GAN (Yoon et al., 2019)
VAE	TimeVAE (Desai et al. 2021)	Recurrent VAE (Fabius et al., 2014) VRNN(Chu g et al., 2015)

In 2014, Fabius et al. developed a sequential-specific Variational Autoencoder (VAE). Their model incorporates RNNs to enhance the VAE architecture. The encoder comprises recurrent connections and uses the RNN's final encoded state to represent the input sequence.

The approach is effective at learning latent representations and creating organized sequences. Fabius et al. demonstrate this by experimenting with MIDI data for music production [5].

Chung et al. (2015) introduced the Variational Recurrent Neural Network (VRNN) model. This novel approach incorporates a VAE into each time step. The VAEs are unique in that they are conditioned on the prior RNN state variable ht−1. Unlike a normal VAE with a multivariate Gaussian prior, the VRNN allows for an isotropic Gaussian distribution where the mean and variance are related to the preceding state. The VAE's strategic extension captures temporal intricacies in sequential data, ensuring the model understands its underlying structure [1].

Mogren (2016) introduced the C-RNN-GAN model, which is ideal for generating classical music data. In this approach, deep Long Short-Term Memory (LSTM) networks are used. These networks take randomly generated noise as input and process it to learn informative patterns. The learned patterns are then passed through fully connected layers to generate new sequential data samples.. In an adversarial training setting, a discriminator assesses both generated and real music samples. This method allows for the construction of diverse musical compositions with different tonal complexity and intensity [8].

Esteban et al. (2017) created the Recurrent Conditional GAN (RCGAN), which focuses on multidimensional time-series data creation. RCGAN uses RNNs for both generator and discriminator functions, as well as a conditioning mechanism to direct the generation process. By conditioning on auxiliary information, RCGAN generates time series data with specified properties while retaining temporal coherence. This element is useful for generating privacy-preserving eICU data [4].

Donahue et al. (2019) introduced WaveGAN, a model for generating raw audio waveforms with GANs. WaveGAN's generator uses CNNs in both the generator and the discriminator to transform random noise into audio waveforms using transposed convolutions. At the same time, the discriminator distinguishes between genuine and produced sounds. This architectural method allows for the creation of high-fidelity audio samples by directly collecting complicated waveform elements [3].

TimeGAN (Yoon et al., 2019) is widely regarded as the most advanced model for generating time-series data. Yoon et al. (2019) propose a novel framework that combines the freedom of unsupervised learning with the control of supervised training. TimeGAN ensures that the dynamics of the training data are maintained during sampling by concurrently optimizing a learnt embedding space with supervised and adversarial objectives. The architecture is made up of four main components: an embedding function, a recovery function, a sequence generator, and a sequence discriminator. TimeGAN's technique captures feature encoding, representation creation, and temporal evolution simultaneously [9].

A more recent technique, TimeVAE (Desai et al., 2021), uses CNN encoders and decoders with specified trend-representing blocks to balance complexity and compatibility [2]. This model uses a generative distribution with the form $p\theta(X \mid z) = N(\mu,I)$. Initially considered for adaptation, an analysis of TimeVAE's code indicated that its generative method resembled reconstruction rather than direct sampling from N(0,I). This departure called into question the model's assumptions, which we eventually abandoned.

In our proposed model, we draw inspiration from Recurrent VAE (Fabius et al., 2014) [5], but we enhance it with a deeper network topology that includes two layers of RNNs. While the Recurrent VAE has found applications in music composition and text/speech production tasks, we believe our model is a unique technique to producing time series data in economics and finance.

III. Methodology

In this part, we describe the approach used to create and build a generative model for synthetic financial data using a variational autoencoder (VAE) and a recurrent neural network (RNN) architecture. The technique begins with a review of VAEs, covering its underlying concepts and aims, and then goes on to explain the model architecture and

training procedure in depth. We also cover the expansion of the VAE architecture to include conditional generation tasks via Conditional Variational Autoencoders (CVAEs) and the construction of the loss function to maximize model performance.

A. VAE

1)Overview

Variational Autoencoders (VAEs), first presented by Kingma and Welling in their 2013 publication "Auto-Encoding Variational Bayes," are a strong generative modeling system that combines autoencoder and probabilistic modeling ideas. Unlike typical autoencoders, which provide point estimates, the VAE encoder produces the distribution of the latent embedding, allowing for the modeling of complicated data distributions and exploration of latent space [7].

The main objective of a Variational Autoencoder (VAE) is to produce samples that accurately capture the statistical distributions inherent in the original dataset, which comprises N independent and identically distributed (i.i.d.) continuous or discrete variables denoted as x. This entails describing the unknown probability function $p_\theta(x)$. VAE operates under the assumption that the observed data is generated by a latent vector that is independent of the data itself. The generative process unfolds in two stages. Firstly, a prior distribution $p_\theta(z)$ is employed to generate the latent variable z, which is then utilized alongside a generating distribution $p_\theta(x \mid z)$. to construct the data point x .Though the specific expressions and probabilities of the aforementioned $p_\theta(z)$ and $p_\theta(x \mid z)$. may not be explicitly known, we operate under the assumption that they exhibit differentiability concerning both the parameters θ and the latent variable z.

The prior $p_\theta(z)$ is often unclear. To solve this, the VAE encoder introduces an approximation of the posterior distribution: $q_\phi(z) \approx p_\theta(z \mid x)$. The encoder models the posterior probability distribution $q_\phi(z)$, while the decoder models the conditional probability $p_\theta(x \mid z)$.During training, the posterior is regularized to fit this chosen prior. This regularization combines a posterior variance approximation of the Kullback-Leibler (KL) divergence with a predetermined prior loss function. Using a predetermined prior enables for simple direct sampling of a concealed representation of the prior distribution. The decoder can generate synthetic data distributions that embody the statistical properties of the original data set from the latent representation. In this way, VAEs allow the exploration of latent space and the generation of new data samples that conform to the data distribution.

In our case, we assume that the prior distribution is a standard isotropic multivariate Gaussian $p_\theta(\mathbf{z}) = \mathrm{N}(0,\mathbf{I})$, and the inferred posterior distribution is a multivariate Gaussian with diagonal covariance $q_\phi(\mathbf{z}|\mathbf{X}) = \mathrm{N}(\mu_\phi(\mathbf{X}),\mathrm{diag}(\sigma_\phi^2(\mathbf{X}))$. The encoder takes $\mathbf{X}$ as input and outputs two vectors $\mu_\phi(\mathbf{X})$ and $\sigma_\phi^2(\mathbf{X})$ in R^d, where d is the dimensionality of the latent space. We let the output distribution be an isotropic multivariate Gaussian, $p_\theta(\mathbf{X}|\mathbf{z}) = \mathrm{N}(\mu_\theta,\mathrm{diag}(\sigma_\theta^2(\mathbf{z})))$. The decoder takes $\mathbf{z}$ as input and outputs the vectors μ_θ and $\sigma_\theta^2(\mathbf{z})$ in R^s. We also introduce a lower bound for $\sigma_\theta^2(\mathbf{z})$, ensuring that the scale for all variables must exceed a certain threshold to avoid an overly narrow output distribution.

In this scenario, we make the assumption that the prior distribution follows a normal isotropic multivariate Gaussian, denoted as $p_\theta(\mathbf{z}) = \mathrm{N}(0,\mathbf{I})$, while the inferred posterior distribution is a multivariate Gaussian with diagonal covariance, expressed as $q_\phi(\mathbf{z}|\mathbf{X}) = \mathrm{N}(\mu_\phi(\mathbf{X}),\mathrm{diag}(\sigma_\phi^2(\mathbf{X}))$. The encoder takes $\mathbf{X}$ as input and generates two vectors $\mu_\phi(\mathbf{X})$ and $\sigma_\phi^2(\mathbf{X})$ in R^d, where d represents the dimensionality of the latent space. We assume an isotropic multivariate Gaussian distribution for the decoder, denoted as $p_\theta(\mathbf{X}|\mathbf{z}) = \mathrm{N}(\mu_\theta,\mathrm{diag}(\sigma_\theta^2(\mathbf{z})))$. The decoder accepts z as input and outputs the μ_θ and $\sigma_\theta^2(\mathbf{z})$ in R^s .To prevent an excessively narrow output distribution, we enforce a lower bound on $\sigma_\theta^2(\mathbf{z})$, ensuring that the scale for all variables remains above a certain threshold.

2) Conditional VAE

Conditional Variational Autoencoders (CVAEs) extend the basic VAE framework to address conditional generation tasks where both input data and additional conditions are incorporated into the generation process. CVAEs are particularly useful in scenarios where the generated data is influenced by specific attributes or contexts, enabling controlled and targeted data synthesis.

In a CVAE, the latent representation z is conditioned not only on the input data x, but also on additional conditioning variables y representing the specific conditions. The prior distribution becomes $p_\theta(z \mid y)$, where y represents the additional conditions. The posterior and generative distributions become $q_\phi(z \mid x,y)$ and $p_\theta(x \mid z,y)$. The CVAE loss function extends the original VAE loss by including the conditioning variables in both the prior and posterior distributions.

3) Loss Function

The CVAE loss function, also known as the obvious inferior loss function (ELBO), can be written as.:

$$L_{\theta,\phi} = -E_{q_\theta} \log p_\phi(\mathbf{X} \mid \mathbf{z}) + D_{KL}(q_\theta(\mathbf{z} \mid \mathbf{X}) \parallel p_\phi(\mathbf{z})) \quad (1)$$

$$L_{\theta,\phi} = -E_{q_\theta} \log p_\phi(\mathbf{X} \mid \mathbf{z},\mathbf{y}) + D_{KL}(q_\theta(\mathbf{z} \mid \mathbf{X},\mathbf{y}) \| p_\phi(\mathbf{z} \mid \mathbf{y})) \quad (2)$$

Our training loss function follows the previously defined ELBO loss function with one modification. We use a β weight on the reconstruction error as a hyperparameter, which serves to increase or decrease the emphasis on the reconstruction loss relative to the KL divergence loss between the approximated posterior $q_{(z|X)}$ and the prior $p_{(z)}$. This variation is also known as the beta-VAE (Higgins et al., 2016) [6].

$$L_{\theta,\phi} = -\beta \times E_{q_\theta} \log p_\phi(\mathbf{X} \mid \mathbf{z}) + D_{KL}(q_\theta(\mathbf{z} \mid \mathbf{X}) \| p_\phi(\mathbf{z})) \quad (3)$$

$$L_{\theta,\phi} = -\beta \times E_{q_\theta} \log p_\phi(\mathbf{X} \mid \mathbf{z},\mathbf{y}) + D_{KL}(q_\theta(\mathbf{z} \mid \mathbf{X},\mathbf{y}) \| p_\phi(\mathbf{z} \mid \mathbf{y})) \quad (4)$$

B. *Model Architecture*

The encoder and decoder components are built with a two-layer recurrent neural network (RNN) architecture. The encoder receives a 3-dimensional array with dimensions $s = N \times T \times D$. N denotes the array size, T represents the number of time steps, and D is the number of function dimensions.

The **Encoder** module starts with an initial RNN layer `RNN 1` that processes the input signal and produces hidden state representations. This intermediate output is then fed into a subsequent RNN layer `RNN 2`, which further refines the hidden state information to produce a more compact and meaningful representation. The final hidden state `hidden_n` obtained from `RNN 2` is reshaped to fit the batch size and embedding dimension `embed_dim`, thus serving as the encoded representation of the whole input sequence. The size of the hidden dimension is twice the size of the embedding dimension.

The **Decoder** component mirrors this architecture by first transforming the encoded representation using a linear layer `fc` to prepare it for the decoding process. The modified encoding is then elongated along the time dimension to align with the length of the original sequence. This elongated tensor is then passed through the initial RNN layer, denoted as RNN 3, within the decoder, followed by another RNN layer labeled as RNN 4.The final hidden state output from `RNN 4` is used to predict both the mean and log scale of the generated sample's multivariate distribution.

In the training stage, the encoder transforms the data into two latent representation vectors, acting as parameters for a multivariate Gaussian distribution. The loss is computed by utilizing the encoded parametric distribution of the latent representation and the decoder distribution of the observations conditioned on the latent representation. In the generation process, a latent vector $\mathbf{z}$ is sampled from a standard isotropic multivariate Gaussian \($p_\theta(\mathbf{z})$, concatenated with the conditional channel $\mathbf{y}$ (if present), and subsequently decoded to yield the isotropic multivariate Gaussian distribution $p_\theta(\mathbf{X}|\mathbf{z})$ for the sample. A synthetic data point $\hat{\mathbf{X}}$ is then sampled from the generative distribution.

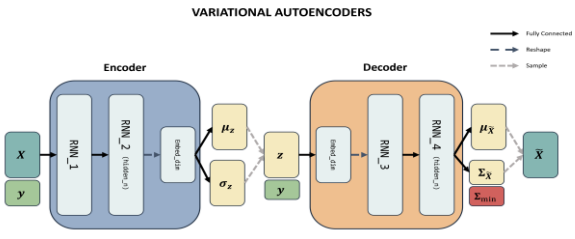

Figure 1:
Model Diagram

C. *Baseline Models*

We have chosen two traditional simulation methods to serve as benchmark baseline models. It's important to note that the Multivariate Gaussian Monte Carlo model (MNS) ignores temporal relationships, while the Multivariate Gaussian State Space model (MGSSN) includes them.

- **Multivariate Gaussian Monte Carlo Model**: This model operates under the assumption that the data adhere to a multivariate Gaussian distribution, characterized by the mean vector (μ) and covariance matrix (Σ). These parameters delineate the properties of the distribution, facilitating the generation of data points through sampling. The process involves the selection of random samples (x) from the Gaussian distribution via Monte Carlo sampling.

- **Multivariate Gaussian State Space Model**: The MGSSN assumes that observed data ($\mathbf{y}$) is derived from an underlying latent state ($\mathbf{x}$) through a linear Gaussian relationship. This model includes two key equations: the state transition equation ($\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{w}_t$) and the observation equation ($\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t$). Here $\mathbf{A}$ is the state transition matrix, $\mathbf{C}$ is the observation matrix, and $\mathbf{w}_t$ and $\mathbf{v}_t$ represent process and observation noise, respectively. The latent state captures the underlying dynamics, while the observation noise accounts for discrepancies between actual observations and model predictions.

IV. Model

Using the methods described above, we created a synthetic financial data generator capable of producing sector-specific synthetic financial data time series for four quarters of a year. This was accomplished by training individual sector-specific models using corresponding real historical data from companies within each respective sector. This process yielded nine fitted VAE (CVAE) models, with each model representing a distinct sector.

In the next experiment, we concentrated on determining the efficacy of one of these models: Consumer, Non-cyclical. We presented cumulative density function charts for both the training and validation datasets to evaluate the calibration. To assess the quality of the generated data, we compared samples generated by our model to those generated by baseline models. Furthermore, we explored whether our reconstruction technique increased feature predictability through denoising by comparing the efficacy of stock selection with original and rebuilt features. We also expanded our model to include the next quarter's return level and a conditional feature. This addon enables for the creation of samples with preset future performance characteristics.

We used Bloomberg to gather five years of historical quarterly financial data from 503 firms across nine sectors of the S&P 500 Index. Yahoo Finance provided historical closing prices for each firm one day before and after the quarter's reporting date, which were then utilized to compute quarterly returns. Every firm was issued a sector label. The distribution of businesses across industries is as follows:

TABLE II. COMPARISO NUMBER OF BUSINESSES IN EACH INDUSTRY

Sector	Number of Companies
Consumer, Non-cyclical	112
Financial	91
Industrial	68
Consumer, Cyclical	63
Technology	59
Communications	34
Utilities	30
Energy	26
Basic Materials	20

For our project, we selected eight fields: Total Debt-to-Total-Capital, Price-to-Book Ratio, Price Earnings Ratio (P/E), Total Asset Growth Rate, Revenue Growth Rate, Return-on-Community-Equity, Return-on-Assessment, and Gross Margin [1]. These fields were chosen for their ability to provide insight into a company's leverage, valuation, growth, profitability, and market relevance as potential factors influencing future stock returns.

Exploratory data analysis revealed weakly correlated, highly skewed variables with many extreme values (see Figure 2). Therefore, several preprocessing steps have been performed to address these characteristics. First, we treated left-skewed variables (Total Debt to Total Capital, Price to Book Ratio, and Price Earnings Ratio (P/E)) by replacing non-positive values with a small positive constant before applying a logarithmic transformation. Second, we used winsorization to replace extreme negative and positive values with the 2nd and 99th percentiles of the corresponding quarter's values given the presence of extreme values in all fields except Price to Book Ratio and Gross Margin. We then partitioned the full dataset into nine separate datasets, each containing companies from a specific industry. We standardized the data in each quarter's cross-section within each separate dataset. For each specific sector and quarter, the values of each field across firms were normalized to have a mean of zero and a standard deviation of one. This standardization process serves two purposes. First, it facilitates deep neural network learning and eliminates market dynamics influence. To allow for future reverse transformations, we recorded the mean and standard deviation time series for each sector. Missing values will be imputed to zeros.

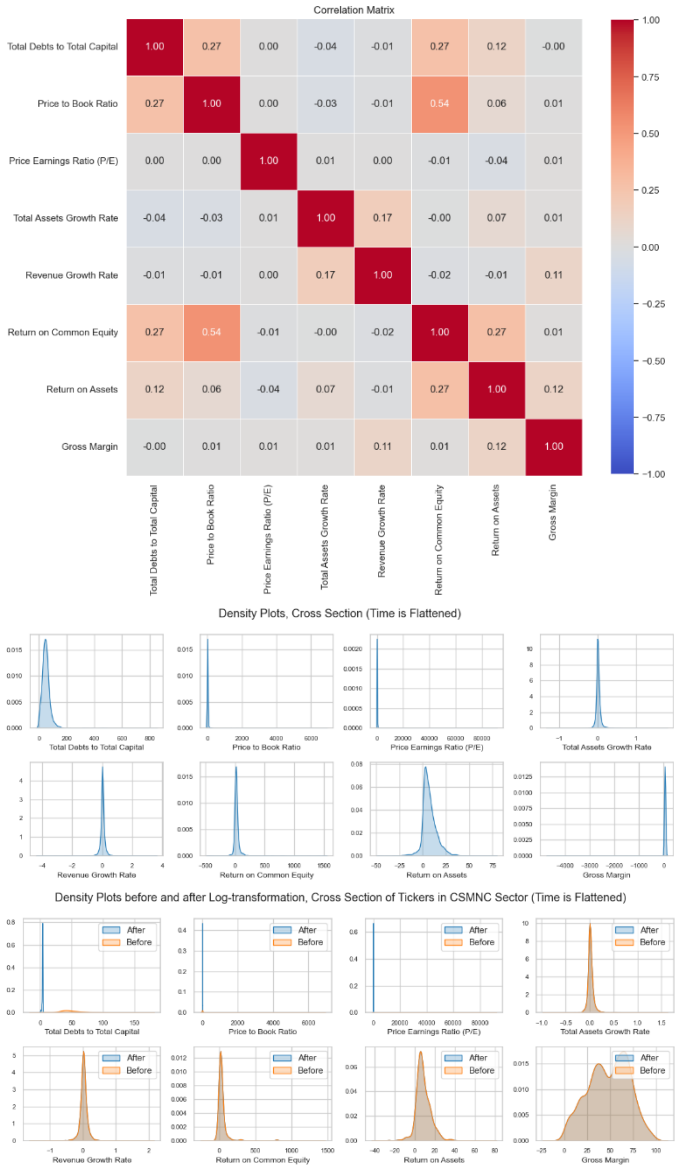

Figure 2: The impact of log transformation on exploratory data analysis.

Each sample in our datasets represents a company's financial time series across four consecutive quarters within a single year ($T = 4$, $D = 8$). An i.i.d. dataset is created by collecting distinct, non-overlapping samples from various firms. The Basic Materials dataset covers five years of data for twenty organizations, resulting in a total sample size of $20 \times 5 = 100$. This strategy has both benefits and drawbacks. While it considerably expands our sample size, it overlooks any potential inter-year relationships within the same organization.

TABLE III. Statistics of the CSMNC dataset before normalization and missing value imputation

	Count	Mean	Std	Min	25%	50%	75%	Max
Log Total Debts to Total Capital	2165	3.69	0.80	0.09	3.52	3.82	4.15	5.10
Log Price to Book Ratio	2151	1.84	1.14	-0.62	1.10	1.67	2.42	8.83
Log Price Earnings Ratio	2178	3.21	0.61	1.47	2.75	3.18	3.56	5.72
Total Assets Growth Rate	2204	1.95	6.39	-29.3	0.89	0.81	3,64	38.3
Revenue Growth Rate	2208	2.01	10.9	-53.6	2.41	1.87	6.71	57.4
Return on Common Equity	2131	35.8	62.1	-46.9	10.6	19.6	35.7	758
Return on Assets	2192	8.44	7.10	-29.4	4.06	7.20	12.4	35.9
Gross Margin	2080	48.2	21.9	-9.47	32.4	49.9	66.0	98.6

A. *Evaluations*

1)Calibration

We evaluate our model's calibration effectiveness using the following method. Our data collection is first separated into two sets: a training set and a test set, with the training set including data from 80% of the organizations. The model trained on the training data produces multivariate distributions. We next proceed with the following stages on both the training and test sets.

1. Using the decoded generative distributions from z, calculate the average CDF (cumulative distribution function) of data for each field and time step.
2. To mimic the distribution of average CDFs, sample a z many times and repeat step 1.

The average CDFs are determined using the equation below.

$$\frac{\sum_{i=1}^{N} \Pr_{p_\theta(x^{s,t}|z)}(a \le x_i^{s,t})}{N}, \; z \sim \mathcal{N}(0, \mathbf{I}) \tag{5}$$

Here, i represents each observed sample in the data sets, and $p_\theta(x^{s,t} \mid z)$ denotes the marginal distribution of the field s at time step t according to the generative distributions. When ideally calibrated, the average cdfs should have a uniform distribution.

Figure 2 shows the histogram and cdf plots representing the distribution of the 1000 simulated average cdfs. The observed pattern suggests that for the majority of features, the distributions of the average cdfs for both the training and test sets closely resemble uniform distributions. This alignment indicates effective calibration of our model and suggests minimal overfitting. However, an interesting observation emerges: in the first and last steps, the average cdfs of some fields cluster around 50% for both the training and testing sets. The reason for this overconfidence in the first and fourth quarter data generation remains unclear. Further investigation is warranted to uncover the reasons.

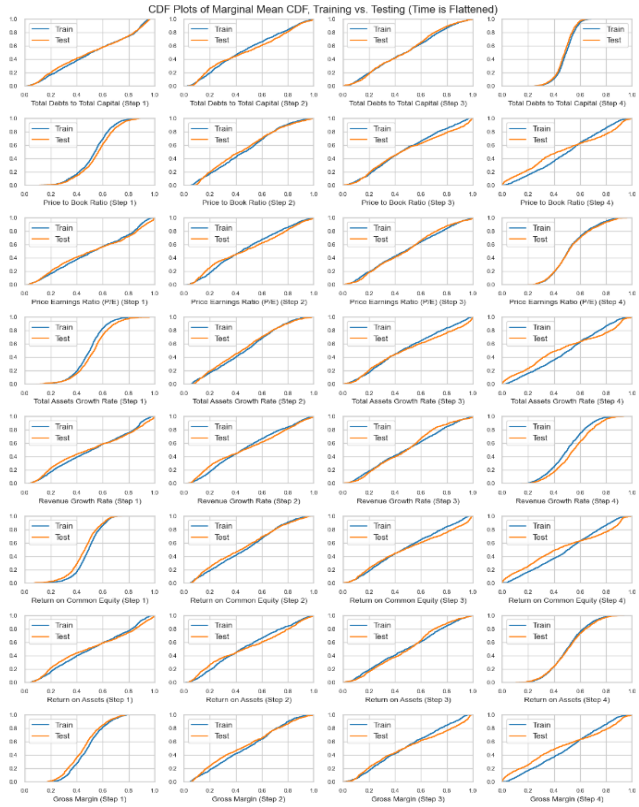

Figure 3: Average cdfs graphs

2)Fidelity

The final model is trained using the entire data set. We compare the fidelity of our model's produced synthetic samples to two baseline models: the Multivariate Gaussian Monte Carlo Model (MNS) and the Multivariate Gaussian State Space Model (MGSSM). There are two parts to the fidelity metrics. First, we examine the visual similarity between the generated synthetic samples and the observed real samples, both in density plots with time-flattened marginal distributions and in TSNE plots with the time dimension reduced to a two-dimensional representation. Second, we apply discriminatory scoring. Specifically, we use supervised learning to build complex classification models to distinguish between actual and synthetic data, with the goal of achieving an accuracy of around 0.5 on the withheld dataset. A number closer to 0 implies greater performance, implying that the created data is closely related to the original data. The three classifiers used are Support Vector Classifier (SVC), Gradient Boosting Machine Classifier (GBM), and Random Forest Classifier (RF).

Figure 3 depicts density plots that demonstrate broad distribution statistics, as well as TSNE plots that indicate temporal trends. Our findings show that our model provides samples with superior visual authenticity than the classic MNS and MGSSM models. The basic models struggle to capture complicated distributions since they both rely on preset Gaussian distributions. In contrast, our model efficiently captures temporal patterns, which is difficult for baseline models. The MNS model overlooks temporal patterns, and MGSSM's simplistic linear Gaussian connection does not correctly capture them, as demonstrated in the TSNE charts. However, it is worth noting that the synthetic data points show less dispersion than the actual observations, implying some degree of overconfidence. Addressing this problem is an area of future research.

In terms of discriminative scores, our model greatly outperforms the baseline models. The VAE model has mean and maximum discriminative scores of 0.33 and 0.37, compared to 0.46 and 0.48 for MGSSN and 0.45 and 0.47 for MNS. The table below shows detailed results. Despite the inherent unpredictability in the sample and generating processes, the overall advantage is clear. The discriminative score, while much better, remains unsatisfactory. The system achieves more than 80% accuracy in discriminating between phony and real data, highlighting the complexities of market modeling.

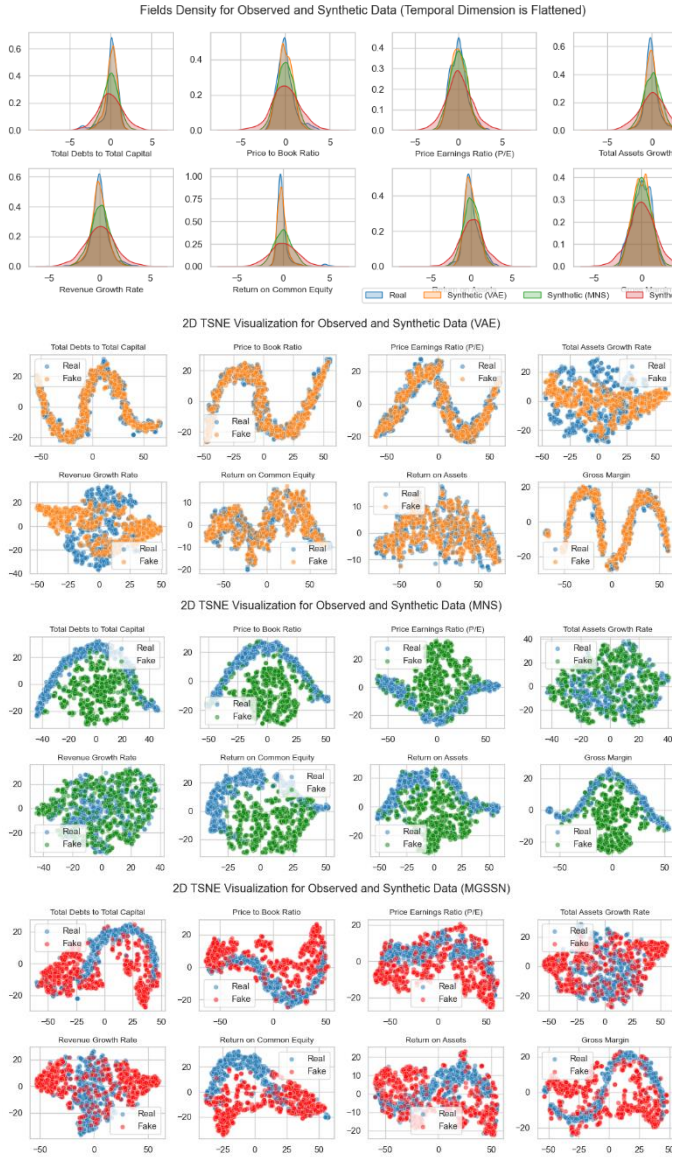

Figure 4: Synthetic sample fidelity.

TABLE IV. Metrics for discriminating between classifiers and models

Classifier / Model	VAE	MNS	MGSSM
Support Vector Classifier	0.3036	0.4232	0.4241
Gradient Boosting Machine	0.3304	0.4509	0.4777
Random Forest Classifier	0.3661	0.4732	0.4732
Max	0.3661	0.4732	0.4777
Mean	0.3333	0.4509	0.4583

3)Utility

Encouraged by the idea that "the bottleneck mechanism inherent to VAEs, connecting the encoder and decoder, functions to remove noise from the input, potentially enhancing subsequent tasks like prediction" (Desai et al., 2021) [2], we designed an experiment to predict next-quarter stock returns using both the original fundamental factors (the eight fields we selected) and their reconstructed counterparts. In contrast to the generative process, the reconstruction phase involves decoding the encoded $\mathbf{z}^{\text{mean}}{}_i$ to obtain $\mathbf{E}[\hat{\mathbf{X}}_i]$, resulting in reconstructed data that closely resemble the observations. Although there is no explicit mathematical grounding, some proponents suggest that the reconstructed samples represent denoised versions of the original data.

Specifically, we compared the forecasting abilities of the original and denoised factor exposures for the testing timeframe (2018Q2 to 2023Q1). Within each quarter of this timeframe, our procedure unfolded as follows. First, we trained an ordinary least squares (OLS) regressor using the factors from the previous quarter and the corresponding returns from the current quarter. Subsequently, we employed the trained regressor to predict returns for the subsequent quarter based on the factor exposures of the current quarter. Our evaluation involved comparing results obtained using both the original factor exposure data and the reconstructed factor exposure data. Evaluation metrics include the information coefficient (IC), information coefficient information ratio (ICIR), rank information coefficient (RankIC), and backtested cumulative returns from the strategy of holding the top 20% stocks with the highest predicted returns every quarter.[1] It is essential to note that the data used for evaluation are in-sample.

The subsequent table 5 and figure 4 display the performance outcomes. In essence, our experiment did not uncover evidence that reconstruction significantly enhances the efficacy of financial factors. All metrics indicate better predictability of the original data than the reconstructed data.

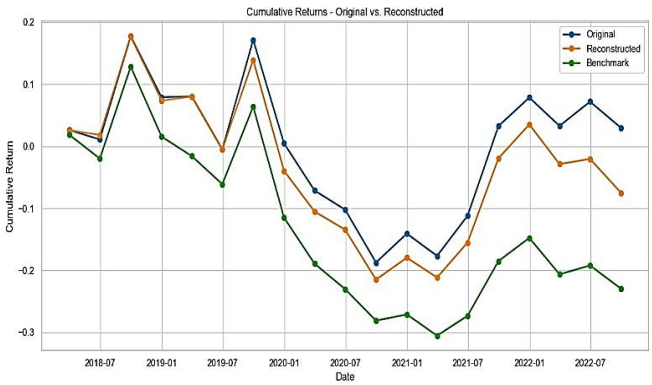

Figure 5: Cumulative return of factor-based stock sefactors v.s. reconstructed factors lection strategy, original factors v.s. reconstructed factors.

TABLE IV. Stock return productivity metrics, original factors v.s. reconstructed factors

Metrics / Factor	Original	Reconstructed
Average IC	0.1362	0.0960
Negative Percentage	21.05%	36.84%
Average RankIC	0.1404	0.0943
Negative Percentage	15.80%	31.58%
ICIR	0.7518	0.5585

B. Incorporating Conditional Features

Whether and how do one-year quarterly fundamental characteristics differ between firms that succeed and those that underperform in the following first quarter? To support this research, we add the Return Rank as a conditional label, allowing us to train a conditional VAE capable of producing synthetic samples suited to certain future performance conditions. The Return Rank is calculated by ranking each company's relative next-quarter stock return on a given date into three categories: **outperforming** (return in the top 20% among sector peers), **underperforming** (return in the bottom 20% among sector peers), and **neutral** (all other cases).

A preliminary look at the marginal distributions of each sample at different quarters offers interesting results. Figure 7 indicates that changes in basic factor patterns between cases of differing rankings are rather minor, consistent with the efficient market hypothesis. However, it is clear that outperforming synthetic stocks display far higher growth rates in both revenue and total assets than underperforming synthetic stocks. This divergence gets even more evident as we look at more recent historical times. This finding shows that current revenue and total asset growth rates may be useful markers for projecting next-quarter success.

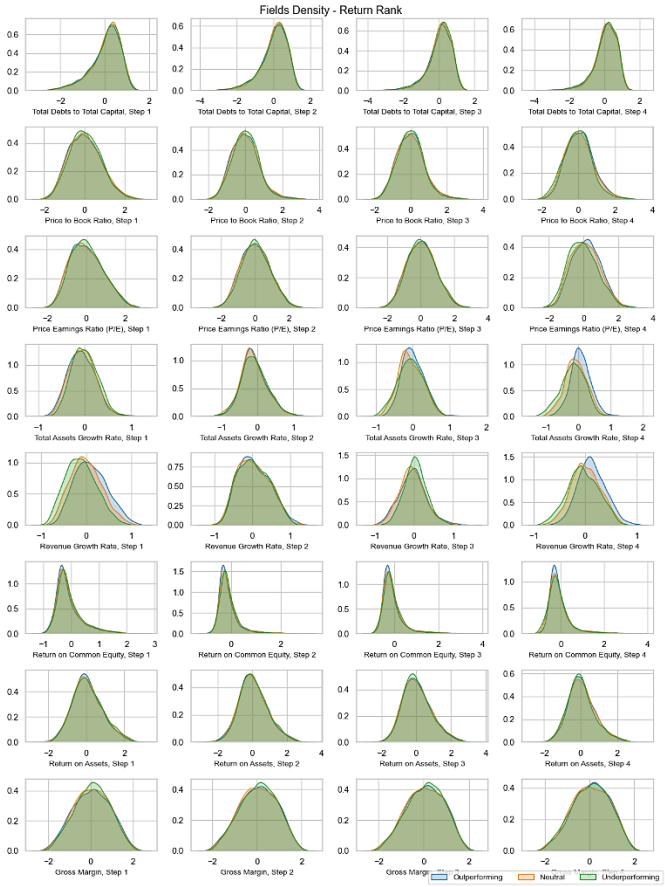

Figure 6: Plots showing distribution density across

C. Product

Finally, we finished the product development phase of our simulator by training two VAE and CVAE models for each of the nine sectors. This product allows customers to enter a sector, a certain year, and a return tier as inputs. The simulator calls the appropriate model (VAE when the return tier is undefined and CVAE when specified) for the selected sector. The algorithm randomly selects a labeled or unlabeled latent vector to create a (T ×D = 4×8) sample with a 4-quarter sequence of the eight financial variables. This sample is then re-transformed (including normalization and log retransformations) using the recorded means and standard deviation data table for the specified sector and year.

Our Synthetic Financial Data Generator has a user-friendly interface based on the Streamlit architecture. Users may view both the produced data table and a line graph representation of their single sample. The interface also allows you to produce and export a dataset with several samples in CSV format. This capability is very valuable for users who want to undertake more population characteristic research and need a collection of synthetic data.

V. Limitations And Future Work

A clear alternative to individually modeling each sector by training multiple models on sector-specific datasets is to develop a unified model using the entire dataset and incorporating the sector as a conditional channel. However, due to time constraints, we did not explore this approach or compare it with the aforementioned methodology.

Although we have evaluated one of the primary models, it is imperative to emphasize that future evaluations should encompass both the conditional and sector-specific models.

In summary, our study undertook a pertinent yet exploratory experiment aimed at replicating a subset of domains, utilizing a relatively modest network and a limited number of organizations. We incorporated a single conditional channel for control. While our model yielded acceptable results, it fell short in some aspects, including overconfidence and elevated discriminative scores. We anticipate that an expanded and refined model, trained on a larger dataset, will afford greater capability and yield more robust commercial applications and economic insights. This envisioned future endeavor aligns with our objective of enhancing the model's performance and broadening its scope of application.

VI. Conclusion

In this endeavour, we adopted a distinctive methodology to generate synthetic financial data, employing a temporal variational autoencoder (VAE) framework. Our model, designed to generate quarterly financial data sequences for a fictitious entity within a specific industry, fulfills various objectives, including data-driven engineering and market research. To capture the complex interplay of multivariate distributions and temporal dependencies inherent in financial time series data, our approach integrates recurrent neural networks (RNNs) within the VAE framework.

Our experimental findings for the primary model demonstrated its advantages over standard baseline models. Furthermore, the trials including sample creation based on particular future performance demonstrated our model's potential for market research.

VII. References

[1] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A model with recurrent latent variables for sequential data. Volume 28 of Advances in Neural Information Processing Systems (2015).

[2] Avi Desai, Corbin Freeman, Zeyu Wang, Ian Beaver. Timevae is a variational autoencoder that generates multivariate time series. preprint arXiv:2111.08095, 2021.

[3] Chris Donahue, Julian McAuley, and Miller Puckette. Audio synthesis with adversarial properties. arXiv preprint arXiv:1802.04208, 2018..

[4] Claudio Esteban, Stephanie L Hyland, and Gunnar Ratsch. Real-valued (medical) time series production using recurrent conditional gans. arXiv preprint 1706.02633, 2017.

[5] Otto Fabius and Joost R. Van Amersfoort. Variational recurrent autoencoders. arXiv preprint 1412.6581, 2014.

[6] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-vae: Learning fundamental visual ideas using a limited variational framework. In the International Conference on Learning Representations, November 2016.

[7] Diederik P Kingma & Max Welling. Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114 (2013)..

[8] Olof Mogren. C-rnn-gan stands for Continuous Recurrent Neural Networks with Adversarial Training. arXiv preprint arXiv:1611.09904 (2016)..

[9] Yoon, David Jarrett, Mihaela Van der Schaar. Generative adversarial networks using time series data. Advances in neural information processing systems, volume 32 2019.